Points	Grade

Team: 6

01228774 Constantin SCHIEBER #1
0122576 Petar KOSIC #2

Digital Integrated Circuits Lab (LDIS)

384.088, Summer Term 2018

Supervisors:
Christian Krieg, Martin Mosbeck, Axel Jantsch

Task 2: Implementing Argon2

Abstract

Parts of a key derivation function (KDF) - Argon2 - had to be implemented and integrated with implementations of other groups. The result of this task is a working KDF that can be reused.

A Permutation function, a compression function and a truncation function and the verification process are subsequently presented in this work.

1 Problem statement and motivation

Team 6 was tasked with the following items:

1. Implement steps 5-8 of Section 3.2 in the irtf-draft[1]
2. Implement compression function G of Section 3.5 in the irtf-draft[2]
3. Implement permutation P of Section 3.6 in the irtf-draft[3]
4. Implement the trunc(a) function in the irtf-draft[4]

As each sub task depends on its mother task it was decided to proceed in a bottom up order.
The truncation function trunc(a) truncates a 64 bit vector to a 32 bit vector and is needed to perform multiplications with two 32 bit vectors, which in turn help to increase the circuit depth of the implementation. This is important when the target hardware are ASICs or FPGAs.
Permutation function P takes one 128 byte input and applies a round function GB on it.
Compression function G takes two 1024 byte inputs and applies an XOR, the permutation function P and another XOR on these inputs.
Steps 5-8 of the Argon2 Operation are related to computing the actual output tag. They operate on a matrix B[i][j] which is, due to its size, placed in the DDR2 SDRAM chip. These steps use the compression function G and a variable hash function that is provided by Team 5. It also uses the computed indices i and j, provided by Team 6.

2 Implementation (proposed solution)

Some decisions regarding all parts of the implementation were made to provide a solution that works in simulation at least. These assumptions do not conform to the specification.

1. Grade achieveable of parallelism is 1
2. Design is not synthesizable and ignores especially the low area criteria
3. Simulation only possible with VHDL-2008 Standard, −std=08 flag mandatory

2.1 Truncation Function trunc(a)

The truncation function trunc(a) is implemented as a vhdl function in a package aswell as a fully fledged component. Our design uses the function, which is located in the permutate_pkg.vhd, due to its easier usage in sequential code (aka in processes).

2.2 Permutation Function P

The permutation function P was also implemented as a vhdl component at first. Due to the heavy usage of sequential code segments a redesign to a vhdl function was done. It resides now in the permutate_pkg.vhd. The implementation is as suggested by the[5] - but heavily sequential. One could implement a more elegant solution that allows for parallelization of round function GB calls that are independet of each other (note that not all calls are independet, as they operate on the same matrix).

2.2.1 Round Function GB

The round function GB doesn't have its own section but is the foundation of all other functionalities. Yet again it was implemented firstly as a component, only to be translated to a function, which also resides in the permutate_pkg.vhd. It is implemented straight forward as specified.

[1]"IRTF-Draft for Argon2". In:
[2]Ibid.
[3]Ibid.
[4]Ibid.
[5]Ibid.

2.2.2 Verification

As the round function represents a vital part of the implementation, the generation of good test cases was a main goal. To achieve this goal the code of the C reference solution[6] was analyzed and altered. The extraction of test vectors at key points of the operation was planned at several parts of the argon2 implementation, including the following files from the /src directory:

- ref.c
- blake2blake2b.c
- blake2blamka-round-ref.h
- ..Makefile

Extraction of the test vectors from the code was done by simple tagged printfs of the UINT64 data blocks. The blocks were printed in decimal for simplicity and further processing (see Listing 1). Removal of the tags was done by hand, a python script (convert_binary.py) then parses the blocks and converts them into their binary representation.

Listing 1: Suboptimal extraction point in blake2b.c

```
                ...
                ...
#define ROUND(r)
    do {
                printf("INPUTSTART\n");
                for (i = 0; i < 16; i++) {
                        printf("%" PRIu64 "\n", v[i]);
                }
                printf("INPUTEND\n");
                G(v[0], v[4], v[8], v[12]);
                G(v[2], v[6], v[10], v[14]);
                ...
                ...
                printf("OUTPUTSTART\n");
                for (i = 0; i < 16; i++) {
                        printf("%" PRIu64 "\n", v[i]);
                }
                printf("OUTPUTSTART\n");
```

First tests proved to be unsuccessful though, as the implementation of the permutation function in the C Code differs from the one proposed in the irtf-draft.[7] To double check on our proposed solution a python script that also implements the permutation P and round GB function was created.

The python and vhdl implementations did deliver the same output. Analyzing the reference solution more closely revealed that there are reference, optimized and actually used code lines that rely heavily on preprocessor statements.

Analysis of the Makefile showed, that a check for the platform was done (??), and if the platform was supported the optimized version of the files was used. After modifying the Makefile accordingly (and now actually using the reference solution) the generated output **did** match the output of our vhdl solution, see also table ??.

Listing 2: Parameters for the argon2 execution

```
echo -n "password" | ./argon2 somesalt -t 1 -m 16 -p 1 -l 24
```

Listing 3: Makefile, deactivate optimizations

```

OPTTEST := $(shell $(CC) -Iinclude -Isrc -march=$(OPTTARGET) src/opt.c -c \
                        -o /dev/null 2>/dev/null; echo $$?)
# Detect compatible platform
ifneq ($(OPTTEST), 0)
$(info Building without optimizations)
        SRC += src/ref.c
else
$(info Building with optimizations for $(OPTTARGET))
        CFLAGS += -march=$(OPTTARGET)
        SRC += src/opt.c
endif
```

For one part the upper half of the 128 Byte Input is initialized in an unexpected way that is not mentioned in the.[8]

[6] *Argon2 Reference Implementation.* URL: https://github.com/P-H-C/phc-winner-argon2.
[7] "IRTF-Draft for Argon2".
[8] Ibid.

Table 1: Permutation Function P Outputs

Input	Python3.5	VHDL	C Reference	C Actual
6A09E667F2BDC948	4B86FAA34237F816	4B86FAA34237F816	4B86FAA34237F816	3D9D014CA238A25D
510E527FADE682D1	826371B4B7CF06DB	826371B4B7CF06DB	826371B4B7CF06DB	D9CE83A69663A233
6A09E667F3BCC908	6915F3A835F68E52	6915F3A835F68E52	6915F3A835F68E52	B8023558C91686D7
510E527FADE682E9	4A645E346BE317D8	4A645E346BE317D8	4A645E346BE317D8	2E207F7532A740EC

This seems to be part of the regular way of implementing blake2b, see the code listing **??** where v[15] is initialized by loading a fixed UINT64 and XORing it with an unknown parameter f[1].

Listing 4: Suboptimal extraction point in blake2b.c

```c
/*Strange Vector Init*/ v[15] = blake2b_IV[7] ^ S->f[1];
        ...
        ...
#define G(r, i, a, b, c, d)
    do {
        a = a + b + m[blake2b_sigma[r][2 * i + 0]];
        d = rotr64(d ^ a, 32);
        c = c + d;
        b = rotr64(b ^ c, 24);
```

2.3 Compression Function G

2.4 Argon2, Steps 5-8

3 Results (verification plan)

4 Discussion

5 Conclusions

6 Assessment

This is the place for the teaching staff to add notes for team assessment.

#	**Issue**	**Yes**	**No**
1 Implementation			
1.1	Does the implementation conform to the specification?		
1.2	Is the implementation resource-efficient?		
1.3	Is the implementation's hardware description language (HDL) complexity low?		
1.4	Is the implementation well-documented?		
1.5	Is the file structure's complexity low?		
2 Coding style			
2.1	Is the line width of code limited to 80 characters?		
2.2	Is white space appropriately used?		
2.3	Are tabs used for indentation?		
2.4	Are separators used to logically divide the file contents?		
2.5	Are meaningful comments given?		
3 Code reuse			
3.1	Is publicly available code re-used?		
3.2	Is non-publicly available code re-used?		
3.3	Are the sources of re-used code cited?		
4 Interaction			
4.1	Was the specification unclear to the team?		
4.2	If yes, did the team contact the teaching staff to make the specification clear?		
5 Report			
5.1	Are there typos?		
5.2	Is the report grammatically correct?		
5.3	Is there redundant information?		
5.4	Is the report's format consistent?		
5.5	Are captions properly used and numbered? Page numbers?		
5.6	Are figures and tables properly referenced in the body text?		
5.7	Are resources properly referenced?		