European Commission

FPGA performances in Cryptography

Performance analysis of different cryptographic algorithms implemented in an FPGA

Final Report

Lubos Gaspar

Editors
Laurent Beslay, Iwen Coisel

2014

Report EUR 26833 EN

Joint Research Centre

European Commission
Joint Research Centre
Institute for the Protection and Security of the Citizen

Contact information
Laurent Beslay
Address: Joint Research Centre, Via Enrico Fermi 2749, TP 361, 21027 Ispra (VA), Italy
E-mail: laurent.beslay@jrc.ec.europa.eu
Tel.: +39 0332 78 6556
Fax: +39 0332 78 9392

JRC Science Hub
https://ec.europa.eu/jrc

JRC91528

EUR 26833 EN

ISBN 978-92-79-40329-3

ISSN 1831-9424

doi:10.2788/14648

Luxembourg: Publications Office of the European Union, 2014

Abstract

The objective of this report is to evaluate the performance of cryptographic algorithms within the oclHashCat software (i.e. namely the MD5 and Keccak hashing functions) and run on the Stratix V FPGA using OpenCL and VHDL languages. This document gives a series of recommendations related to performance optimizations and pitfall avoidance when implementing hash functions using Altera OpenCL SDK.

The Institute for the Protection and Security of the Citizen (IPSC) of the Joint Research Centre (JRC) is conducting the project "Prevention and Prosecution techniques for Identity Cybercrime " (P2TIC) which is supported by research activities related to fight against Cybercrime, and related ICT solutions to support criminal investigations, especially, target identification and digital evidence collection The project takes place within close collaboration with the European Cybercrime Center (EC3) of EUROPOL. One part of this project is to conduct a comparative benchmark analysis between two types of hardware assessing their efficiency to perform so-called brute-force attacks against several cryptographic algorithms. The two families of components are respectively the GPU (Graphic Processing Unit), and the FPGA (Field Programmable Gate Array).

In the frame of this project, the JRC has commissioned under the Appointment Letter Number 258903 a study to an external researcher, about the performances of cryptographic algorithms in FPGA. The objective of the work is to deliver a fine tuned and optimized implementation of decryption software, oCLHashcat, on a FPGA (Field Programmable Gate Array) board. The work consists of two series of deployment and performance evaluation, and of the preparation of a synthetic report on the opportunities and challenges of FPGA programming.

The author of this report is:

- Lubos Gaspar, Université catholique de Louvain, Belgium

Responsible Administrator
Laurent Beslay
Digital Citizen Security unit
European Commission, DG Joint Research Centre
Directorate G - Institute for the Protection and Security of the Citizen
Unit G07 - Digital Citizen Security
TP 361
Via Enrico Fermi 2749
21027 Ispra (VA), ITALY
Tel: +39 0332 78 6556
Fax: +39 0332 78 9392

Content

CONTENT 3

TABLE OF FIGURES 5

1. INTRODUCTION 6

2. INTRODUCTORY DAY 7

3. FIRST IMPLEMENTATION OF OCLHASHCAT 7

4. OPTIMIZATION OF THE FIRST IMPLEMENTATION 10

5.1 VHDL Implementation of the MD5 Hash Function 10

5. SECOND IMPLEMENTATION OF OCLHASHCAT 12

6.1 Optimizations Using OpenCL Attributes 12
6.2 Optimizations of the Kernel Throughput Using Rules 14
6.3 Optimization Using Single Work Item Kernel (Task Approach) 16
6.4 Adding More Rules 18
6.5 Keccak Implementation (Dictionary Attacks) 18

6. KERNEL MODIFICATIONS FOR BRUTE-FORCE ATTACKS 19

7.1 ASCII Password Message Generator 19
7.2 MD5 Brute-force Implementation 20
7.3 MD5 Brute-force Implementation with 10 Rules 20

7. COMPARISON OF FPGAS AND GPUS 21

8.1 OpenCL Implementation Results Summary 21
8.2 Third-party GPU Implementations' Performance Comparison 22

8. FUTURE PERSPECTIVES 26

9. DESIGN RECOMMENDATIONS 26

10. CONCLUSIONS 27

REFERENCES 28

APPENDICES 29

A. Description of Tasks 29
B. Global Memory Bandwidth Analysis 30

Table of Figures

Figure 1: Estimated resource utilization of the simple MD5 kernel 8
Figure 2: Estimated throughput of the simple MD5 kernel 8
Figure 3: High performance pipelined implementation of the MD5 algorithm 10
Figure 4: Estimated resource utilization of the vectorized MD5 OpenCL kernel 12
Figure 5: Throughput estimation of the vectorized MD5 OpenCL kernel 13
Figure 6: Estimated resource utilization of the replicated MD5 OpenCL kernel 13
Figure 7: Throughput estimation of the replicated MD5 OpenCL kernel 14
Figure 8: Throughput estimation of the MD5 OpenCL kernel with 2 rules 15
Figure 9: Estimated resource utilization of the MD5 OpenCL kernel with two rules 15
Figure 10: Throughput estimation of the MD5 OpenCL kernel with 2 rules and comparison 16
Figure 11: Estimated resource utilization of the MD5 OpenCL kernel with two rules and comparison . 16
Figure 12: Estimated resource utilization of the single-work-item MD5 OpenCL kernel with two rules and comparison 17
Figure 13: Estimated resource utilization of the MD5 OpenCL kernel with internal vectorization 17
Figure 14: Estimated resource utilization of the MD5 OpenCL kernel with 4 rules 18
Figure 15: Estimated resource utilization of the SHA-3 (Keccak) OpenCL kernel 19
Figure 16: Estimated resource utilization of the MD5 brute-force attack kernel 20
Figure 17: Estimated resource utilization of the MD5 brute-force attack kernel with 10 rules 21

1. Introduction

The objective of this report is to evaluate the performance of cryptographic algorithms within the oclHashCat software (i.e. namely the MD5 and Keccak hashing functions) and run on the Stratix V FPGA using OpenCL and VHDL languages. This document gives a series of recommendations related to performance optimizations and pitfall avoidance when implementing hash functions using Altera OpenCL SDK.

In the last decade the domain of the high performance computing gained a lot on its significance. This rapid development has been driven by new achievements in performance, power consumption and costs of the available hardware platforms. Initially, Central Processing Units (CPUs) or Application Specific Integrated Circuits (ASICs) were the only available platforms. ASICs deliver the highest performance with relatively low power consumption. “However, the price per unit is extremely high except when they are deployed on a very large market sector. Unlike ASICs, substantially slower CPUs are consumer products, and so their prices are relatively low.

Since early 2000s, wide-spread Graphic Processing Units (GPUs) gained their ground in the high performance computing market. GPUs are highly parallel ASIC devices, with relatively high clock frequencies and very large memory bandwidths. Thanks to their parallelism, many algorithms formerly developed for CPUs were ported to GPUs, and thus much higher performance could be achieved. Nevertheless, not all the algorithms can be optimally ported to GPU’s due to the fixed (and limited) instruction set available to program them. Moreover, the power consumption of these devices has significantly risen (i.e. more than 250 W for modern GPUs, and 100 W to 200 W for modern CPUs).

Recently, more and more attention has been given to Field-Programmable Gate Arrays (FPGAs). FPGAs are specially designed ASICs with configurable structure. The ability to be configured (and also reconfigured) leads to a very high parallelism, flexibility and low power consumption. Parallel algorithms implemented on FPGAs can be highly optimized due to their flexible inner structure. However, due also to a highly complex structure, FPGAs cannot reach the same high frequencies as CPUs or GPUs and are therefore less suitable for highly sequential algorithms. A large range of prices are present for FPGAs. The low-performance devices can be obtained for less than 10 €, and the price of the highest-performance devices can exceed 10,000 €. Although the FPGA market is already quite large and is rapidly growing, the development boards are still produced in small series and thus their prices can be higher than those of GPUs cards. Unlike GPUs, the power consumption of FPGAs hardly exceeds 20 W. For this reason, FPGAs can be considered as much "greener", while still being as fast as GPUs or even more when appropriately optimised.

Recently, an initiative to define an easily parallelizable programming language that spans various hardware platforms has been undertaken. This new programming language, proposed by the Khronos group, is called OpenCL and is derived from the C99 language. Although, this language was originally proposed for CPUs and GPUs, Altera extended it for its FPGAs by introducing Altera OpenCL SDK. Unlike traditional low level hardware languages (i.e. VHDL, Verilog), OpenCL designs can be developed in substantially less time. Moreover, the code originally written for GPUs should be relatively easy to port to FPGAs. Thus, we have decided to examine the design and performance aspects of the cryptographic kernels developed with Altera OpenCL SDK. Moreover, we compare the results with those obtained for GPUs and give some recommendations on how to optimize FPGA OpenCL kernels.

The document is structured as follows. Chapter 2 discusses the work that has been delivered for Task 1. First implementation of oclHashcat algorithms related to Task 2 is provided in Chapter3. Optimized VHDL version of this implementation related to Task 3 is detailed in Chapter 4. Second implementation of oclHashcat algorithms, as described by Task 4, is provided in Chapter 5. Designs of oclHashcat algorithms for brute-force attacks are described in Chapter 6. Chapter 7 analyzes the performance of GPU implementations and provides an in-depth comparison of GPUs and FPGAs while considering various aspects. Future perspectives are outlined in Chapter 8 and design recommendations are provided in Chapter 9. Chapter 10 summarizes the main achievements and concludes this report. The five tasks of the appointment letter are described in Appendix A.

2. Introductory Day

The first half a day in JRC started with the introduction of the Institute for the Protection and Security of the Citizen (IPSC) and discussions about the on-going research projects and eventual future development. More precisely, we discussed possible employment of Field-Programmable Gate Arrays (FPGAs) as hardware accelerators of various computationally extensive algorithms (i.e. cryptographic algorithms, on-line correlation computations, on-line Fourier or Wavelet transforms, etc.).

The discussions were followed by an introductory FPGA presentation. The first half of the presentation describes the FPGA technology, introduction to the fine-grain (i.e. logic fabric) and course-grain (i.e. embedded memory blocks, DSP blocks, PCI express and DDR3 hard IP blocks, etc.) resources and the interconnection network. In order to demonstrate the capabilities of modern FPGAs, the basic parameters of the Altera Stratix V FPGA family were discussed and some examples of the cryptographic implementations were given (i.e. Lapin authentication scheme and Spring cipher) and compared with the equivalent software implementations executed in modern CPUs.

The last part of Task 1 was dedicated to the demonstration of the Nallatech 385 A7 FPGA board (1) supporting OpenCL programming. We made preliminary performance tests using the published Altera OpenCL examples. We have also discussed the specifications of the final product. In order to facilitate at a later stage the comparison of different hardware platforms, we have selected the MD5 cryptographic algorithm (2) as a first suitable candidate.

3. First Implementation of oCLHashcat

This task was carried out in one and a half days in Ispra. In order to test portability of the OpenCL code, the oclGaussCrack software was selected (3). We chose this software, because it is developed in OpenCL and its code is publically available. This software can be used to brute-force the password used by the Gauss virus encryption (4). OclGaussCrack kernel computes 10,000 MD5 loops and compares the resulting hash. The kernel is optimized for GPUs.

The first step in porting the oclGaussCrack project to Altera platform is to compile the kernel source code using the following command:

```
aoc-v --board PCIe385n_a7 oclGaussCrack.cl -ooclGaussCrack.aocx
```

However, the compilation procedure failed. After a thorough examination of the issue, we have found out that Altera OpenCL SDK does not support vector type constant arrays that are used in the oclGaussCrack source code (see Altera OpenCL programming guide (5), page 22). For this reason, the compilation failed when evaluating "all" function on the work register vector. Moreover, other limitations of the Altera OpenCL compiler are listed in Appendix B of the Programming Guide (5). Because of these limitations, it is not straightforward to port a general OpenCL code to Altera platform and some modifications of the source code may be necessary.

In order to overcome those limitations, we have decided to write our own MD5 kernel which was simplified in order to eliminate possible incompatibility issues. Before fully compiling the kernel a throughput and size estimations were performed (see Figure 1). Interestingly, the simplified MD5 kernel with its subsystem (including DMA, PCIe interface, DDR3 interface, Avalon interconnect, etc.) occupied 20 % of the logic resources, 9 % of registers and 16 % of the embedded memory blocks. No DSP blocks were used. This low utilization of resources is beneficial, since the unused resources can be used to replicate the kernel and execute more hashes in parallel. We will discuss these optimizations later.

```
+-----------------------------------------------------------------+
; Estimated Resource Usage Summary                                ;
+--------------------------------------+--------------------------+
; Resource                             + Usage                    ;
+--------------------------------------+--------------------------+
; Logic utilization                    ;   20%                    ;
; Dedicated logic registers            ;    9%                    ;
; Memory blocks                        ;   16%                    ;
; DSP blocks                           ;    0%                    ;
+--------------------------------------+--------------------------;
```

Figure 1: Estimated resource utilization of the simple MD5 kernel

The performance evaluation (see Figure 2) indicated that one kernel was implemented (i.e. compute units = 1) and it contains one pipeline[1] (i.e. simd work items = 1). More interestingly, two different throughput estimations were given. The first value *Throughput due to control flow analysis* indicates how many work items (i.e. hash values) can be calculated in one second by the kernel assuming each work item is calculated in 1 clock cycle. This value thus also expresses the overall pipeline latency of 240 clock cycles. This estimation does not reflect global memory bandwidth limitations (i.e. external DDR3 memory bandwidth). Our MD5 kernel implementation reached 240 million hashes per second (MH/s). The second value *Kernel global memory bandwidth analysis* estimates the necessary global memory bandwidth to satisfy the maximal kernel throughput of 240 MH/s. The necessary global memory bandwidth is 20,210.53 MB/s (in this case MB is assumed as 10^6 bytes, not as 2^{20} bytes), which is equal to 19,274.26 MiB/s.

```
.. simd work items                         : 1
.. compute units                           : 1
.. throughput due to control flow analysis : 240.00 M work items/second
.. kernel global memory bandwidth analysis : 20210.53 MB/second
.. kernel number of local memory banks     : none
```

Figure 2: Estimated throughput of the simple MD5 kernel

[1] Pipeline is a chain of data-processing hardware stages. In general, each stage can process data in one time instance (1 clock cycle), then pass the results to the next stage and load new data from the previous stage. An N-stage pipeline can process one data block in only one time instance, but with the latency of N time instances.

If we assume that the MD5 kernel inputs 512-bit data and outputs 128-bit hash values, roughly 640 bits of data needs to be transferred between the external DDR3 memories and the MD5 kernel in the FPGA. Since the kernel can process 240 million work item per second, this corresponds to 240M * 640b = 18,310.55 MiB/s. This value is lower than required global memory bandwidth, because it does not include inefficiencies of the transfer, pipeline stalls and potential bubbles[2] in the pipeline related to DDR3 addressing and other control flow operations. Nevertheless, the required global bandwidth is below the maximum bandwidth of the two memory banks of 25,600 MB/s (see Appendix B for the calculation of this value), and so the core can be overclocked to reach the bandwidth of the two DDR3 banks. This is done automatically by the Altera compiler. Manually it can be calculated as follows: (25,600 / 20,210.53) * 240 MHz = 304 MHz. After the full compilation, this kernel frequency can be verified in the *acl_quartus_report.txt* file. If we assume the clock frequency of 304 MHz, the maximal kernel throughput increases to 304 million work items per second.

This throughput estimation was followed by the computationally extensive full compilation and synthesis of the kernel (in approximately 1 hour). In order to test the kernel a host application (later referred to as *the host*) was developed. The host randomly generates 10 million 512-bit messages, sends them to the external DDR3 memories on the Nallatech board and starts the kernel computations. Since the profiling feature was enabled, the kernel computation duration is precisely measured and sent back to the host when completed. The complete duration including data transfers is also measured and displayed. Finally, the host calculates expected hashes and compares them with the obtained hashes from the kernel to verify the computation. The complete duration for processing 10 million 512-bit messages is 680ms from which the kernel computations are performed in only 39.889ms.This clearly shows that the data transfer between the PC main memory and the Nallatech card DDR3 memory represents the most time-consuming operation. Moreover, the transferred data are already aligned to 64-byte blocks to utilize the full potential of the PC DMA controller.

In order to estimate maximal kernel throughput we can calculate the measured kernel throughput as follows: (1,000ms/39.889ms) * 10,000,000 work items = 250,695,680 work items per second. This measured throughput is inferior to the estimated throughput of 304 M work items / second. The reason of this throughput difference is most probably caused by the suboptimal pipeline utilization. Verification of this assumption is not straightforward, because it requires analyzing the Quartus II[3] project generated by Altera OpenCL compiler. Unfortunately, the generated low-level Verilog files in the Quartus II project are only hardly readable. Moreover, Altera neither provides any documentation to this project nor any lower-level analyses procedure. For this reason, we have decided not to further analyze the generated Verilog project and rather try to optimize the high-level OpenCL project.

In order to easily follow the performance improvements as they are introduced throughout this document, we summarize the results in Table 3 in Chapter 7. The results of this first implementation are in the first row.

[2] If a pipeline is not filled completely (i.e. "bubbles" are inside the pipeline), the output of the pipeline will not produce a result in each time instance. This lowers the pipeline performance. Bubbles can be prevented if the pipeline execution is stalled until enough data are available on its input. Stalls also decrease performance.

[3] Quartus II is the main Altera software for development and verification of the FPGA designs. Altera OpenCL compiler only converts the OpenCL code to low-level Verilog code and the rest is carried out by Quartus II (synthesis, place and routes, bitfile generation).

4. Optimization of the First Implementation

4.1 VHDL Implementation of the MD5 Hash Function

Task 2 elaborated on the size and performance of the OpenCL implementation of the MD5 hash core. However, to assess the maximum performance of the MD5 implementation on the Stratix V FPGA, we developed a highly optimized VHDL implementation of the MD5 core. The comparison of the VHDL and OpenCL implementations is the best way to assess the performance gap between these two design approaches.

Designing a full FPGA system on chip would require implementing the DDR3, PCI express and DMA controllers interconnected via a high-performance interface to the kernel. However, such work would require much more design effort than what was foreseen for this application letter. Therefore we only concentrate on the kernel design in the VHDL code, a precise estimation of the maximum clock frequency, and clock-accurate simulation. This information gives an accurate estimation of the kernel throughput.

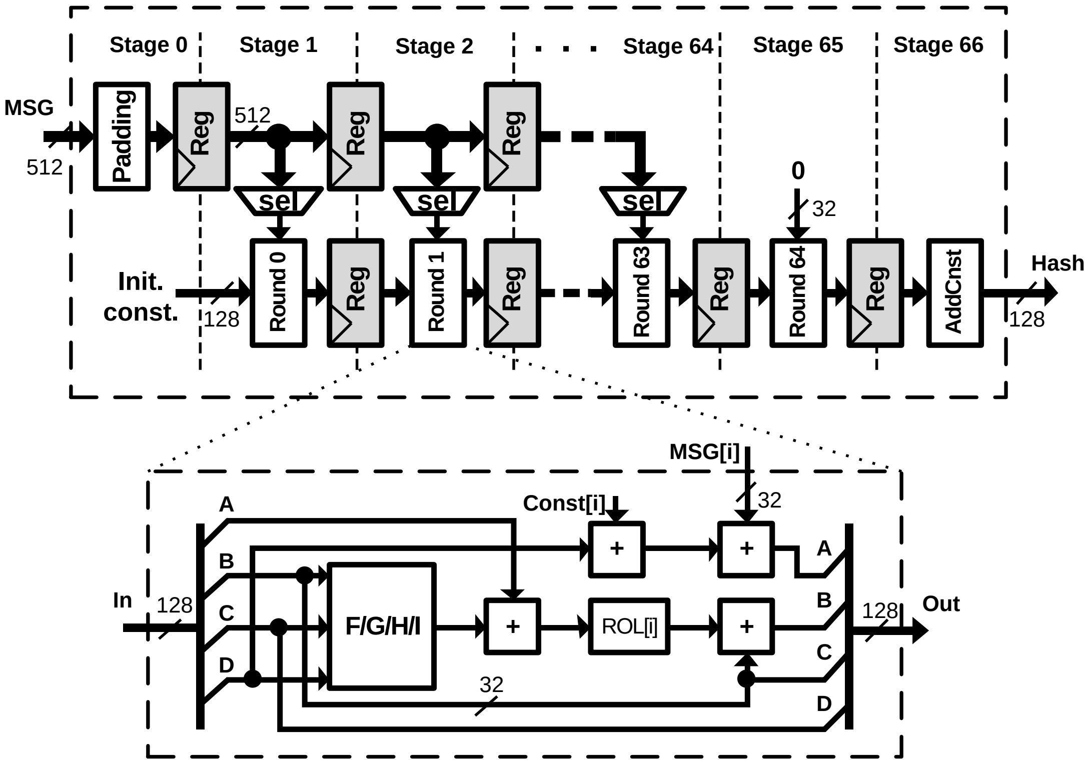

Figure 3: High performance pipelined implementation of the MD5 algorithm

The MD5 implementation, slightly modified from the one described in (6), is illustrated in Figure 3. The MD5 core features fully unrolled pipeline composed of 66 stages. Each stage contains a combinatorial round unit except the initial and final stages. Lower part of Figure 3details the round unit structure. Depending on the stage index, a round unit can implement one of the four MD5 sub-functions (i.e. F, G, H or I), one of 16 rotations, three full additions, and one addition of a round constant. Each round unit processes a corresponding 32-bit word of the padded input message. The last stage generates a 128-bit hash output value. One may notice that the size of an input message is fixed to exactly one 512-bit block. This optimization is possible, because our MD5 core is designed for password cracking, where size of each password, even including salt, is inferior to 512 bits (less than 64 bits for an 8 alphanumerical characters password).

Since the core is fully pipelined, one 128-bit hash value of a 512-bit input message can be calculated in each clock cycle with the initial delay of 66 clock cycles. However, if stalls or bubbles are present in the pipeline, the performance can be degraded. This can be avoided by processing a large number of input messages to keep the pipeline constantly filled. MD5 core could be used either for the dictionary attacks (unfortunately it requires implementing the aforementioned DDR3/PCIe interface) or the brute-force attacks. In case of the brute-force attacks enough messages can be generated directly in the FPGA. Moreover, comparison of a resulting hash value with a reference hash can be carried out by the FPGA, too. This new pipeline avoids the bottleneck of the transfer from the PC to the board and thus increases significantly the performances. In addition, the performance can be multiplied if this core pipeline is replicated several times to fully occupy the FPGA resources.

The full MD5 password cracker consists of one 66-stage MD5 pipeline, a LFSR-based generator of input messages (using case sensitive alphanumerical alphabet, thus 64 characters) and a hash comparator. In order to estimate required FPGA resources, it was synthesized for Altera Stratix V FPGA (the same chip as the one in the Nallatech OpenCL A7 card). Results of the MD5 password cracker are given in Table 1. As can be observed, the full password cracker occupies only 7% of all logic resources. Altera recommends keeping the project size below 85% to simplify placement and routing steps. This way, approximately 12 pipelines can be instantiated in one such FPGA device, thus multiplying the throughput by 12.

Table 1: Resource utilization / performance estimation of the high performance MD5 password cracker described in VHDL

Logic utilization (ALMs)	14,749 of 234,720 (6.3 %)
Registers	29,149
Maximum frequency	490.2 MHz
Max. throughput - 1 pipeline	490.2 MH/s
Max. throughput - 12 pipelines*	5,882.4 MH/s

* Estimated from the size/performance of 1 pipeline

The maximum clock frequency of the password cracker was estimated to 490.2 MHz. Since one hash value can be computed in one clock cycle, 490.2 million hashes could be calculated per second by one pipeline operating at the maximum clock frequency. In case of 12 pipelines the speed would be 12 * 490.2 MH/s = 5,882.4 MH/s. Both values for a single pipeline and 12 pipelines represent very promising results. The TimeQuest[4] time analysis reported critical paths inside the round unit. Indeed, each round unit contains a long path from the inputs B, C and D to the output B (see the I/Os of the round unit in Figure 3) consisting of two 32-bit adders connected in series. The critical paths may be improved by specifying more timing constraints and compiling the project repeatedly until these constraints are met. This, however, required more development effort. Additionally, performance could be improved if the adders could be replaced by the hardwired adders inside DSP blocks[5]. This approach requires further examination.

[4] TimeQuest is the Altera tool for analyzing the low-level timing of the developed core. The tool can be used to locate the critical paths (the longest paths in the core). Long paths are caused by sequentially interconnected combinatorial logic blocks. The longer the critical paths are, the lower the maximum clock frequency of the design could be reached.

[5] Digital Signal Processing (DSP) blocks available in FPGAs are hardwired blocks (hardware is fixed as in ASICs) that allow much higher performance than if the same functions are implemented in the reconfigurable logic.

If we compare the throughput of one pipeline described in the OpenCL and VHDL codes, we may assess a performance gap between the two implementations. The MD5 password cracker described in VHDL is about 1.95 times faster than the OpenCL implementation. However, the OpenCL kernel frequency may not reach its maximum frequency, because of the DDR3 memory bandwidth bottleneck. In order to compare the two design approaches fairly, we redesigned the OpenCL kernel to execute brute-force attacks (more details are given in Chapter 6). The OpenCL brute-force MD5 kernel reached 277.65 MH/s and operated at 278.47 MHz clock frequency. Thus, we can observe that the VHDL optimized implementation is about 1.77 times faster than the comparable one written in OpenCL. As mentioned before, the complexity of the OpenCL subsystem is the most probable cause of this significant difference in the maximum clock frequency.

The results obtained in this VHDL implementation can be compared with our other implementations using Table 3 and Table 4.

5. Second Implementation of oclHashcat

These sections explain the optimization design space of the MD5 and KeccakOpenCL kernels. First, we would like to explore optimization possibilities assuming dictionary attacks. The considerations for brute-force attacks are summarized in Chapter 6.

5.1 Optimizations Using OpenCL Attributes

The simplest way to multiply performance is to vectorize the kernel (i.e. replicate its internal pipeline). Altera proposes this option as the most convenient, since no further changes in the kernel or the host code are necessary (except for specifying the work group size using the attribute *reqd_work_group_size*). For this reason, we have added the following two lines in the kernel:

```
__attribute__((num_simd_work_items(4)))
__attribute__((reqd_work_group_size(256,1,1)))
```

The kernel resource estimation is shown in Figure 4. We can see the logic utilization increase from 20% to 35%. From this observation we can conclude that that one kernel pipeline occupies approximately 5 % and the 15 % are occupied by the rest of the system (i.e. kernel dispatcher, DDR3, PCIe, DMA and the bus interfaces).

```
+----------------------------------------------------------------+
; Estimated Resource Usage Summary                               ;
+--------------------------------------+--------------------------+
; Resource                             + Usage                    ;
+--------------------------------------+--------------------------+
; Logic utilization                    ;   35%                    ;
; Dedicated logic registers            ;   15%                    ;
; Memory blocks                        ;   24%                    ;
; DSP blocks                           ;    0%                    ;
+--------------------------------------+--------------------------;
```

Figure 4: Estimated resource utilization of the vectorized MD5 OpenCL kernel

The throughput estimation of the code is illustrated in

Figure 5. It can be observed that the theoretical kernel throughput is 960 million work items per second, which is four times more than in the previous OpenCL implementation. However, the required memory bandwidth increased to roughly 86 GB per second which is far more than the real DDR3 interface throughput (25,600 MB/s). In order to match the real DDR3 throughput, compiler must reduce the throughput by 71 %. As a result, the estimated throughput limited by the global memory decreases to 283.17 million work items per second. This way, the advantage of the vectorization is lost.

```
.. simd work items                    : 4
.. compute units                      : 1
.. throughput due to control flow analysis : 960.00 M work items/second
.. kernel global memory bandwidth analysis : 86789.11 MB/second
.... must derate throughput by a factor of : 0.29
.. kernel number of local memory banks     : none
GLOBAL Bandwidth Exceeded !!
.. throughput limited by global memory     : 283.17 M work items/second
```

Figure 5: Throughput estimation of the vectorized MD5 OpenCL kernel

The hardware benchmark confirmed the result. The vectorized kernel computed 10.24 million hashes in 41.231ms. This result corresponds to the measured kernel throughput of 248 million work items per second. However, the hardware benchmark has also discovered that not all returned hash values were correct. Strangely enough, 190 hash values at the beginning were correct and all the rest corrupted. Moreover, we have executed the same test several times and the number of correct hash values at the beginning varied from about 32 to 200. This is a very strange result that we were not able to explain. It can be a bug in the OpenCL compiler, since Altera claims that kernel behavior should not change (except for the speed increase).

Another approach described in the Altera documentation suggests replicating the whole kernel. This can be done by adding the following code to the original kernel:

```
__attribute__((num_compute_units(2)))
```

The kernel compilation resource estimation is given in Figure 6. We can see the logic utilization increase from 20% to 27%. From this observation we can conclude that each complete kernel occupies roughly 7% and the rest of the system about 13%. In this case the rest of the system is smaller than in the previous case (15%), because most of the control logic is embedded directly into each core.

```
+-----------------------------------------------------------------+
; Estimated Resource Usage Summary                                 ;
+--------------------------------------+--------------------------+
; Resource                             + Usage                    ;
+--------------------------------------+--------------------------+
; Logic utilization                    ;   27%                    ;
; Dedicated logic registers            ;   11%                    ;
; Memory blocks                        ;   20%                    ;
; DSP blocks                           ;    0%                    ;
+--------------------------------------+--------------------------;
```

Figure 6: Estimated resource utilization of the replicated MD5 OpenCL kernel

Since the memory bottleneck has already been reached by the original kernel, we did not expect to reach high performance, because of the same issue. However, we conducted this optimization to examine the impact on the kernel result correctness. The throughput estimation is shown in Figure 7. As expected the required memory throughput is two times higher than of the original kernel. As a result, performance is reduced to 63% of the kernel throughput.

```
.. simd work items                     : 1
.. compute units                       : 2
.. throughput due to control flow analysis : 480.00 M work items/second
.. kernel global memory bandwidth analysis : 40421.05 MB/second
.... must derate throughput by a factor of : 0.63
.. kernel number of local memory banks     : none
GLOBAL Bandwidth Exceeded !!
.. throughput limited by global memory     : 304.00 M work items/second
```

Figure 7: Throughput estimation of the replicated MD5 OpenCL kernel

Unlike the kernel vectorization strategy, all returned hash values were correct in case of the kernel replication strategy. However, the computation of 10.24 million hash values required 162.085 ms, which corresponds to the measured kernel throughput of 63 million work items per second. This result is clearly much worse than the result of the original kernel. It may be explained by the fact that two kernels are competing for the memory access which leads to undesirable memory bus contentions.

None of the two strategies proposed by Altera is suitable for our application. Thus, we have decided to modify the kernel and its communication protocol to decrease memory transfers and reuse the transferred data several times.

5.2 Optimizations of the Kernel Throughput Using Rules

This section introduces the concept of rules for dictionary attacks that allow multiplying the number of computed hash values by the kernel while maintaining the same global memory throughput. The idea is in reading a message from the global memory and applying several rules on it. The rules could be as follows:

Rule 1: The most basic rule would be to keep the message as it is and calculate hash (as was done before).

Rule 2: This additional rule could convert all lowercase ASCII characters in the same input message to uppercase and then calculate the hash value. This way we reuse a single memory transfer two times.

5.2.1 PC Verifies All Hashes

The two generated hash values could be sent back to PC for verification. Although this introduced some transfer overhead (< 20%), the number of calculated hashes doubled. However, it is challenging to write the code such that the Altera OpenCL compiler could fully fill the pipeline or even instantiate another pipeline if the capacity of the first one is exceeded[6]. For this reason, we have decided to describe the execution of the two rules using a "for" loop and completely rely on the intelligence of the compiler. Moreover, the aforementioned strategies to replicate the kernel or its pipeline using attributes failed, so we didn't consider them any longer.

The estimated throughput is shown in Figure 8. When compared to kernel without rules, the global memory bandwidth increased, because two hash values are transferred back to PC. However, the estimated kernel throughput is the same, since the kernel processed the same number of work items.

```
.. simd work items                    : 1
 .. compute units                     : 1
 .. throughput due to control flow analysis : 240.00 M work items/second
 .. kernel global memory bandwidth analysis : 24252.63 MB/second
 .. kernel number of local memory banks     : none
```

Figure 8: Throughput estimation of the MD5 OpenCL kernel with 2 rules

Figure 9 reports the estimated number of resources for the kernel. As one may observe, addition of the new rule required additional 4% of logic resources, 1% of register and 2% of embedded RAM memory resources. Moreover, less than a quarter of the FPGA resources are utilized, so there is still enough room for addition of new rules.

```
+-----------------------------------------------------------------+
; Estimated Resource Usage Summary                               ;
+--------------------------------------+--------------------------+
; Resource                          + Usage                ;
+--------------------------------------+--------------------------+
; Logic utilization                  ;   24%                ;
; Dedicated logic registers          ;   10%                ;
; Memory blocks                      ;   18%                ;
; DSP blocks                         ;    0%                ;
+--------------------------------------+--------------------------;
```

Figure 9: Estimated resource utilization of the MD5 OpenCL kernel with two rules

The kernel computation time has increased to 50.456 ms for 10.24 million work items using two rules. This corresponds to the measured kernel throughput of (1000 ms/50.456ms) * 10.24 million * 2 = 405.9 MH/s which represents 62% increase. The total time (including transfers) increase to 806.749 ms. This increase is caused by transferring more messages (10.24 million when compared to previous 10.00 million) and one extra hash value per message.

Although the compiler gives no details how the data dispatching and pipelines are implemented, from the measured throughput and utilized resources we can assume that the compiler implemented the two pipelines in parallel automatically.

Thus, the kernel outputs only when the match is found. However, neither the Altera nor the Khronos group (i.e. the OpenCL development group) documentation explain how to modify the programming model, such that kernel is executed many times (i.e. once for each

[6] If the capacity of one pipeline is filled, the only way to increase the throughput is to create another pipeline that works in parallel with the first one. This way, data can be dispatched into the two pipelines and fill them equally to effectively double the performance.

work item) but only the result of one work item (i.e. the password message where hashes match) is returned back to the PC.

5.2.2 Hash Verification by the Kernel

The throughput estimation is given in Figure 10. Strangely enough, required global memory throughput is much higher than in the previous case. Since we do not know how the compiler interprets the kernel code, it is hard to explain these results. Moreover, the throughput is limited to only 99.16 million work item per second, which is probably related to the decrease of the kernel clock frequency due to the high required global memory throughput.

```
.. simd work items                    : 1
.. compute units                      : 1
.. throughput due to control flow analysis : 240.00 M work items/second
.. kernel global memory bandwidth analysis : 61961.53 MB/second
.... must derate throughput by a factor of : 0.41
.. kernel number of local memory banks     : none
GLOBAL Bandwidth Exceeded !!
.. throughput limited by global memory     : 99.16 M work items/second
```

Figure 10: Throughput estimation of the MD5 OpenCL kernel with 2 rules and comparison

Although we expected less utilized resources, because kernel returns only one message, the kernel is finally much bigger (see Figure 11) than in the previous case.

```
+-----------------------------------------------------------------+
; Estimated Resource Usage Summary                                ;
+--------------------------------------+--------------------------+
; Resource                             + Usage                    ;
+--------------------------------------+--------------------------+
; Logic utilization                    ;   34%                    ;
; Dedicated logic registers            ;   15%                    ;
; Memory blocks                        ;   28%                    ;
; DSP blocks                           ;    0%                    ;
+--------------------------------------+--------------------------;
```

Figure 11: Estimated resource utilization of the MD5 OpenCL kernel with two rules and comparison

Surprisingly, the measured kernel time reached 5031 ms which is absolutely inacceptable result. For this reason, we decided to abandon the *multi-work-item* programming model and to try a so called *single-work-item* programming model.

5.3 Optimization Using Single Work Item Kernel (Task Approach)

The single-work-item kernel programming model is often used when only a single execution of the kernel is necessary. This programming model is very close to classical programming where a function is executed only once. Although, password cracking requires processing much more messages, the single-work-item kernel can be adapted to this problem, too. The idea is that the kernel is executed only once, but it reads one message after another from the global memory, computes their hash values, compare them with the reference one, and outputs a single result (either positive or negative). What is not clear is how the Altera compiler would compile such a kernel, (i.e. what will be the pipeline depth, how many pipelines will be instantiates, etc.).

The multi-work-item kernel can be transformed to single-work-item one by adding the following attribute:

```
__attribute__((task))
```

Moreover, the global index is no more valid in this programming model. Instead, we can work with the input message memory pointer as with an array that contains N messages and simply access these messages by reading the elements of this array in a "for" loop. Moreover, two previously explained rules are applied to the input message to increase performance.

Unlike multi-work-item programming model, in single-work-item mode no throughput estimation was given by the compiler. This is caused by the fact, that the compiler is not aware of how many messages will be in the global memory and if a pipeline could be filled. Thus, the throughput cannot be estimated.

Figure 12 summarized the utilized resources. One may observe a slight increase of them. This may be caused by the additional hash comparator as well as a necessary local storage for the reference hash value.

```
+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                    ;
+----------------------------------------+----------------------------+
; Resource                               + Usage                      ;
+----------------------------------------+----------------------------+
; Logic utilization                      ;   26%                      ;
; Dedicated logic registers              ;   11%                      ;
; Memory blocks                          ;   20%                      ;
; DSP blocks                             ;    0%                      ;
+----------------------------------------+----------------------------;
```

Figure 12: Estimated resource utilization of the single-work-item MD5 OpenCL kernel with two rules and comparison

The measured kernel time to process 10 million messages using 2 rules was 47.152 ms. The total time including message transfers decreased to 588.162 ms.Thus, the measured kernel throughput increased to (1,000 ms/47.152ms) * 10,000,000 * 2 = 424,16 MH/s. This result is about 4.5% better than in case of multi-work-item model with all hash values returned. Moreover, this decreases the computational effort of the CPU, because no comparison is necessary by the host. In the light of all these advantages, we considered only single-work-item programming model from this point.

5.3.1 Optimization Using Single Work Item Kernel with Internal Vectorization

In order to test the capabilities of the Altera OpenCL compiler we decided to try a different approach to increase parallelism of the kernel while keeping the single-work-item kernel. Instead of executing both rules in a *"for" loop*, each MD5 variable is replaced by a *vector variable* (i.e. uint is replaced by uint2). This vector variable is similar to a structure containing a 2-element array of the same variable. Each rule is represented as one component of the vector. This approach represents the internal vectorization of the kernel.

The resource utilization is given in Figure 13. Surprisingly, this internal vectorization strategy resulted in almost identical implementation size.

```
+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                    ;
+----------------------------------------+----------------------------+
; Resource                          + Usage                   ;
+--------------------------------------+--------------------------+
; Logic utilization                   ;   26%                  ;
; Dedicated logic registers           ;   11%                  ;
; Memory blocks                       ;   20%                  ;
; DSP blocks                          ;    0%                  ;
+--------------------------------------+--------------------------;
```

Figure 13: Estimated resource utilization of the MD5 OpenCL kernel with internal vectorization

Moreover, the measured kernel time for processing 10 million messages is 47.175 ms, which is almost identical to the previous result. As both strategies are equivalent, the internal vectorization solution was not considered anymore.

5.4 Adding More Rules

Performance can be further increased if more rules are added. For this reason, we define two more rules (rules 3 and 4) as follows:

Rule 3: This rule locates only the first ASCII character in the message and converts it to uppercase. This is often the case (i.e. names) when words start with an uppercase character.

Rule 4: This rule translates the selected ASCII characters in the message to the so called *leetspeak* (7). For this purpose, we have selected the letters given in Table 2.

Table 2: Conversion of the selected letters to leet speak alphabet

ASCII	a	b	e	g	l	o	r	s	t
LEET	4	8	3	6	1	0	2	5	7

The resource utilization is shown in Figure 14. The two additional rules required 12% more of logic resources, 5% more of registers and 3% more of embedded RAM blocks. This is clearly an excellent result if the potential gain is the twofold performance increase.

```
+-----------------------------------------------------------------+
; Estimated Resource Usage Summary                                ;
+--------------------------------------+--------------------------+
; Resource                          + Usage                   ;
+--------------------------------------+--------------------------+
; Logic utilization                   ;   38%                  ;
; Dedicated logic registers           ;   16%                  ;
; Memory blocks                       ;   23%                  ;
; DSP blocks                          ;    0%                  ;
+--------------------------------------+--------------------------;
```

Figure 14: Estimated resource utilization of the MD5 OpenCL kernel with 4 rules

The measured kernel time to process 10 million messages is 45.754 ms. Total execution time (including transfers) is 559.141 ms. Thus, the measured kernel throughput is (1,000 ms/45.754ms) * 10 million * 4 = 874.24 MH/s. This very promising result can be further increased if more rules are added.

5.5 Keccak Implementation (Dictionary Attacks)

Until now, attention was given to MD5 hashing algorithm. For the sake of comparison, we have decided to implement a much more complex algorithm SHA-3 (Keccak-256) (8). This algorithm accepts 1,088-bit long messages (i.e. rate = 1,088) and produces 256-bit hash value. In order to keep the same message length as in case of MD5, we assume that the password is no longer than 512 bits. The rest is padded according to the SHA-3 standard.

SHA-3 algorithm is a *sponge* construction, which absorbs the input message and then the hash is squeezed from it. Main component of the sponge is the *function F* that is composed of 24 rounds. SHA-3 operates on a state of 5 * 5 * 64 bits = 1,600 bits.

For the sake of simplicity, we have decided to disable rules. The resource utilization is given in Figure 15. Surprisingly, despite much higher complexity of the SHA-3 kernel, the resource utilization is relatively low. Only 29% of the logic resources, 10% of registers, and 16% of embedded RAM blocks are utilized by SHA-3.

```
+--------------------------------------------------------------+
; Estimated Resource Usage Summary                             ;
+--------------------------------------+-------------------------+
; Resource                     + Usage                 ;
+--------------------------------------+-------------------------+
; Logic utilization            ;   29%                 ;
; Dedicated logic registers    ;   10%                 ;
; Memory blocks                ;   16%                 ;
; DSP blocks                   ;    0%                 ;
+--------------------------------------+-------------------------;
```

Figure 15: Estimated resource utilization of the SHA-3 (Keccak) OpenCL kernel

The kernel processed 10 million 512-bit input messages, compared the hash value with the reference hash and send back the results in 58.310 ms. The total time including transfers was 568.308 ms. The longer kernel execution time can be related to lower clock frequency of the complex kernel or stalls or bubbles in its pipeline. Thus, the measured kernel throughput equals to (1,000 ms/58.31ms) * 10 million * 1rule = 171.50 MH/s. All in all, this is an excellent result, and since only 29% of logic is utilized, there is still space for implementation of rules. Thus, even higher performance could be reached.

6. Kernel Modifications for Brute-force Attacks

The previous chapters (except for the MD5 implementation in VHDL) assumed that attacks are conducted using a dictionary. This way, a dictionary is loaded into DDR3 memory modules on the Nallatech card, kernel computes all hashes, compares them with the reference hash value and returns the result. If the dictionary is larger than the size of the DDR3 memory, it can be divided into several parts and the kernel can be executed for each of these parts separately. However, in such case a dictionary part can be transferred into the DDR3 only when to computations with the previous parts are finished. Thus, the total computation time (i.e. kernel time + DDR3 memory transfers) is multiplied by the number of dictionary parts, leading to much slower attacks. In order to address this issue, we have decided to examine the brute-force attacks implementation space.

Unlike dictionary attacks, brute-force attacks do not require transfers of large dictionaries, because all tested messages can be generated directly inside the kernel. The kernel message generator requires a seed that can be sent to the kernel from the PC. Thus, the seed and the result status are the only data that must be exchanged between the kernel and the PC. As a consequence, total attack time is very close to the kernel time and the attack speed is much higher.

6.1 ASCII Password Message Generator

Similarly to the MD5 VHDL design, the OpenCL brute-force kernel implementation receives a seed from the host which is loaded into the LFSR-based generator of input messages. The input message is translated into an ASCII string which is subsequently processed by the MD5 function. The resulting hash value is compared with a reference hash. If the value does not match, the LFSR is incremented and a new MD5 computation and hash comparison is carried out. This way, the password cracker continues either until the hash is found or until the defined problem space is fully explored.

6.2 MD5 Brute-force Implementation

The MD5 brute-force kernel is derived from the dictionary one. The complex dictionary loading is replaced by the aforementioned ASCII message generator initialized with an input message seed, and the problem space is limited for the testing purposes to 10 million messages. For the sake of simplicity, this kernel does not implement rules in a preliminary step.

Early size estimations shown that the password cracker requires only 21% of resources. The LFSR is very compact, and so its impact on the resource utilization increase is negligible.

```
+-----------------------------------------------------------------+
; Estimated Resource Usage Summary                                ;
+-------------------------------------+---------------------------+
; Resource                            + Usage                     ;
+-------------------------------------+---------------------------+
; Logic utilization                   ;   21%                     ;
; Dedicated logic registers           ;    9%                     ;
; Memory blocks                       ;   15%                     ;
; DSP blocks                          ;    0%                     ;
+-------------------------------------+---------------------------;
```

Figure 16: Estimated resource utilization of the MD5 brute-force attack kernel

The kernel clock frequency read from the *acl_quartus_report.txt* file is 278.47 MHz. If the internal pipeline is fully used and if we assume that a new hash value is computed on each clock cycle, the maximum throughput reaches 278.47 MH/s. The hardware benchmarks reported the kernel time of **36.017 ms** for processing 10 million messages. The corresponding measured kernel throughput equals to (1,000 ms / 36.017 ms) * 10,000,000 = 277.65 MH/s. We see that this value is very close to the theoretical maximum. This proves the correctness of our assumption that the kernel computes a new hash value in each clock cycles and also that the pipeline is saturated. More interestingly, the total time of the computation (including data transfer from the PC to the FPGA and back) is only **37.407 ms**, which is much less than the total time for comparable dictionary attack MD5 kernel (680 ms).

6.3 MD5 Brute-force Implementation with 10 Rules

To further improve performance, we can multiply number of messages by adding rules. This approach is reasonable, because the message generator generates messages that do not contain uppercase ASCII characters. Thus, the additional rules can convert certain number of lowercase characters to uppercase. For the sake of simplicity, we've decided to add the following 10 rules:

RULE 1:	keep the message as it is (use directly by the MD5 algorithm)
RULE 2:	convert all lowercase letters to uppercase
RULE 3:	convert only the first lowercase letter to the uppercase one
RULE 4:	convert the first 2 lowercase letters to uppercase ones
RULE 5:	convert the first 3 lowercase letters to uppercase ones
RULE 6:	convert the first 4 lowercase letters to uppercase ones
RULE 7:	convert the first 5 lowercase letters to uppercase ones
RULE 8:	convert the first 6 lowercase letters to uppercase ones
RULE 9:	convert the first 7 lowercase letters to uppercase ones
RULE 10:	convert the first 8 lowercase letters to uppercase ones

The resource estimation is given in Figure 17. Utilization of logic resources reached 87% which is close to Altera's recommended maximum of 85%. Despite this high utilization of the FPGA, the clock frequency decreased only to 255.81 MHz.

```
+-------------------------------------------------------------+
; Estimated Resource Usage Summary                             ;
+-------------------------------------+-------------------------+
; Resource                            + Usage                   ;
+-------------------------------------+-------------------------+
; Logic utilization                   ;   87%                   ;
; Dedicated logic registers           ;   37%                   ;
; Memory blocks                       ;   41%                   ;
; DSP blocks                          ;    0%                   ;
+-------------------------------------+-------------------------;
```

Figure 17: Estimated resource utilization of the MD5 brute-force attack kernel with 10 rules

Using the estimated clock frequency, we calculate the estimated kernel throughput as follows: 255.81 MHz * 1hash/clock cycle * 10 rules = 2,558.1 MH/s. The measured kernel time to process 10 million messages using 10 rules reached 39.144 ms. The total computation time was 40.560 ms. The measured kernel throughput is (1,000 ms / 39.144 ms) * 10,000,000 * 10 rules = **2,554.67 MH/s**.

7. Comparison of FPGAs and GPUs

This chapter summarizes the results obtained for the OpenCL and VHDL implementations of the MD5 and Keccak hash functions. These FPGA results are compared with the GPU results published at the oclHashcat website (9).

7.1 OpenCL Implementation Results Summary

The results of the MD5 OpenCL implementations for dictionary attacks are summarized in Table 3. As can be observed the best results were obtained for the single-work-item programming model using 4 rules. This implementation could compute hash values of 874.2 million hashes per second. Since this kernel utilized only 38 % of logic resources, new rules can be added in the future to achieve higher performance.

For the sake of simplicity, there were only two Keccak OpenCL implementations. Despite their high complexity, the performance loss when compared to similar MD5 implementation was only approximately 20% without rules and 27% with 2 rules. As we will show later, these results are substantially much better than in case of the comparable implementations on GPUs. Moreover, the Keccak kernel with two rules occupies only 54% of logic resources and thus more rules could be used to increase the throughput. As we'll show later the throughput of both Keccak versions largely exceeds the throughputs achieved by most of the graphic cards.

Table 3: Summary of results obtained for dictionary attacks on MD5 and SHA-3 (Keccak-256)

	Functionality			**Resources [%]**			**Ex. time [ms]**		**Rate [MH/s]**	**DDR3 rate [GB/s]**
	Prog. model[4]	**# of rules**	**CMP**[5]	**Logic**	**Regs**	**RAMs**	**Kernel**	**Total**		
MD5 - simple[1]	MWI	1	no	20	9	16	39.9	680	250.7	20.21
MD5 - 4x vect. (attrib.)[2,3]	MWI	1	no	35	15	24	41.2	700	248.0	86.79
MD5 - 2x repl. (attrib.)[2]	MWI	1	no	27	11	20	162	790	63.0	40.42
MD5 - 2x rules[2]	MWI	2	no	24	10	18	50.45	807	405.9	24.25
MD5 - 2x rules + comp.[2,3]	MWI	2	yes	34	15	28	5031	5845	4.07	61.96
MD5 - task + cmp[1]	SWI	2	yes	26	11	20	47.15	588	424.2	N/A
MD5 - 2x int. vector[1]	SWI	2	yes	26	11	20	47.17	580	423.9	N/A
MD5 - 4x rules[1]	**SWI**	**4**	**yes**	**38**	**16**	**23**	**45.75**	**559**	**874.2**	**N/A**
Keccak 256 - simple[1]	SWI	1	yes	29	10	16	58.31	568	171.5	N/A
Keccak 256 - 2xrules[1]	**SWI**	**2**	**yes**	**54**	**18**	**42**	**64.5**	**65.9**	**310.1**	**N/A**

[1]Benchmark performed using 10 million messages
[2]Benchmark performed using 10.24 million messages
[3]Resulting hash were not correct
[4]Single Work Item execution (SWI), Multi Work Item execution (MWI)
[5]Comparison of hashed by the kernel

FPGA results for the brute-force OpenCL implementations are given in Table 4. The MD5 algorithm with 10 rules occupies 87 % of the FPGA logic resources and reaches the maximal throughput of 2,554.7 million hashes per second which corresponds to 1.22 Tib/s = 152 GiB/s assuming 512-bit messages!

Table 4: Summary of results obtained for brute-force attacks on MD5

	Functionality			**Resources [%]**			**Ex. time [ms]**		**Rate [MH/s]**	**DDR3 rate [GB/s]**
	Prog. model	**# of rules**	**CMP**	**Logic**	**Regs**	**RAMs**	**Kernel**	**Total**		
MD5 - simple + cmp[1]	SWI	1	yes	21	9	15	36.0	37.4	277.7	N/A
MD5 - 10x rules + cmp[1]	SWI	10	yes	87	37	41	39.1	40.5	2554.7	N/A

[1]Benchmark performed using 10 million generated messages

7.2 Third-party GPU Implementations' Performance Comparison

One of the most known hash cracking software implemented on GPUs is oclHashcat(9). This software can perform both dictionary and brute-force attacks and supports more than 100 different hashing algorithms. For the sake of comparison, we summarize in Table 5 our performance results of MD5 and Keccak for FPGAs as well as those for GPUs (for five different PCs) published on the oclHashcat web site (9).The results of the MD5 and Keccak implemented on GPUs are presented for five different PC configurations listed in Table 6. Unfortunately, oclHashcat authors do not specify how the results were obtained and if these results are for dictionary or rather brute-force attacks.

Table 5: Summary of results obtained for MD5 and Keccak implemented on GPUs and FPGAs

		FPGA OpenCLimpl.				VHDL impl.		GPUs OpenCLimpl. [1]				
		1x rule	2x rules	4x rules	10x rules	1x rule	12x rules[2]	PC1	PC2	PC3	PC4	PC5
MD5 [MH/s]	Dict.	250	424	870	---	---	---	8,089	2,414	10,742	1,838	81,549
	Brute f.	277	470	---	**2,555**	490	**5,882**					
Keccak [MH/s]	Dict.	**171**	**310**	---	---	---	---	142	94	175	59	1,714
	Brute f.	---	---	---	---	---	---					

[1]oclHashcat web site does not specify if the results are for dictionary or brute-force attacks
[2]estimated throughput

Table 6: PC configurations for benchmarking of MD5 and Keccak GPU implementations (9)

	Operating system	GPU		Specifications of one GPU card					
		brand	**GPU cards**	**stream proc. / cuda cores**	**core clock [MHz]**	**memory clk (eff.)**	**bus width**	**bandw. [GB/s]**	**Power [W]**
PC1	Ubuntu 13.04	AMD HD7970	1	2048	925	5500	384	264	250
PC2	Windows 7	Nvidia GTX580	1	512	772	4008	384	192	240
PC3	Ubuntu 12.04.3	AMDHD6990	1[1]	2x 1536	830	5000	2x 256	2x 160	375
PC4	Ubuntu 12.04.3	Nvidia GTX560Ti	1	448	732	3800	320	152	210
PC5	Ubuntu 12.04.3	AMD R9 290X	8	2816	1000	5000	512	320	290

[1]this GPU card contains two GPU cores

As can be observed the MD5 kernel running on the FPGA reached 2,555 MH/s performance using 10 rules. This result outperforms the oclHashcat running on the Nvidia cards (i.e. PC 2 and PC4). MD5 kernel throughput is 6% higher than in case of PC2 and 39% higher than PC4.However, the FPGA kernel could not reach the performance of the oclHashcat executed on the AMD cards (i.e. PC1, PC3 and PC5). In this case, the FPGA kernel reaches only 31% performance of PC1, 24% of PC3 and 3% in case of PC5.

The Keccak kernel mapped to the FPGA achieved the performance of 310 MH/s using 2 rules. Unlike in the case of MD5 kernel, this much more complex kernel outperforms oclHashcat running on all PCs except for the PC5, which contains 8 graphic cards. The kernel is 118% faster than PC1; 230% faster than PC2; 77% than PC3 and 425% than PC4! It is reaches only 18% of the PC5.

In order to understand why these results are so different, we must analyze the following aspects of these hardware platforms:

- Core frequency
- Memory bandwidth
- Flexibility
- Number of processing units
- Power consumption
- Cost

7.2.1 Clock Frequency

The clock frequency of the execution core is especially important when the executed algorithm has a sequential character. In this case, highly complex FPGAs cannot reach as high clock frequency as customized ASICs such as GPUs. Moreover, the maximum clock frequency of the FPGA implementations highly depends on their complexity and critical path length. As we have observed, the maximum frequency for our OpenCL FPGA kernels was inferior to 300 MHz (except the optimized VHDL design), which is far from GPU frequencies indicated in Table 6. This FPGA drawback can be, however, mitigated by the other aspects where FPGAs are significantly superior to GPUs.

7.2.2 Memory Bandwidth

Memory bandwidth is an important parameter when dictionary attacks are considered. These attacks require transfer of messages from/to the external DDR memories, and thus a too narrow bus interface can cause bottlenecks. On the other hand, memory bandwidth is not important in case of brute-force attacks, where external memories are used only to transfer the resulting hash value (not more than 256 bits).

The memory bandwidth is highly dependent on the PCB card architecture and also by the GPU/FPGA bus interface width. We worked with the Nallatech OpenCL 385 - A7 card which contains DDR3-1600 memory chips connected to the Stratix V FPGA via 72-bit bus interface. This configuration offers 25.6 GB/s memory throughput. However, GPU cards embed more powerful GDDR5 memories with much higher effective clock frequency, and bus larger buses (between 320 and 512 bits). This gives GPUs a significant advantage over the Nallatech card. However, a custom FPGA card could be developed that supports GDDR5s with much wider memory buses than 512 bits to close the performance gap.

7.2.3 Flexibility

FPGAs are highly flexible devices. Thus, an OpenCL kernel can be exactly mapped to hardware resources resulting in a highly parallel deeply-pipelined architecture. In the best case, each input message can be processed in only 1 clock cycle if the pipeline is saturated.

However, GPUs contain a fixed instruction set and a much less pipeline stages. Thus, an OpenCL kernel executed on a GPU requires more instructions to perform certain operations (e.g. fixed rotations are for free in FPGAs, but require several instructions on GPUs). This leads to substantially much lower performance on GPUs per clock cycle, although it is partially compensated by much higher GPU clock frequency.

We may have observed that the performances of MD5 kernels are much lower when mapped to FPGAs than when executed on GPUs. This can be explained by the fact that MD5 can be efficiently executed using the GPUs instruction set. However, a complex algorithm like Keccak cannot benefit from the general GPU instruction set and so we can observe significant performance loses when compared to MD5. On the other hand, Keccak can be much more efficiently mapped to FPGAs, and so the performance losses are much smaller than in case of MD5. This fact is very important in our study as we've shown that one FPGA can beat GPUs with even two cores (PC3) and much larger memory bus widths.

7.2.4 Number of Processing Units

The main processing units are the so called stream processors in case of AMD GPUs or cuda cores in case of Nvidia GPUs. OpenCL kernel maps its work items to these processing units, and so the work is significantly parallelized. When comparing the results of the MD5 or Keccak cores, one may observe that performance of AMD cards is significantly superior to the Nvidia cards. Interestingly, this difference is reflected by the fact that Nvidia cards contain less cuda cores than the number of stream processors embedded in the AMD cards.

Unlike GPUs, FPGAs do not contain such processing units, but rather a configurable logic. However, this logic resource can implement very deep pipelines which emulate the parallelism typical for GPUs. Moreover, performance can be also multiplied by instantiating several such pipelines in parallel. This way, we can consider each pipeline as a powerful processing unit in a multi-pipeline architecture.

7.2.5 Power Consumption

The power consumption is often neglected when comparing performance of graphic cards with other devices. However, if we compare the number of MH/s per 1 watt of consumed energy the results will significantly change in the favor of FPGAs. This is caused by the fact that FPGAs hardly consume more than 20 watts[7], whereas GPUs at least 10 times more (see Table 6). For this reason, we propose to consider the power consumption aspect and show that FPGAs are not only very powerful devices capable of outperforming GPUs, but also a much greener solutions.

Table 7 shows the results normalized by the power consumption [MH/(s.W)]. The power consumption for normalization of results is indicated in Table 6 in case of GPUs and is fixed to 20 W in case of FPGAs (estimated using Altera PowerPlay early power estimator).

Table 7: Summary of results GPUs and FPGAs when considering the power aspect

		FPGA OpenCLimpl.				**VHDL impl.**		**GPUs OpenCLimpl.** [1]				
		1x rule	**2x rules**	**4x rules**	**10x rules**	**1x rule**	**12x rules**[2]	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**
MD5 [MH/(s.W)]	Dict.	12.5	21.2	43.5	---	---	---	32.4	10.06	28.65	8.75	35.15
	Brute f.	13.9	23.5	---	**127.8**	24.5	**294.1**					
Keccak [MH/(s.W)]	Dict.	**8.55**	**15.5**	---	---	---	---	0.568	0.392	0.467	0.281	0.739
	Brute f.	---	---	---	---	---	---					

[1]oclHashcat web site does not specify if the results are for dictionary or brute-force attacks
[2]estimated throughput

As can be observed, FPGAs can deliver significantly higher performance per 1 watt of energy than any GPU. The performance per watt of the MD5 kernel with 10 rules is **3.63** times higher than that of 8 GPUs in PC5. Moreover, a VHDL implementation of MD5 with 12 rules is **8.4** times faster than in case of PC5!

The comparison is even more positive when comparing the Keccak implementations. Keccak kernel with 2 rules can deliver **20.9** times higher performance than PC5 with 8 graphic cards!

We note that a two-core AMD HD6990 GPU card consuming 375 watts of energy can be replaced by 18 FPGAs consuming together the same amount of energy. These 18 FPGAs could delivering up to **4.5** times higher performance (i.e. 2,555 MH/s * 18 = 45,990 MH/s) than the HD6990 (i.e. 10,742MH/s) in case of MD5 OpenCL kernel, **9.86** times in case of MD5 VHDL core (i.e. 5,882 * 18 = 105,876 MH/s) and **31.9** times in case of Keccak OpenCL kernel (310 * 18 = 5,580 MH/s)! These results are significant and underline the possible huge

[7] FPGAs operate on much lower clock frequency than GPUs or CPU, perform only the dedicated operations and the unused part of the circuit stays inactive. Unlike FPGAs, CPUs and GPUs are much more complex, execute other operations in the background (multitasking and operation system related).

advantages of FPGAs over GPUs when power consumption is an issue, as for example in the case of very large High Performance Computing (HPC) infrastructure.

7.2.6 Cost

The cost is not easy to estimate, since FPGAs are not end-user devices like GPUs. Thus, FPGAs are produced in smaller series than GPUs, and so their prices are higher. Moreover, a price of a complete PCB solution like the Nallatech 385 A7 OpenCL card is higher than those of GPU cards, because of small-series production. However, the FPGA market is growing which may imply a decrease of the prices in the coming years.

8. Future Perspectives

This study explored many ways to increase performance of two hashing algorithms implemented in OpenCL and mapped into Altera Stratix V FPGA. As we mentioned before, there are still many unanswered questions and interesting tracks that need to be explored. We list some of them:

- More hashing algorithms should be ported to FPGAs to test their performance and compare FPGAs and GPUs more fairly.

- More rules can be added to Keccak to fill the FPGA to 85% and also a brute-force implementation of Keccak can be designed.

- Better performance could be achieved using "aoc -O3" compiler switch. Thus, the design is much more optimized, but requires also more time to compile.

- Generated Quartus II projects require in-depth examination to understand properly how Altera OpenCL compiler transforms the code to Verilog. Proper understanding of this process may clarify problems we had experienced, and lead to better design choices when writing the OpenCL code. Unfortunately, Altera does not provide so far any documentation about this aspect.

- The MD5 core developed in VHDL could not communicate with the PC via PCIe. Also it could not access the external DDR3 memory banks. Thus, such a PCIe - DDR3 - core subsystem should be designed in the future to test the performance of the VHDL highly optimized cores and compare them with the OpenCL kernels fairly. This also allows better evaluating the performance gap between VHDL and OpenCL.

- The dictionary attacks without rules suffered the insufficient memory bandwidth. It would be interesting to test the same projects on different OpenCL FPGA cards (with potentially higher bandwidth) and compare their results.

- We noticed that DSP blocks were not utilized in any of the kernels. Altera does not provide any compiler directives or other means to instruct compiler to embed more DSPs. This has to be examined, because high-speed DSP could significantly increase performance if implemented correctly.

9. Design Recommendations

Although many design recommendations have been mentioned in the previous chapters already, we give a summary of the most important ones:

- Kernel source cannot be loaded into an FPGA directly (like in the case of GPUs), but must be compiled first and only then the resulting binary can be configured to an FPGA.

- We recommend using the single-work-item programming model for FPGAs, if the number of input/output work items is not the same. This way, better optimizations can be achieved.

- Performance can be increased by unrolling[8] predictable loops. Unrolling can be specified by a compiler directive "#unroll 4", where 4 is the degree of unrolling. Alternatively, a degree of unrolling does not have to be specified (i.e. using "#unroll"). Thus, compiler decides what the most suitable unrolling degree is.

- Unpredictable or data-dependent loop exit conditions should be avoided if possible to improve kernel optimizations.

- It is recommended to carry out the kernel size estimation using "aoc -c" command before performing full compilation. The size estimation is very fast and can help programmer to fine-tune the design as well as detecting design mistakes in a prior stage.

- Kernel should not exceed 85 % of the FPGA logic resources to simplify placement and routing and also to shorten the compilation time. In the worst case, larger kernels may also fail to compile.

- Portability of the GPUs code is not always guaranteed because of various limitations of the Altera OpenCL compiler. The list of these limitations can be found in (5). However, we expect that Altera will keep adding these features into next versions of the compiler, so that an OpenCL code could be ported from GPUs or CPUs to FPGAs directly without modifications.

- More optimizations and recommendations are detailed in the Altera OpenCL Optimization guide (10).

10. Conclusions

This document reports on the implementations of the MD5 and Keccak hashing functions in OpenCL and also the MD5 optimized version described in VHDL. The work was divided into 5 tasks that were all successfully accomplished.

It has been shown that OpenCL implementation of MD5 algorithm can reach the throughputs of 2,554.7 million hashes per second, thus outperforms two PC configurations

[8] Loop unrolling is the technique to replicate the hardware normally executed several times inside a loop and chain these replications to create a pipeline. This technique is used to increase performance at the expense of additional hardware resources

out of five. More interestingly, the optimized VHDL implementation could operate on a much higher clock frequency, resulting in the throughput of 5,888 million hashes per second.

OpenCL implementation of Keccak with dictionary attacks could compute 310 million hashes per second thus effectively outperforming all PC configurations except for PC5 which contains 8 graphic cards.

It has been shown that MD5 algorithm can reach high performance on GPUs because of its simplicity, and thus can outperform OpenCL MD5 implementations on FPGAs. On the other hand, Keccak algorithm is much more complex. We have shown that Keccak computations are much faster on FPGAs than on GPUs.

When considering the power consumption aspect, FPGAs clearly outperform GPUs for both MD5 and Keccak algorithms. The best MD5 OpenCL implementation with 10 rules delivers 3.63x higher performance per watt than the best PC5 system with eight R9 290X GPUs. Moreover, MD5 VHDL implementation with 12 rules delivers 8.4x higher performance per watt than PC5! The performance advantage of FPGAs is even more obvious in case of Keccak with 2 rules, where it delivers 20.9x higher performance per watt than the best PC5 with 8 graphic cards.

In order to stimulate further research, we listed possible future steps and explain what needs to be understood for better kernel optimizations or comparisons between FPGAs and GPUs.

In the end, guidelines for design of OpenCL FPGA projects were provided.

References

1. **Nallatech.** *PCIe-385N-A7 - Altera Stratix V FPGA Computing Card for OpenCL.* [Online] 2014. http://www.nallatech.com/images/stories/product_brief/pcie_385pb_v1_2.pdf.

2. **Rivest, Ronald.** *The MD5 Message-Digest Algorithm.* 1992.

3. **Steube, Jens.** *oclGaussCrack.* [Online] 2014. http://hashcat.net/oclGaussCrack/.

4. **GReAT.** *The mystery of the Encrypted Gauss Payload.* [Online] 08 14, 2012. http://www.securelist.com/en/blog/208193781/The_Mystery_of_the_Encrypted_Gauss_Payload.

5. **Altera.** *Altera SDK for OpenCL - Programming Guide.* [Online] 2014. http://www.altera.com/literature/hb/opencl-sdk/aocl_programming_guide.pdf.

6. *Ultra high throughput implementation for MD5 hash algorithm on FPGA.* **Yuliang Wang, Qiuxia Zhao, Liehui Jiang, Yi Shao.** s.l. : Springer, 2010.

7. **Wikipedia.** Leet speak. [Online] 2014. http://en.wikipedia.org/wiki/Leet.

8. **NIST.** *(DRAFT of SHA-3 Standard) FIPS 202: Permutation-Based Hash and Extendable-Output Functions.* [Online] 2014. http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf.

9. **Steube, Jens.** *oclHashCat.* [Online] 2014. http://hashcat.net/oclhashcat/.

10. **Altera.** *Altera SDK for OpenCL - Optimization Guide.* [Online] 2014. www.altera.com/literature/hb/opencl-sdk/aocl_optimization_guide.pdf.

Appendices

A. Description of Tasks

The work that was planned in the appointment letter and that is reported in this document consists of the following tasks:

Task 1: Introduction

- *Time and location*: Half a day in Ispra (Italy)
- *Purpose*: Meet the team; receive a first presentation of the hardware and software as well as an explanation of the specifications for the final product.

Task 2: First Implementation of oclHashCat

- *Time and location*: One day and a half in Ispra.
- *Purpose*: The task consists of the migration and test of oclHashCat on an FPGA board, leading to a first implementation of the software of oclHashCat. The software being already developed in OpenCL, this first migration should be trouble-free. The implementation should already be fine-tuned as much as possible, in particular via the OpenCL SDK software. At the end of this task, the performance test for the first implementation will be done together with the JRC scientific officer. The code of this first implementation will also be extracted from the FPGA.

Task 3: Optimization of the first implementation

- *Time and location:* Four days, expert's premise.
- *Purpose:* An optimized version of the first implementation has to be developed directly in VHDL (or Verilog). Some preliminary testing on the FPGA board can be remotely conducted with the help of the JRC scientific officer. The second and new version has to be provided to the JRC together with a report explaining the optimization implemented and providing a first evaluation of the quality of the code provided by the OpenCL SDK software.

Task 4: Second implementation of oclHashcat

- *Time and location:* One day and a half in Ispra.
- *Purpose:* The optimized version delivered in Task 3 will be migrated on the FPGA card and a second series of performance tests will be performed. The optimized version will be further fine-tuned during this process in order to enhance as much as possible the performance tests. The code of this second implementation will also be extracted from the FPGA.

Task 5: Synthesis and recommendations

- *Time and location:*Two days, expert's premise.

- *Purpose:* These last two days are dedicated to the redaction of the final report. The report shall contain the results of the two performance tests. Based on these results a comparative study of the two codes of oclHashcat will be realized illustrating the added value obtained. A global evaluation of the OpenCL SDK software has to be provided as well as some recommendations regarding the usage of such a tool.

B. Global Memory Bandwidth Analysis

The Nallatech PCIe385n A7 card implements two separate DDR3 memory banks. Each bank communicates with the FPGA via 64-bit wide data bus. This way, two DDR3 banks together can send 2 * 64b = 128 bits in parallel. DDR3 technology allows synchronizing data transfers with 4 time higher clock frequency than the base frequency and also exchanging data at both rising and falling edges of the clock signal. This way, eight 128-bit words can be sent in one clock cycle. According to specifications of the DDR3-1600 memory, the memory clock frequency is 200 MHz. This represents the maximum theoretical memory bandwidth of 128 * 8 * 200,000,000 b/s = 25,600,000,000 B/s = 25,600 MB/s = 24,414.0625 MiB/s.

European Commission
EUR 26833 EN – Joint Research Centre – Institute for the Protection and the Security of the Citizen

Title: FPGA performances in Cryptography

Author: Lubos Gspar

Luxembourg: Publications Office of the European Union

2014 – 32 pp. – 21.0 x 29.7 cm

EUR – Scientific and Technical Research series –ISSN 1831-9424

ISBN 978-92-79-40329-3

doi:10.2788/14648

LB-NA-26833-EN-N

JRC Mission

As the Commission's in-house science service, the Joint Research Centre's mission is to provide EU policies with independent, evidence-based scientific and technical support throughout the whole policy cycle.

Working in close cooperation with policy Directorates-General, the JRC addresses key societal challenges while stimulating innovation through developing new methods, tools and standards, and sharing its know-how with the Member States, the scientific community and international partners.

Serving society
Stimulating innovation
Supporting legislation

Publications Office

doi:10.2788/14648

ISBN 978-92-79-40329-3