Course Organization and Introduction

Today’s Agenda

Intended topics for today’s session

- FSoC Laboratory Project Presentation
- The Secure Hash Algorithm 1 (SHA-1) ...
- Lab Organization
- The FSoC Design Challenge
- Recommended Readings

FPGA-based SoC Design

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob

University of Applied Sciences Darmstadt

h_da

Faculty of Electrical Engineering and Information Technology

fbeit

Course Organization and Introduction

Today’s Agenda

Intended topics for today’s session

- The FSoC laboratory project offers the possibility to directly apply the theoretical knowledge gained in lectures within a practical context.
- Continuous work on a given project with gradually increasing complexity provides optimal exam preparation.
- The chronological order of the project milestones and assignments is closely linked with the respective lecture topics.

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

FSoC Laboratory WS22/23

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob

University of Applied Sciences Darmstadt

h_da

Faculty of Electrical Engineering and Information Technology

fbeit

FSoC Lab WS22/23

- The lab focuses on teaching practical skills related to FPGA based SoC design using C and SystemVerilog:
  - Design and implementation of custom hardware accelerators and ISA extensions.
  - HW/SW integration of custom accelerators into existing FPGA based SoC architectures followed by profiling and benchmarking of the respective solutions.
- Students can work in as a team of two. Group registration is accomplished with the submission of the first assignment.

Hardware Accelerators and Heterogeneous Computing – A few Definitions …

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Hardware Accelerators and Heterogeneous Computing

- General-purpose CPUs implement a fixed set of instructions, able to execute nearly any kind of computation.
- This great flexibility offered by a CPU based computing platform also represents one of its major drawbacks.
- The sequential execution of algorithms, often characterized as ‘temporal computing’ allows the implementation of almost all existing algorithms, however, not always with their best potential performance.

Hardware Accelerators and Heterogeneous Computing

- Wikipedia: Hardware acceleration is the use of computer hardware designed to perform specific functions more efficiently when compared to software running on a general-purpose central processing unit (CPU).
- The slowdown and the nearing end of transistor scaling play a vital role in the rise of domain-specific architectures.
- Domain-specific architectures are based on “accelerators”: Specialized hardware structures designed to efficiently execute dominating computation patterns and improve performance under a given transistor budget.

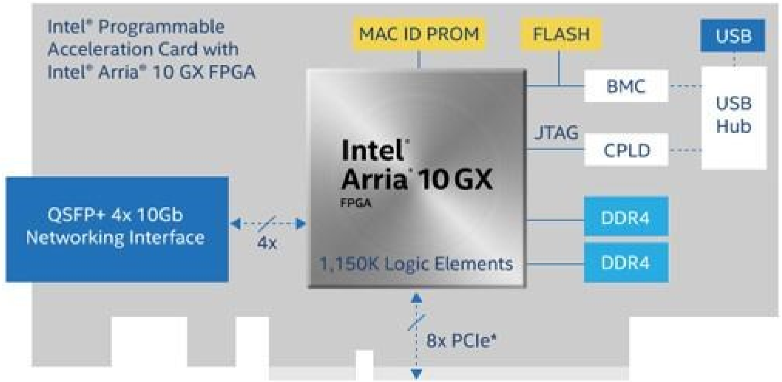
Hardware Accelerators and Heterogeneous Computing

- Specialized computing units have become common place in modern computer architectures

PCI Express (PCIe) accelerator card
  - Uses Arria 10, Stratix 10 field-programmable gate array (FPGA)

FPGAs implement custom hardware to accelerate software
  - Especially effective for energy-efficient acceleration

PAC provides FPGA with on-board DDR4 RAM
  - Can also access host processor's main memory
  - Provides networking interface for low-latency networking



Design and Implementation of a SHA-1 Co-Processor

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Overview

- Our intention is to design an FPGA based Embedded System similar to the ones described before: The task is to analyze, design and implement a cryptographic hash function (SHA-1) in hardware (SystemVerilog) and software (C/C++ bare-metal on the Intel Nios II softcore processor or on the ARM Cortex A9 running Linux).
- The hardware unit finally acts as dedicated accelerator placed in the peripheral-set of the Nios II CPU to accelerate the computation of the SHA-1 algorithm.
- It is almost entirely about coding. The focus is set on well documented source code including benchmark numbers related to resource requirements, throughput, latency and power consumption …
- There will be open-lab sessions throughout the semester that can be used to discuss your ideas, implementations and findings.

The Secure Hash Algorithm 1 (SHA-1)

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

The Secure Hash Algorithm 1 (SHA-1)

- SHA was designed by the National Institute of Standards and Technology (NIST) and is the US federal standard for hash functions, specified in FIPS-180 (1993).
- SHA-1, revised version of SHA, specified in FIPS-180-1 (1995) use with Secure Hash Algorithm). It produces 160-bit hash values.
- Applications: Hash functions are widely used for rapid data lookup, password storage, data integrity and authentication checks.
- Since 2005 SHA-1 has not been considered secure against well-funded opponents. Since 2010 many organizations have recommended its replacement by SHA-2 or SHA-3.
- Thursday, 23 February 2017, researchers at the Dutch research institute CWI and Google jointly announce that they have broken the SHA-1 internet security standard in practice, , publishing two dissimilar PDF files which produced the same SHA-1 hash. We have broken SHA-1 in practice: https://shattered.io/

The Secure Hash Algorithm 1 (SHA-1)

- Just a simple function ...

Message (arbitrary length)

FSoC 2022 is fun!

→ SHA-1 Function →

Hash Value (fixed length)

`cb419b01c6c5402e0153b17785d899eaffb59d94`

160 bit

- The SHA-1 algorithm on the left is a one-way function that transforms an arbitrary-length message into a 160-bit hash value (fixed length digest).
- It is very easy to compute the hash for a given input. The other way around is extremely difficult (or practically impossible).
- Cryptographic hash values are sometimes referred to as digital fingerprints.

The Secure Hash Algorithm 1 (SHA-1)

```
student@FSoC-edasys:~$ sudo apt-get install openssl
```

[Linux console]: Install the openSSL toolset under Ubuntu Linux ...

```
student@FSoC-edasys:~$ echo -n "FSoC 2022 is fun!" | openssl sha1
(stdin)= cb419b01c6c5402e0153b17785d899eaffb59d94
```

[Linux console]: SHA-1 Hash computation – String Entry

```
student@FSoC-edasys:~$ echo -n "FSoC 2022 is fun!" > sha1_input.txt
student@FSoC-edasys:~$ cat sha1_input.txt && echo
FSoC 2021 is fun!
student@FSoC-edasys:~$ sha1sum sha1_input.txt
cb419b01c6c5402e0153b17785d899eaffb59d94  sha1_input.txt
student@FSoC-edasys:~$
```

[Linux console]: SHA-1 Hash computation – File Entry

SHA-1 - The Algorithm in detail ...

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob

University of Applied Sciences Darmstadt

h_da

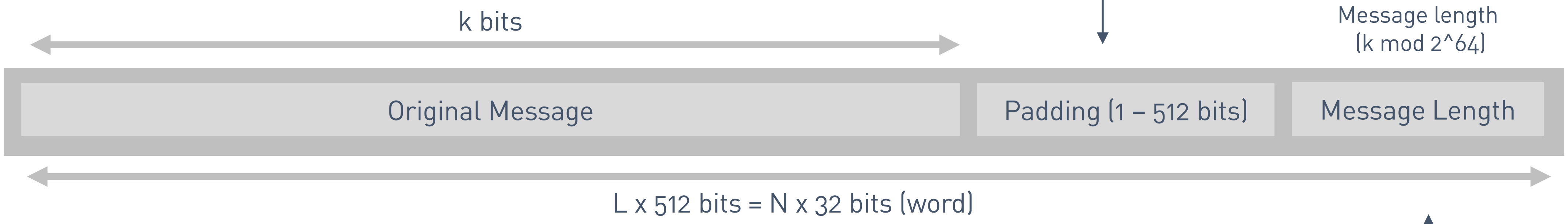
Faculty of Electrical Engineering and Information Technology

fbeit

Initial Preprocessing

Preparation 1: **Append Padding Bits**: Message is "padded" with a **single binary 1** and as many **0's** as necessary to bring the message length to 64 bits fewer than an even multiple of 512:

(512 bits – 64 bits = 448 bits)



Preparation 2: **Append the Message Length**: 64 bits are appended to the end of the padded message.
These bits hold the binary format of 64 bits indicating the length of the original message.

- Before the actual computation starts, the algorithm has to preprocess the message.

Initial Preprocessing - Example

Taken from: Paar, C. & Pelzl, J.: Understanding Cryptography. A Textbook for Students and Practitioners, pp. 308-309, Springer, 2009.

- Given is the message "abc" consisting of three 8-bit ASCII characters with a total length of l = 24 bits:

01100001	01100010	01100011
a	b	c

- We append a "1" followed by k = 423 zero bits, where k is determined by

$$\text{Number of Zeros: } k \equiv 448 - (\text{Message Length } l + 1) = 448 - 25 = \mathbf{423} \bmod 512$$

- Finally, we append the 64-bit value which contains the binary representation of the length *l* = 24(dec) = 11000(bin). The padded message is then given by:

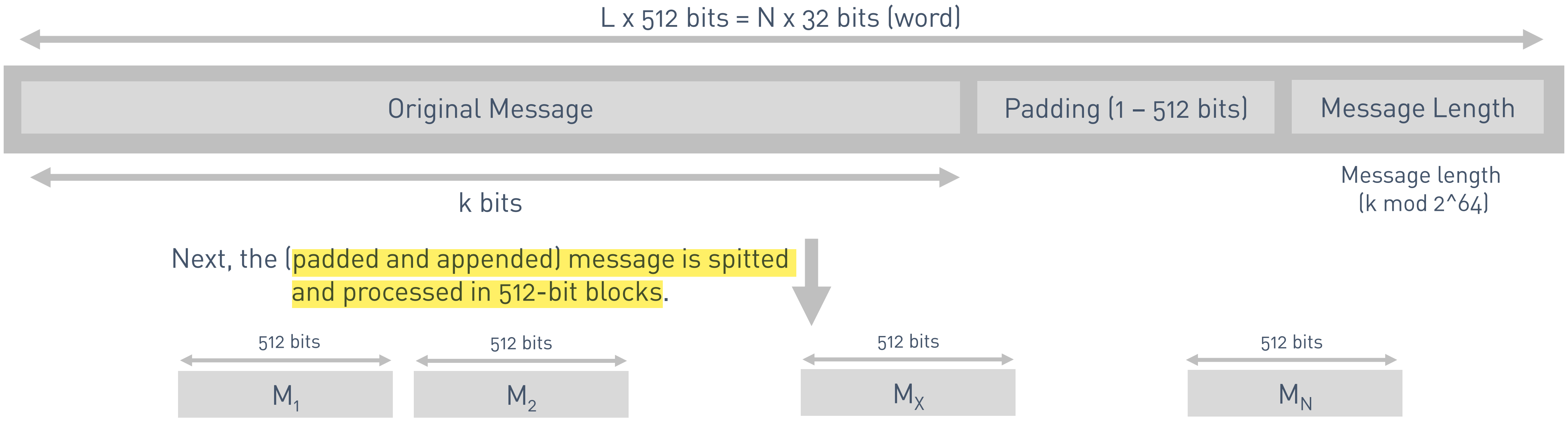
8-bit	8-bit	8-bit	1-bit	423-bit	64-bit	
01100001	01100010	01100011	1	00 ...0	00 ... 011000	l = 24 (bits)
a	b	c				

Message Length l + 1

Initial Preprocessing - Homework

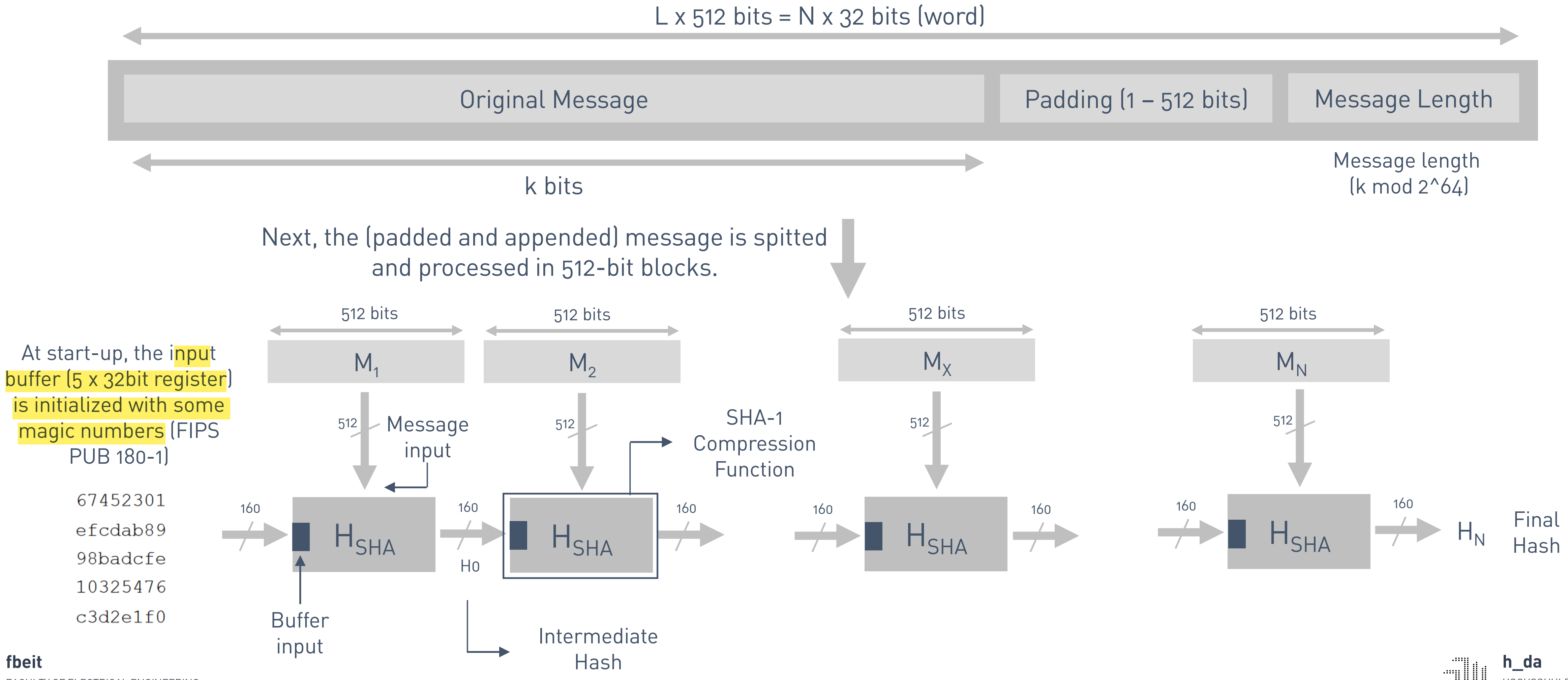
- **Summary**: The actual message is padded by appending a single 1, followed by 0 bits until the message has a length of 448 bits in total. Next, we represent the length of the original message as a 64 bit number and append this the previous 448 bit frame, producing a message that is 512 bits long.
- Prepare the following message for SHA-1 processing: 0x6162636465

Initial Preprocessing

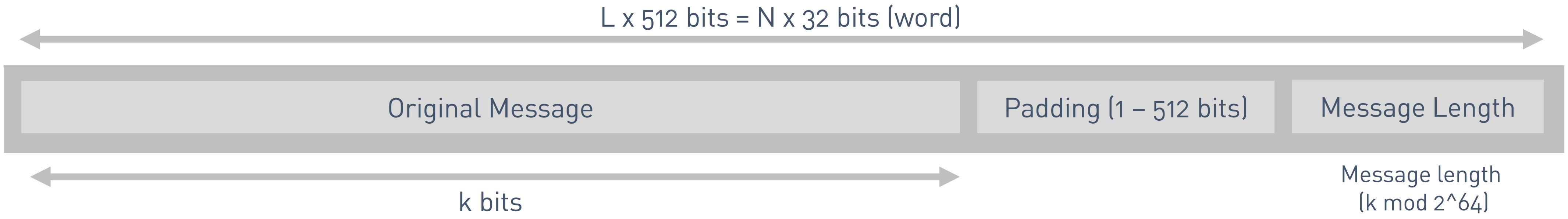


- During the actual SHA-1 hash computation, the compression function processes the message in 512-bit chunks.

Basic SHA-1 Structure



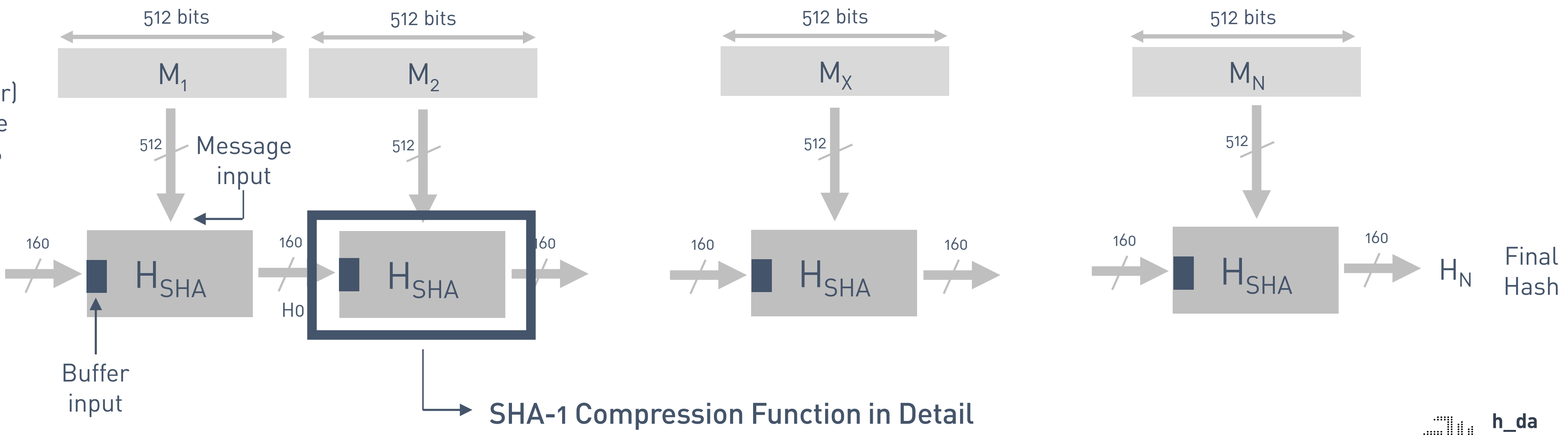
Basic SHA-1 Structure



Next, the (padded and appended) message is spitted and processed in 512-bit blocks.

At start-up, the input buffer (5 x 32bit register) is initialized with some magic numbers (FIPS PUB 180-1)

```
67452301
efcdab89
98badcfe
10325476
c3d2e1f0
```



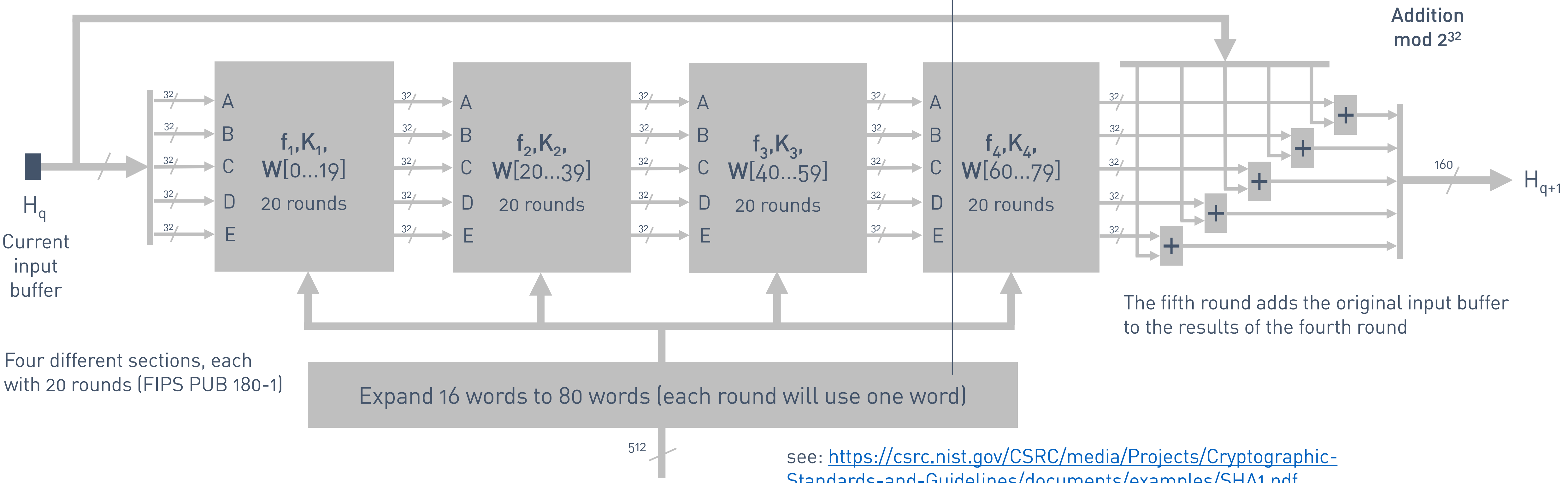
SHA-1 Compression Function

The 512 bit wide input message is mapped on the first 16 **W** words

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

For the remaining rounds 16 to 79, the W words are computed based on previous W words.

The compression function consists **of 80 rounds** which are divided into four stages of 20 rounds each (FIPS PUB 180-1)



Current message block

The fifth round adds the original input buffer to the results of the fourth round

Four different sections, each with 20 rounds (FIPS PUB 180-1)

see: https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/SHA1.pdf for further details …

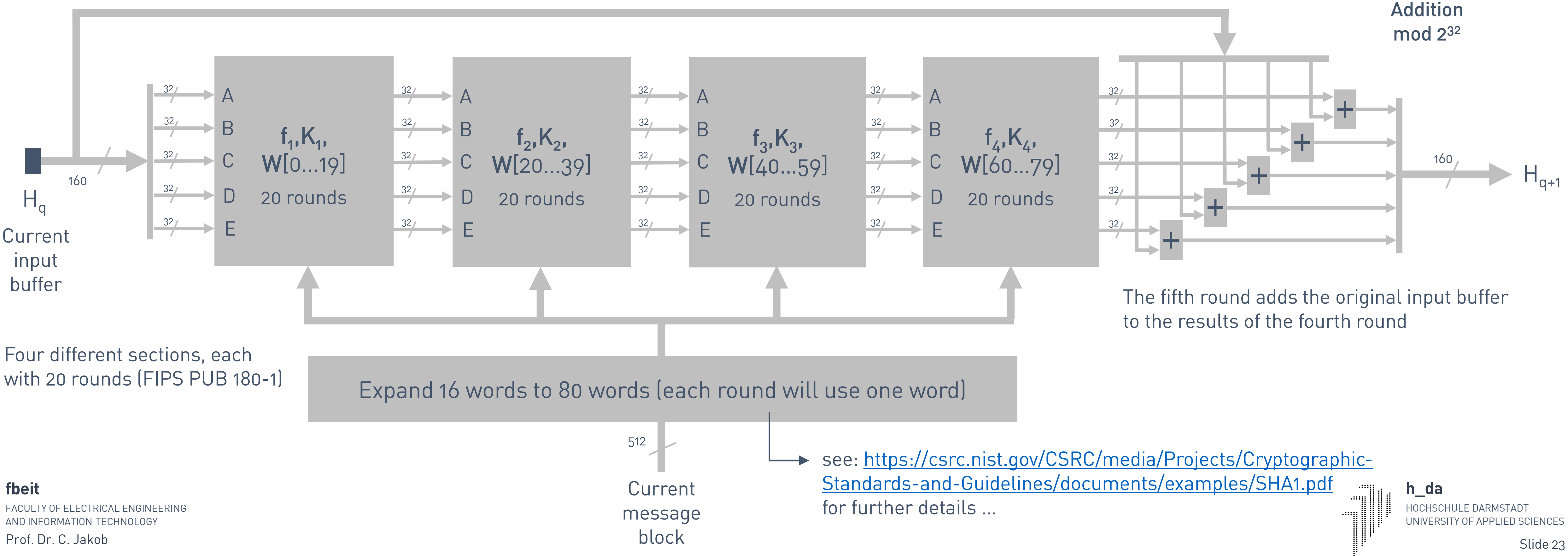
SHA-1 Compression Function

SHA-1 Functions

$$f_t(x, y, z) = \begin{cases} Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x, y, z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x, y, z) = x \oplus y \oplus z & 60 \le t \le 79. \end{cases}$$

SHA-1 Constants

$$K_t = \begin{cases} \texttt{5a827999} & 0 \le t \le 19 \\ \texttt{6ed9eba1} & 20 \le t \le 39 \\ \texttt{8f1bbcdc} & 40 \le t \le 59 \\ \texttt{ca62c1d6} & 60 \le t \le 79 \end{cases}$$



The fifth round adds the original input buffer to the results of the fourth round

Four different sections, each with 20 rounds (FIPS PUB 180-1)

see: https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/SHA1.pdf for further details …

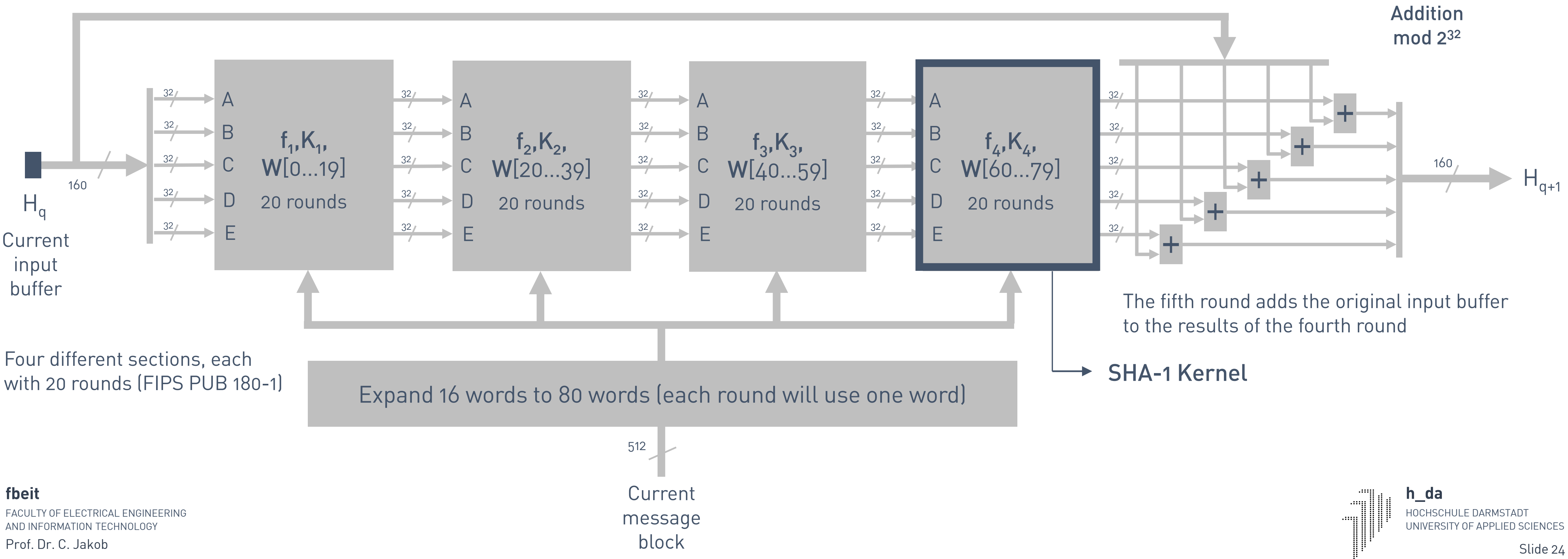
SHA-1 Compression Function

SHA-1 Functions

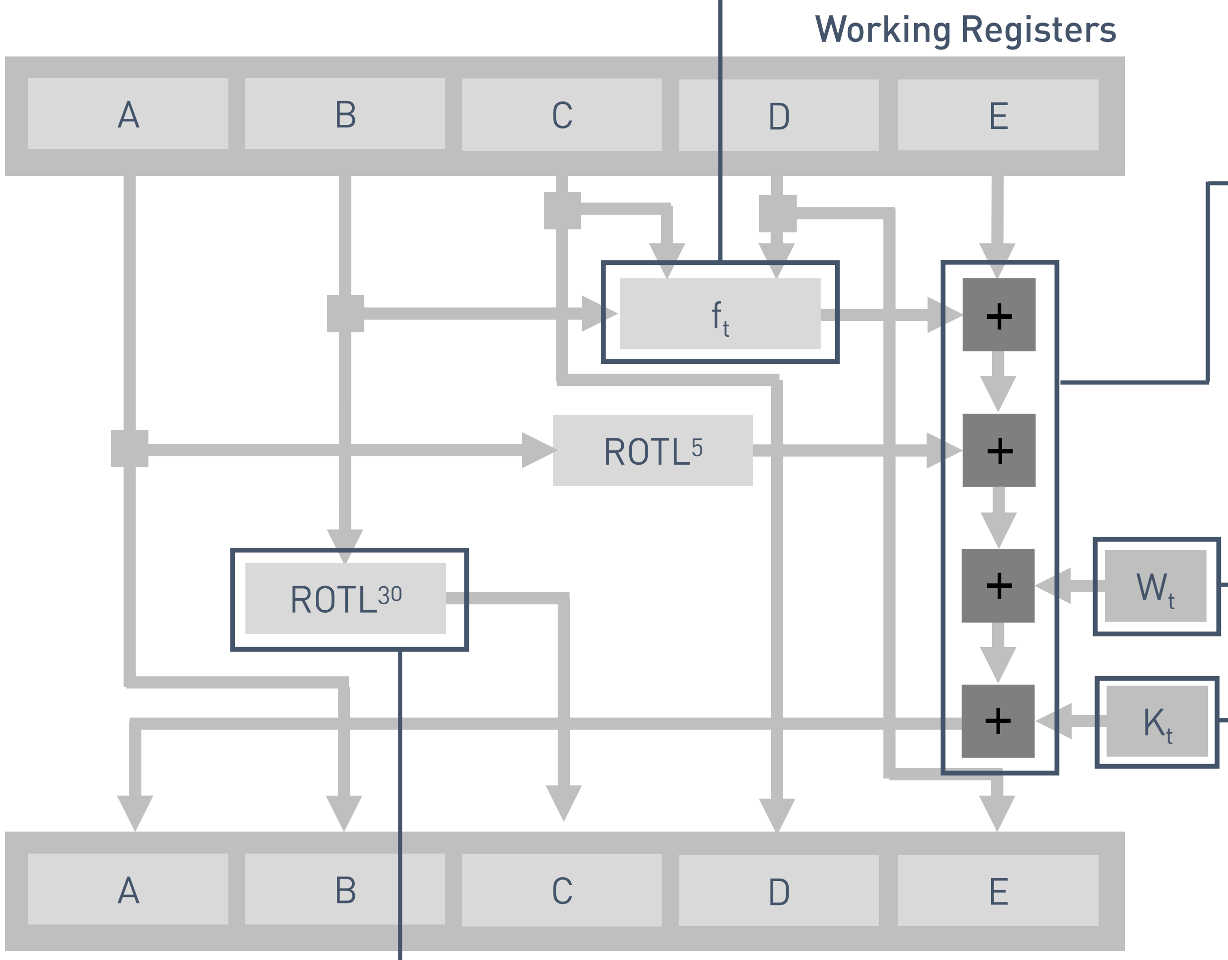
$$f_t(x,y,z) = \begin{cases} Ch(x,y,z)=(x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z)=x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z)=(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z)=x \oplus y \oplus z & 60 \le t \le 79. \end{cases}$$

SHA-1 Constants

$$K_t = \begin{cases} \texttt{5a827999} & 0 \le t \le 19 \\ \texttt{6ed9eba1} & 20 \le t \le 39 \\ \texttt{8f1bbcdc} & 40 \le t \le 59 \\ \texttt{ca62c1d6} & 60 \le t \le 79 \end{cases}$$



SHA-1 Kernel



$$f_t(x, y, z) = \begin{cases} Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x, y, z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x, y, z) = x \oplus y \oplus z & 60 \le t \le 79 \end{cases}$$

Addition (+) is performed modulo 2^{32}

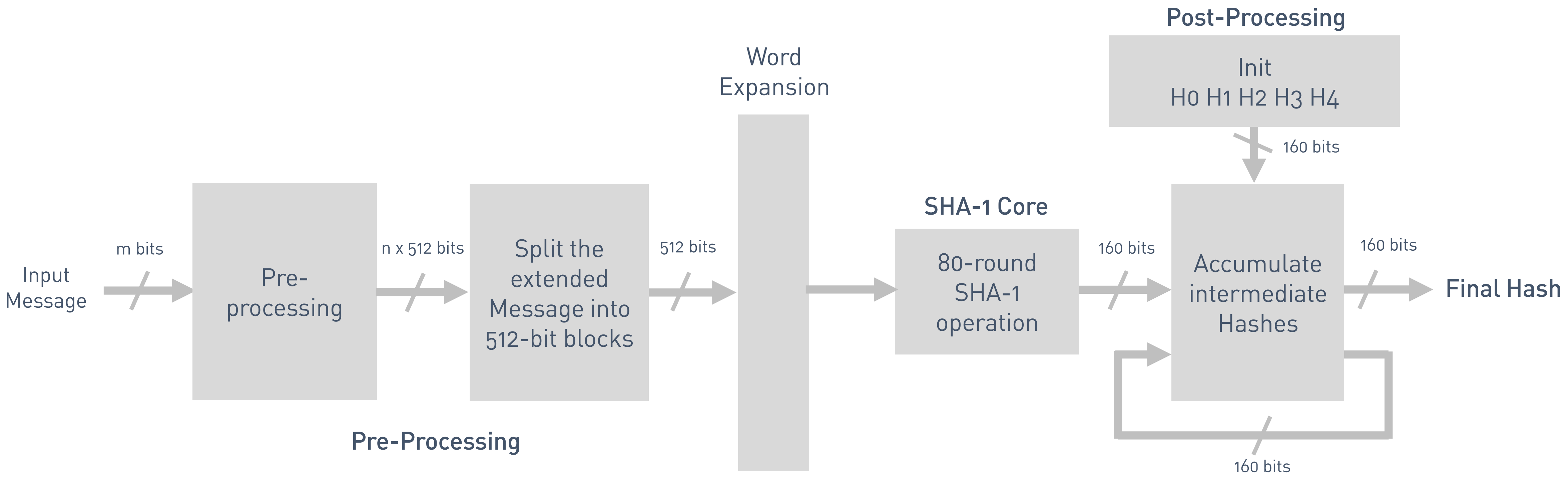
$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \le t \le 79 \end{cases}$$

$$K_t = \begin{cases} \texttt{5a827999} & 0 \le t \le 19 \\ \texttt{6ed9eba1} & 20 \le t \le 39 \\ \texttt{8f1bbcdc} & 40 \le t \le 59 \\ \texttt{ca62c1d6} & 60 \le t \le 79 \end{cases}$$

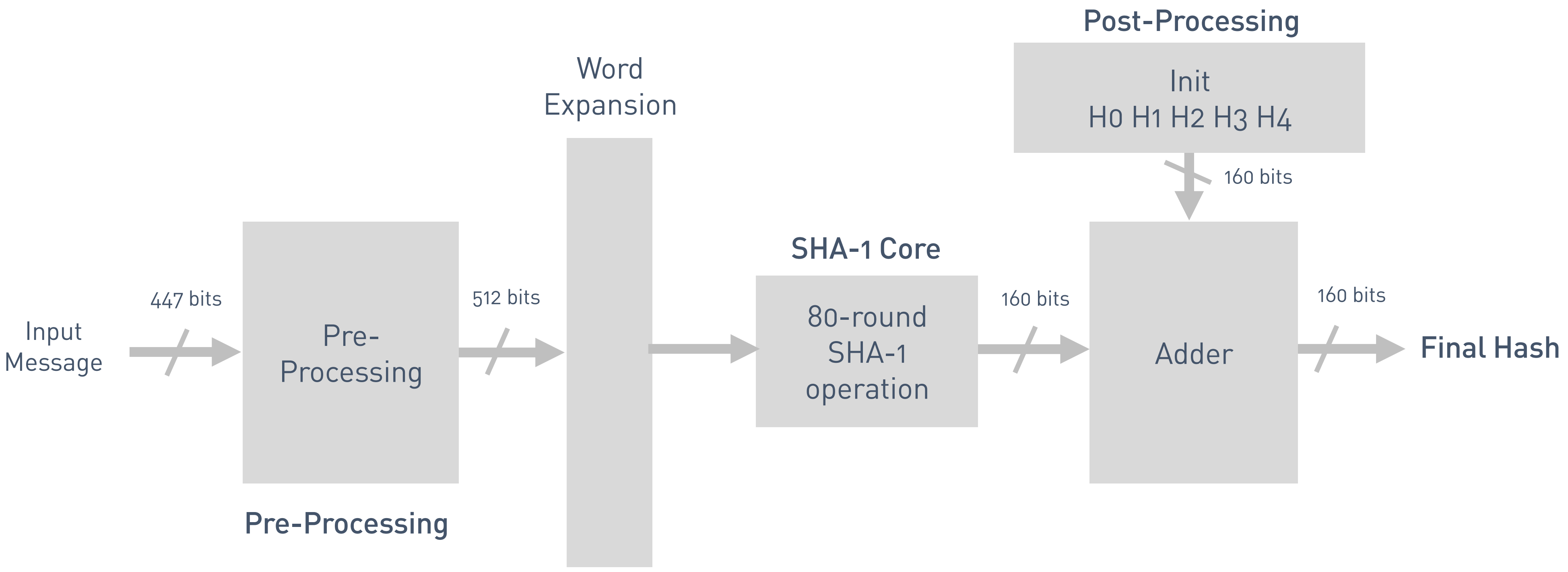
$$ROTL^n(x) = (x \ll n) \vee (x \gg w - n).$$

Thus, $ROTL^n(x)$ is equivalent to a circular shift (rotation) of x by n positions to the left.

Simplified Block Diagram



Simplified Block Diagram



We restrict our HW/SE implementations on that part of the algorithm, so we restrict the computation to input messages smaller than 448 bits ...

SHA-1 - Verification

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

How to verify your implementation?

Taken from: NIST, Cryptographic Standards and Guidelines, https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/SHA1.pdf

One Block Message Sample

Input Message: "abc"

Input Message (in hex)

```
61626380
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000018
```

‘1’

Message Length: 3 byte = 24 bit (0x18)

Initial hash value:

```
H[0] = 67452301
H[1] = EFCDAB89
H[2] = 98BADCFE
H[3] = 10325476
H[4] = C3D2E1F0
```

	A	B	C	D	E
t= 0:	0116FC33	67452301	7BF36AE2	98BADCFE	10325476
t= 1:	8990536D	0116FC33	59D148C0	7BF36AE2	98BADCFE
t= 2:	A1390F08	8990536D	C045BF0C	59D148C0	7BF36AE2
t= 3:	CDD8E11B	A1390F08	626414DB	C045BF0C	59D148C0
t= 4:	CFD499DE	CDD8E11B	284E43C2	626414DB	C045BF0C
t= 5:	3FC7CA40	CFD499DE	F3763846	284E43C2	626414DB
t= 6:	993E30C1	3FC7CA40	B3F52677	F3763846	284E43C2
t= 7:	9E8C07D4	993E30C1	0FF1F290	B3F52677	F3763846
t= 8:	4B6AE328	9E8C07D4	664F8C30	0FF1F290	B3F52677
t= 9:	8351F929	4B6AE328	27A301F5	664F8C30	0FF1F290
t=10:	FBDA9E89	8351F929	12DAB8CA	27A301F5	664F8C30
t=11:	63188FE4	FBDA9E89	60D47E4A	12DAB8CA	27A301F5
t=12:	4607B664	63188FE4	7EF6A7A2	60D47E4A	12DAB8CA
t=13:	9128F695	4607B664	18C623F9	7EF6A7A2	60D47E4A
t=14:	196BEE77	9128F695	1181ED99	18C623F9	7EF6A7A2
t=15:	20BDD62F	196BEE77	644A3DA5	1181ED99	18C623F9
t=16:	4E925823	20BDD62F	C65AFB9D	644A3DA5	1181ED99
t=17:	82AA6728	4E925823	C82F758B	C65AFB9D	644A3DA5
t=18:	DC64901D	82AA6728	D3A49608	C82F758B	C65AFB9D
t=19:	FD9E1D7D	DC64901D	20AA99CA	D3A49608	C82F758B
t=20:	1A37B0CA	FD9E1D7D	77192407	20AA99CA	D3A49608
t=21:	33A23BFC	1A37B0CA	7F67875F	77192407	20AA99CA
t=22:	21283486	33A23BFC	868DEC32	7F67875F	77192407
t=23:	D541F12D	21283486	0CE88EFF	868DEC32	7F67875F
t=24:	C7567DC6	D541F12D	884A0D21	0CE88EFF	868DEC32
t=25:	48413BA4	C7567DC6	75507C4B	884A0D21	0CE88EFF
t=26:	BE35FBD5	48413BA4	B1D59F71	75507C4B	884A0D21
t=27:	4AA84D97	BE35FBD5	12104EE9	B1D59F71	75507C4B
t=28:	8370B52E	4AA84D97	6F8D7EF5	12104EE9	B1D59F71
t=29:	C5FBAF5D	8370B52E	D2AA1365	6F8D7EF5	12104EE9
t=30:	1267B407	C5FBAF5D	A0DC2D4B	D2AA1365	6F8D7EF5
t=31:	3B845D33	1267B407	717EEBD7	A0DC2D4B	D2AA1365
t=32:	046FAA0A	3B845D33	C499ED01	717EEBD7	A0DC2D4B
t=33:	2C0EBC11	046FAA0A	CEE1174C	C499ED01	717EEBD7
t=34:	21796AD4	2C0EBC11	811BEA82	CEE1174C	C499ED01
t=35:	DCBBB0CB	21796AD4	4B03AF04	811BEA82	CEE1174C
t=36:	0F511FD8	DCBBB0CB	085E5AB5	4B03AF04	811BEA82
t=37:	DC63973F	0F511FD8	F72EEC32	085E5AB5	4B03AF04
t=38:	4C986405	DC63973F	03D447F6	F72EEC32	085E5AB5
t=39:	32DE1CBA	4C986405	F718E5CF	03D447F6	F72EEC32
t=40:	FC87DEDF	32DE1CBA	53261901	F718E5CF	03D447F6
t=41:	970A0D5C	FC87DEDF	8CB7872E	53261901	F718E5CF
t=42:	7F193DC5	970A0D5C	FF21F7B7	8CB7872E	53261901
t=43:	EE1B1AAF	7F193DC5	25C28357	FF21F7B7	8CB7872E
t=44:	40F28E09	EE1B1AAF	5FC64F71	25C28357	FF21F7B7
t=45:	1C51E1F2	40F28E09	FB86C6AB	5FC64F71	25C28357
t=46:	A01B846C	1C51E1F2	503CA382	FB86C6AB	5FC64F71
t=47:	BEAD02CA	A01B846C	8714787C	503CA382	FB86C6AB
t=48:	BAF39337	BEAD02CA	2806E11B	8714787C	503CA382
t=49:	120731C5	BAF39337	AFAB40B2	2806E11B	8714787C
t=50:	641DB2CE	120731C5	EEBCE4CD	AFAB40B2	2806E11B

How to verify your implementation?

Taken from: NIST, Cryptographic Standards and Guidelines, https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/SHA1.pdf

t=51:	3847AD66	641DB2CE	4481CC71	EEBCE4CD	AFAB40B2
t=52:	E490436D	3847AD66	99076CB3	4481CC71	EEBCE4CD
t=53:	27E9F1D8	E490436D	8E11EB59	99076CB3	4481CC71
t=54:	7B71F76D	27E9F1D8	792410DB	8E11EB59	99076CB3
t=55:	5E6456AF	7B71F76D	09FA7C76	792410DB	8E11EB59
t=56:	C846093F	5E6456AF	5EDC7DDB	09FA7C76	792410DB
t=57:	D262FF50	C846093F	D79915AB	5EDC7DDB	09FA7C76
t=58:	09D785FD	D262FF50	F211824F	D79915AB	5EDC7DDB
t=59:	3F52DE5A	09D785FD	3498BFD4	F211824F	D79915AB
t=60:	D756C147	3F52DE5A	4275E17F	3498BFD4	F211824F
t=61:	548C9CB2	D756C147	8FD4B796	4275E17F	3498BFD4
t=62:	B66C020B	548C9CB2	F5D5B051	8FD4B796	4275E17F
t=63:	6B61C9E1	B66C020B	9523272C	F5D5B051	8FD4B796
t=64:	19DFA7AC	6B61C9E1	ED9B0082	9523272C	F5D5B051
t=65:	101655F9	19DFA7AC	5AD87278	ED9B0082	9523272C
t=66:	0C3DF2B4	101655F9	0677E9EB	5AD87278	ED9B0082
t=67:	78DD4D2B	0C3DF2B4	4405957E	0677E9EB	5AD87278
t=68:	497093C0	78DD4D2B	030F7CAD	4405957E	0677E9EB
t=69:	3F2588C2	497093C0	DE37534A	030F7CAD	4405957E
t=70:	C199F8C7	3F2588C2	125C24F0	DE37534A	030F7CAD
t=71:	39859DE7	C199F8C7	8FC96230	125C24F0	DE37534A
t=72:	EDB42DE4	39859DE7	F0667E31	8FC96230	125C24F0
t=73:	11793F6F	EDB42DE4	CE616779	F0667E31	8FC96230
t=74:	5EE76897	11793F6F	3B6D0B79	CE616779	F0667E31
t=75:	63F7DAB7	5EE76897	C45E4FDB	3B6D0B79	CE616779
t=76:	A079B7D9	63F7DAB7	D7B9DA25	C45E4FDB	3B6D0B79
t=77:	860D21CC	A079B7D9	D8FDF6AD	D7B9DA25	C45E4FDB
t=78:	5738D5E1	860D21CC	681E6DF6	D8FDF6AD	D7B9DA25
t=79:	42541B35	5738D5E1	21834873	681E6DF6	D8FDF6AD

```
H[0] = 67452301 + 42541B35
H[1] = EFCDAB89 + 5738D5E1
H[2] = 98BADCFE + 21834873
H[3] = 10325476 + 681E6DF6
H[4] = C3D2E1F0 + D8FDF6AD

Initial hash value:
 H[0] = 67452301
 H[1] = EFCDAB89
 H[2] = 98BADCFE
 H[3] = 10325476
 H[4] = C3D2E1F0
```

How to verify your implementation?

Taken from: NIST, Cryptographic Standards and Guidelines, https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/SHA1.pdf

t=51:	3847AD66	641DB2CE	4481CC71	EEBCE4CD	AFAB40B2
t=52:	E490436D	3847AD66	99076CB3	4481CC71	EEBCE4CD
t=53:	27E9F1D8	E490436D	8E11EB59	99076CB3	4481CC71
t=54:	7B71F76D	27E9F1D8	792410DB	8E11EB59	99076CB3
t=55:	5E6456AF	7B71F76D	09FA7C76	792410DB	8E11EB59
t=56:	C846093F	5E6456AF	5EDC7DDB	09FA7C76	792410DB
t=57:	D262FF50	C846093F	D79915AB	5EDC7DDB	09FA7C76
t=58:	09D785FD	D262FF50	F211824F	D79915AB	5EDC7DDB
t=59:	3F52DE5A	09D785FD	3498BFD4	F211824F	D79915AB
t=60:	D756C147	3F52DE5A	4275E17F	3498BFD4	F211824F
t=61:	548C9CB2	D756C147	8FD4B796	4275E17F	3498BFD4
t=62:	B66C020B	548C9CB2	F5D5B051	8FD4B796	4275E17F
t=63:	6B61C9E1	B66C020B	9523272C	F5D5B051	8FD4B796
t=64:	19DFA7AC	6B61C9E1	ED9B0082	9523272C	F5D5B051
t=65:	101655F9	19DFA7AC	5AD87278	ED9B0082	9523272C
t=66:	0C3DF2B4	101655F9	0677E9EB	5AD87278	ED9B0082
t=67:	78DD4D2B	0C3DF2B4	4405957E	0677E9EB	5AD87278
t=68:	497093C0	78DD4D2B	030F7CAD	4405957E	0677E9EB
t=69:	3F2588C2	497093C0	DE37534A	030F7CAD	4405957E
t=70:	C199F8C7	3F2588C2	125C24F0	DE37534A	030F7CAD
t=71:	39859DE7	C199F8C7	8FC96230	125C24F0	DE37534A
t=72:	EDB42DE4	39859DE7	F0667E31	8FC96230	125C24F0
t=73:	11793F6F	EDB42DE4	CE616779	F0667E31	8FC96230
t=74:	5EE76897	11793F6F	3B6D0B79	CE616779	F0667E31
t=75:	63F7DAB7	5EE76897	C45E4FDB	3B6D0B79	CE616779
t=76:	A079B7D9	63F7DAB7	D7B9DA25	C45E4FDB	3B6D0B79
t=77:	860D21CC	A079B7D9	D8FDF6AD	D7B9DA25	C45E4FDB
t=78:	5738D5E1	860D21CC	681E6DF6	D8FDF6AD	D7B9DA25
t=79:	42541B35	5738D5E1	21834873	681E6DF6	D8FDF6AD

```
H[0] = 67452301 + 42541B35 = A9993E36
H[1] = EFCDAB89 + 5738D5E1 = 4706816A
H[2] = 98BADCFE + 21834873 = BA3E2571
H[3] = 10325476 + 681E6DF6 = 7850C26C
H[4] = C3D2E1F0 + D8FDF6AD = 9CD0D89D
```

```
Initial hash value:
 H[0] = 67452301
 H[1] = EFCDAB89
 H[2] = 98BADCFE
 H[3] = 10325476
 H[4] = C3D2E1F0
```

Final hash

A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

SHA-1 - Applications

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

SHA-1 Applications on Wikipedia

Cryptography [edit]

Further information: Cryptographic hash function § Applications

SHA-1 forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec. Those applications can also use MD5; both MD5 and SHA-1 are descended from MD4.

Data integrity [edit]

Revision control systems such as Git, Mercurial, and Monotone use SHA-1 not for security but to identify revisions and to ensure that the data has not changed due to accidental corruption. Linus Torvalds said about Git:

> If you have disk corruption, if you have DRAM corruption, if you have any kind of problems at all, Git will notice them. It's not a question of *if*, it's a guarantee. You can have people who try to be malicious. They won't succeed. ... Nobody has been able to break SHA-1, but the point is the SHA-1, as far as Git is concerned, isn't even a security feature. It's purely a consistency check. The security parts are elsewhere, so a lot of people assume that since Git uses SHA-1 and SHA-1 is used for cryptographically secure stuff, they think that, Okay, it's a huge security feature. It has nothing at all to do with security, it's just the best hash you can get.
> ...
> I guarantee you, if you put your data in Git, you can trust the fact that five years later, after it was converted from your hard disk to DVD to whatever new technology and you copied it along, five years later you can verify that the data you get back out is the exact same data you put in. ...
> One of the reasons I care is for the kernel, we had a break in on one of the BitKeeper sites where people tried to corrupt the kernel source code repositories.[22]

However Git does not require the second preimage resistance of SHA-1 as a security feature, since it will always prefer to keep the earliest version of an object in case of collision, preventing an attacker from surreptitiously overwriting files.[23]

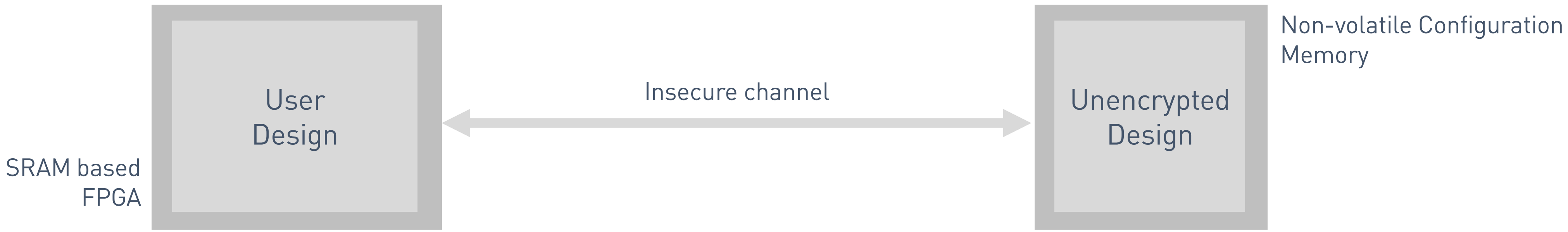
SHA-1 based Challenge and Response System Authentication

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

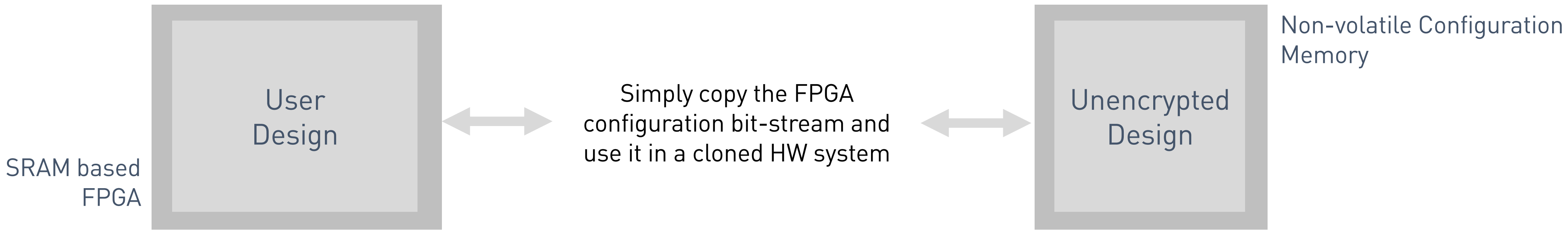
SHA-1 based Challenge and Response System Authentication

- **Problem**: SRAM based FPGA are volatile devices and loose their configuration with turning off the power supply. Therefore, the respective configuration is stored in an external non-volatile memory and loaded into the FPGA at every system start.



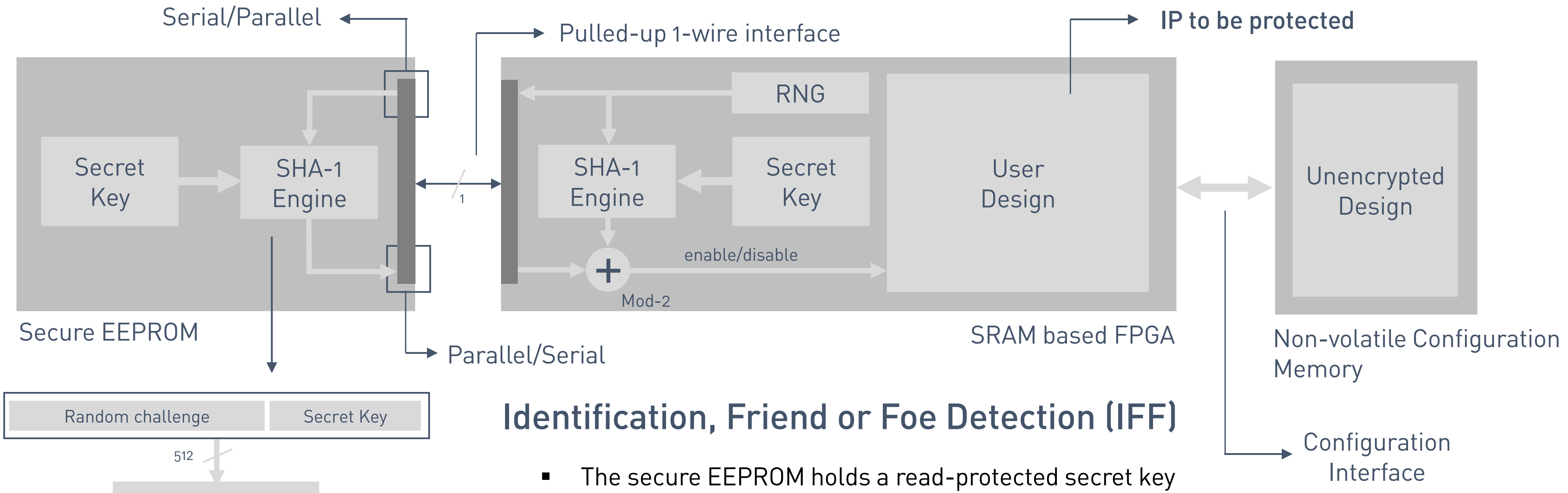
SHA-1 based Challenge and Response System Authentication

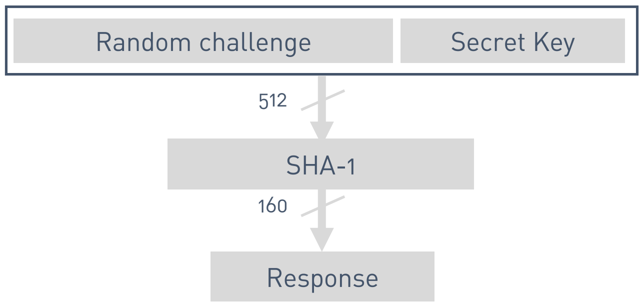
- SRAM based FPGA designs are vulnerable to design theft because configuration bit-streams can be easily captured and copied. FPGAs are more vulnerable to cloning of the entire design rather than to intellectual property (IP) theft, since extracting IP from the bit-stream is nearly impossible.



- A common way to protect the FPGA configuration bit-stream systems is to use the "identification, friend or foe" (IFF) design security approach.
- This solution disables the design within the FPGA until the hash algorithm computation matches in both the FPGA and a secure memory device, so the design remains secure even if the configuration data bit-stream is captured.

SHA-1 based Challenge and Response System Authentication





Identification, Friend or Foe Detection (IFF)

- The secure EEPROM holds a read-protected secret key
- The secret key is also part of the FPGA design.
- Why do we need the RNG in the FPGA design?
- What guarantees the security of this solution?

?

FSoC Lab Organization

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob

University of Applied Sciences Darmstadt

h_da

Faculty of Electrical Engineering and Information Technology

fbeit

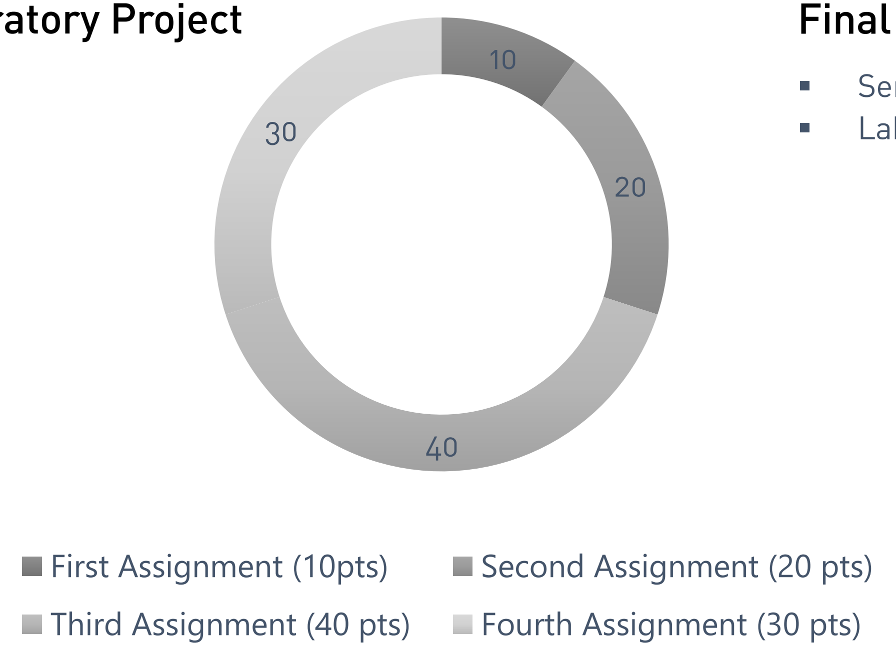
FSoC Lab Organization

1. **First Assignment** – **SHA-1 Software Design**
   C/Python @ PC
   10 pts.
2. **Second Assignment** – **SHA-1 Software Design**
   C @ Nios II
   20 pts.
3. **Third Assignment** – **SHA-1 Hardware Design**
   (SystemVerilog Implementation)
   40 pts.
4. **Fourth Assignment** – **SHA-1 Hardware/Software**
   HW/SW Co-Design
   (SystemVerilog AXI Bus Wrapper + C based BSP)
   30 pts.
5. **FSoC Design Contest** – **Optional: Performance Benchmark**
   Evaluate the performance of your solution in a given application context.
   Fame ;-)
   FSoC WS22/23

FSoC Lab Organization

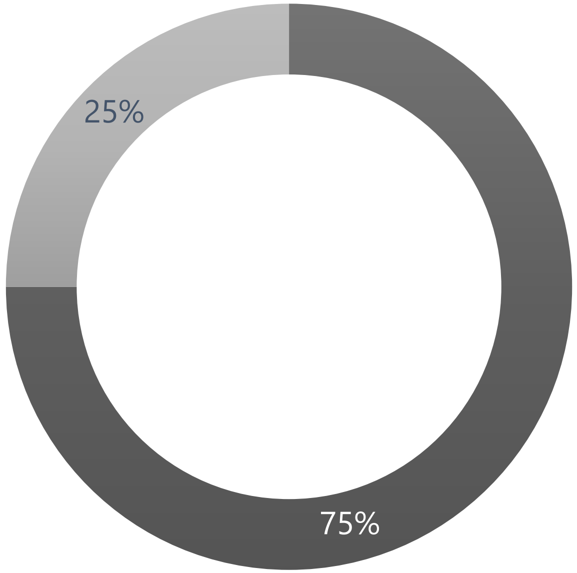
- The highest workload is associated with the third project task. In general, the workload is reflected b the overall number of points than can be achieved for the respective tasks.

Laboratory Project



Final Grade

- Semester Project (75%)
- Lab part (25%)



FSoC Lab Organization

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
					1.10	2.10
3.10	4.10	5.10	6.10	7.10	8.10	9.10
10.10	11.10	12.10	13.10	14.10	15.10	16.10
17.10	18.10	19.10	20.10	**21.10**	22.10	23.10
24.10	25.10	26.10	27.10	28.10	29.10	30.10
31.10						

October 2022

First Lab Submission
13.11.2022

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
	1.1	2.11	3.11	**4.11**	5.11	6.11
7.11	8.11	9.11	10.11	11.11	12.11	$\boxed{13.11}$
14.11	15.11	16.11	17.11	**18.11**	19.11	20.11
21.11	22.11	23.11	24.11	25.11	26.11	27.11
28.11	29.11	30.11				

November 2022

Second Lab Submission
04.12.2022

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
			1.12	**2.12**	3.12	$\boxed{4.12}$
5.12	6.12	7.12	8.12	9.12	10.12	11.12
12.12	13.12	14.12	15.12	**16.12**	17.12	18.12
19.12	20.12	21.12	22.12	23.12	24.12	25.12
26.12	27.12	28.12	29.12	30.12	31.12	

December 2022

Third Lab Submission
08.01.2023

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
						1.01
2.01	3.01	4.01	5.01	6.01	7.01	$\boxed{8.01}$
9.01	10.01	11.01	12.01	**13.01**	14.01	15.01
16.01	17.01	18.01	19.01	20.01	21.01	22.01
23.01	24.01	25.01	26.01	**27.01**	28.01	29.01
30.01	31.01					

January 2023

Fourth Lab Submission
05.02.2023

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
		1.02	2.02	3.02	4.02	$\boxed{5.02}$
6.02	7.02	8.02	9.02	$\boxed{\textbf{10.02}}$	11.02	12.02
13.02	14.02	15.02	16.02	17.02	18.02	19.02
20.02	21.02	22.02	23.02	24.02	25.02	26.02
27.02	28.02					

February 2023

Design Contest
10.02.2023

The Laboratory Tasks in Detail

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Laboratory Tasks

Start today!

- Get to know the SHA-1 algorithm

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
					1.10	2.10
3.10	4.10	5.10	6.10	7.10	8.10	9.10
10.10	11.10	12.10	13.10	14.10	15.10	16.10
17.10	18.10	19.10	20.10	**21.10**	22.10	23.10
24.10	25.10	26.10	27.10	28.10	29.10	30.10
31.10						

October 2022

Laboratory Tasks

First Lab Submission

13.11.2022

- **Theoretical part**: Question set
- **Practical part**: C Implementation of the SHA-1 algorithm

Mo.						
3.10						
10.10						
17.10						
24.10						
31.10						

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
	1.1	2.11	3.11	4.11	5.11	6.11
7.11	8.11	9.11	10.11	11.11	12.11	13.11
14.11	15.11	16.11	17.11	18.11	19.11	20.11
21.11	22.11	23.11	24.11	25.11	26.11	27.11
28.11	29.11	30.11			November 2022	
Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.

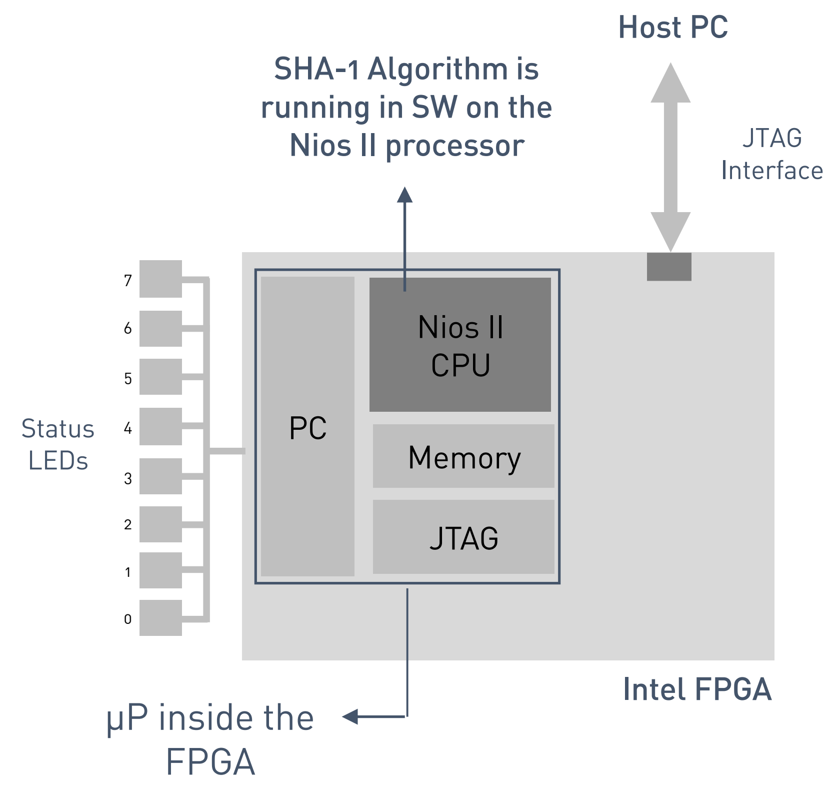
Suggested Procedure

- Set-up an Eclipse Platform using the Eclipse C/C++ Development Tools (CDT). Use this platform for your first C implementation of the SHA-1 algorithm.

Theoretical Part

1. What is a one way function?
2. What are typical applications of one way functions?
3. Define preimage resistance and the second preimage resistance characteristic of a one way function.
4. What is a collision and how does it affect the security of a hash function?
5. Does the Boolean XOR function represent a valid way to verify the integrity of a message. Justify your answer!
6. Why do collisions necessarily exist?
7. Explain the birthday problem and state the relation to hash collisions.
8. What role plays SHA-256 in the context of the cryptocurrency Bitcoin?

Laboratory Tasks



Second Lab Submission

04.12.2022

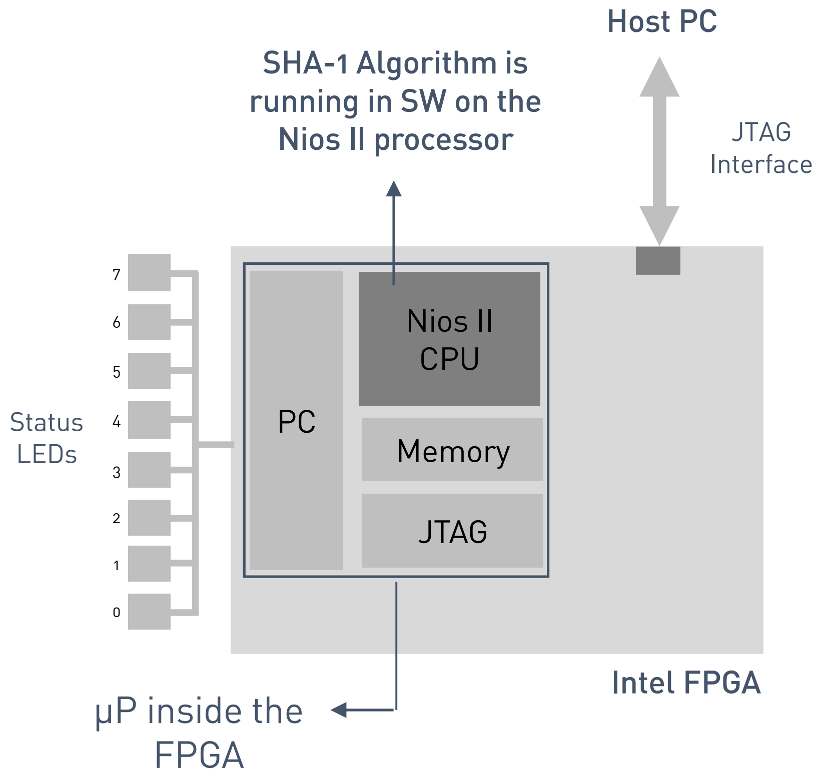
Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
			1.12	**2.12**	3.12	4.12
5.12	6.12	7.12	8.12	9.12	10.12	11.12
12.12	13.12	14.12	15.12	**16.12**	17.12	18.12
19.12	20.12	21.12	22.12	23.12	24.12	25.12
26.12	27.12	28.12	29.12	30.12	31.12	

December 2022

Laboratory Tasks

```
-----------------------------------------------
Altera Nios2 Command Shell [GCC 4]

Version 17.1, Build 590
-----------------------------------------------

Jakob@Jonathan /cygdrive/c/intelFPGA/20.1
$ nios2-terminal

nios2-terminal: connected to hardware target using JTAG Uart on cable
nios2-terminal: "USB-Blaster [USB-0]" device 1, instance 0
nios2-terminal: (USE the IDE stop button or CTRL-C to terminate)

FSoC 2022 is fun!

cb419b01c6c5402e0153b17785d899eaffb59d94
```

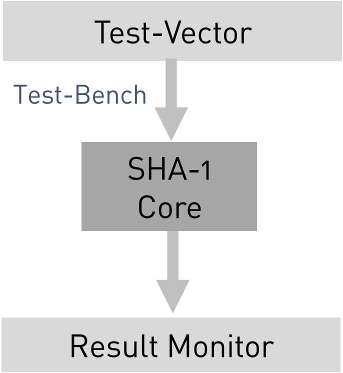
SHA-1("FSoC 2022 is fun!")



- Test-string provided by the user via the JTAG system console. The console is also used for writing back the result.

Laboratory Tasks

- The task is to implement the SHA-1 algorithm according to a given SystemVerilog template.
- Besides the actual data path, students need to design a corresponding control path.
- The correct functionality needs to be demonstrated by a dedicated test-bench and for a given set of test-vectors.



Third Lab Submission

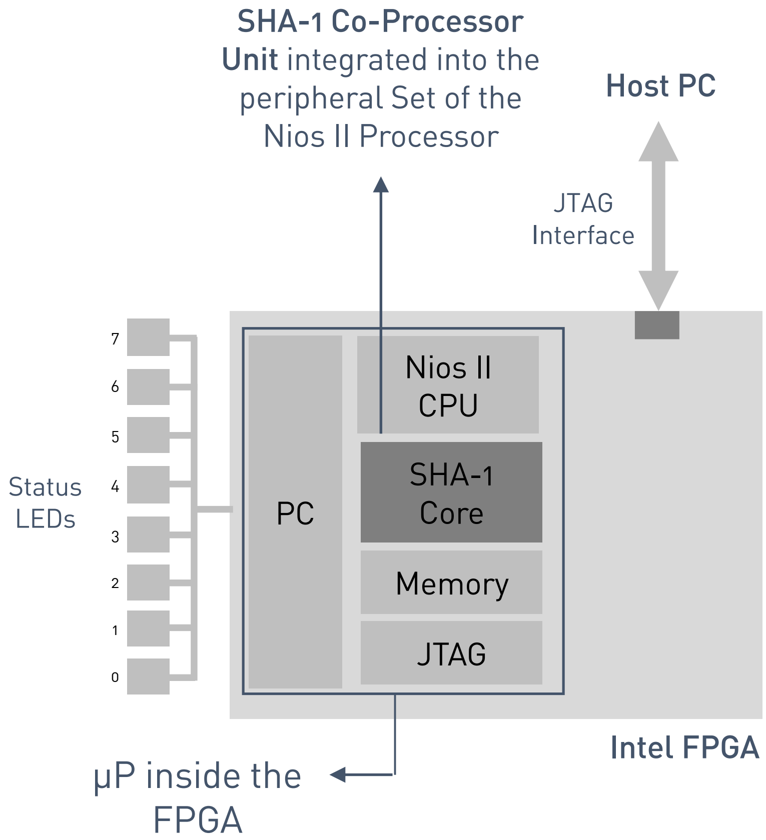
08.01.2023

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
						1.01
2.01	3.01	4.01	5.01	6.01	7.01	$\boxed{8.01}$
9.01	10.01	11.01	12.01	**13.01**	14.01	15.01
16.01	17.01	18.01	19.01	20.01	21.01	22.01
23.01	24.01	25.01	26.01	**27.01**	28.01	29.01
30.01	31.01					

January 2023

- SystemVerilog based SHA-1 Implementation according to a given set of requirements
- Testbench based functional verification using Mentor Graphics ModelSim

Laboratory Tasks



Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
		1.02	2.02	3.02	4.02	5.02
6.02	7.02	8.02	9.02	**10.02**	11.02	12.02
13.02	14.02	15.02	16.02	17.02	18.02	19.02
20.02	21.02	22.02	23.02	24.02	25.02	26.02
27.02	28.02				February 2023	

Fourth Lab Submission

05.02.2023

- Integrating the SHA-1 core into the peripheral set of a Nios II processor (Avalon wrapper + BFM based test-bench)

Laboratory Tasks

We are looking for the most powerful SHA-1 implementation

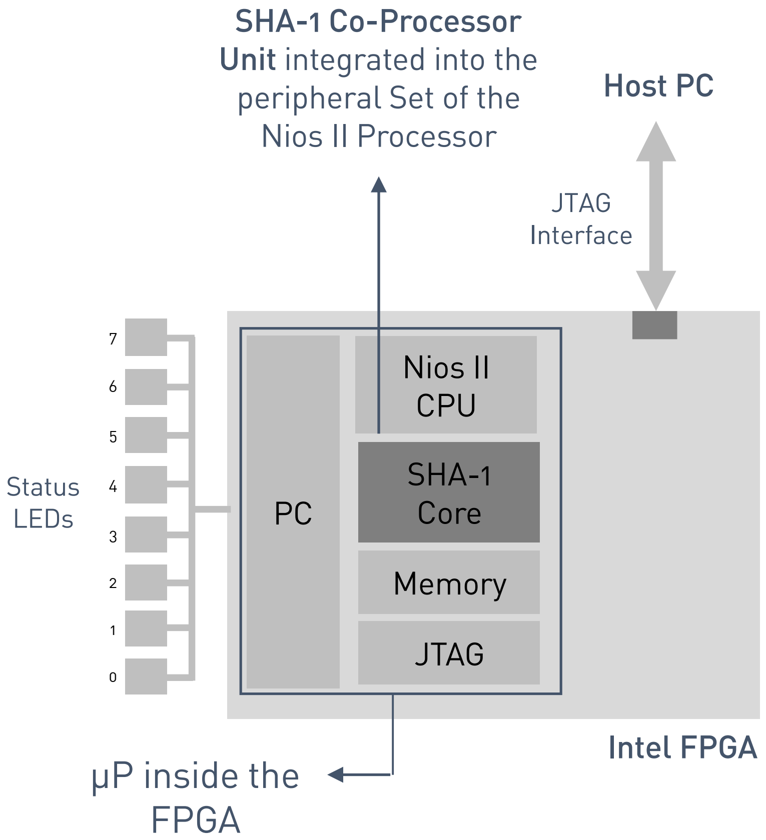
- Use and benchmark your design in a dedicated application context.

Mo.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
		1.02	2.02	3.02	4.02	5.02
6.02	7.02	8.02	9.02	10.02	11.02	12.02
13.02	14.02	15.02	16.02	17.02	18.02	19.02
20.02	21.02	22.02	23.02	24.02	25.02	26.02
27.02	28.02					

February 2023

Design Contest
10.02.2023

Laboratory Tasks



We are looking for the most powerful SHA-1 implementation

- Use and benchmark your design in a dedicated application context.
- Inspired by Patrick R. Schaumont annual design challenge running at the Professor Bradley Department of Electrical and Computer Engineering Virginia Tech.

Laboratory Tasks

- The task is to find an input string that yields a hash in which the leading n bits are zero, and the remaining (160-n) bits are don't cares. For small n, these types of bit sequences are very easy to find, however it becomes drastically more complex with increasing n.

- The parameter n obviously defines the difficulty of the search problem. This task is pretty similar to what is done in Bitcoin. Here, miners are looking for a SHA-256 hash that is less than or equal to some hash target. For example, if the hash target is 0000a1b2c3e4f5, any hash less than or equal to this number is a valid block hash. Many hashes would satisfy this requirement and any one of those would be a valid. However, it is an extremely difficult task to find such a hash.
- Lesser the hash target, the more difficult it is to find a satisfying hash.

Laboratory Tasks

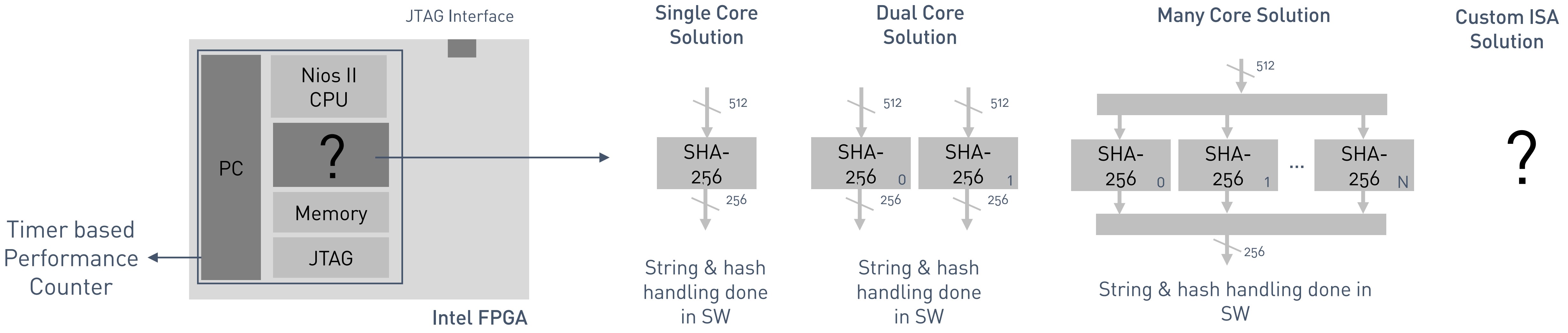
- There is no known way to do that except brute-force, adjusting a portion of the input block and calculating hashes over and over again until one of them finally fulfills this requirement by chance.

- So, what we’ve would like to do
  1. Choose an input string: “XXXA random input sequence”
  2. Pick a target, for example 14 bits.
  3. Replace the XXXX at the front of the string with a 32-bit counter value, starting from 0.
  4. Compute the hash of the resulting input string and check if the number of leading zeroes is equal to, or exceeds, 14 bits. If this is the case you have found a valid target. Report the counter value used and exit.
  5. Otherwise, if the number of leading zeroes does not meet the target, then increase the counter and repeat from step

- The previous algorithm is bound in speed by the speed at which you can compute a SHA-1 digest.

Laboratory Tasks

- So, who can build the fastest SHA-1 search engine?



- The average hit rate is determined by another HW peripheral. This timer based performance counter calculates the average time between two successive hits ...

Contest Rules and Ranking Criteria

- The following metrics will be used to evaluate and rank your design:
  - Functional correctness is mandatory
  - The average detection rate of your implementation. The higher the better.
  - The area efficiency, expressed in terms of detection rate per Logic Element (LE). Here, a smaller LE counter corresponds to a smaller design.
- The FSoC Design Challenge award:
  - One way to get famous ;-)
  - Official h_da fbeit certificate for outstanding project work
  - The contest event is organized with industrial partners

Recommended Readings

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Recommended Readings and Online Resources

- [**Online**]: National Institute of Standards and Technology NIST: FIPS PUB 180-4: Secure Hash Standard, 2012.
- [**Online**]: Das CrypTool-Portal - https://www.cryptool.org/de/
- [**Book**]: Security Engineering, Ross Anderson - http://www.cl.cam.ac.uk/~rja14/book.html
- [**Book**]: Applied Cryptography, Bruce Schneier - https://www.schneier.com/books/applied_cryptography
- [**Book**]: Understanding Cryptography : A Textbook for Students and Practitioners – SpringerLink
- [**Book**]: Practical Cryptography in Python - SpringerLink
- [**Online**]: Awesome security - an open repository - https://github.com/sbilly/awesome-security
- [**Online**]: CVE security vulnerability database/information source - http://www.cvedetails.com

fbeit
FACULTY OF ELECTRICAL ENGINEERING
AND INFORMATION TECHNOLOGY
Prof. Dr. C. Jakob
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
Slide 56

Academic Integrity

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Academic Integrity

- Writing source code is similar to academic writing in that when you use or adapt code developed by someone else as part of your project, you must cite your source.
- However in this particular case, the task is intended to be primarily individual effort. This means that all source code and documentation submitted for evaluation must be the student's original work.

Academic Integrity

- Therefore, every laboratory submission (written report as well as any SW and HW source codes in form of an achieved project directory) must include a declaration of authorship:

The full name of the author (surname, first name) including the Student IDs

I hereby declare that the work submitted is our own unaided work. I am aware that this work in digital form will be examined for the use of unauthorized aid and in order to determine whether this work as a whole or parts incorporated in it may be deemed as plagiarism.