FPGA-based Implementation Alternatives for Keyed-Hash Message Authentication Code in Networked Embedded Systems

Enrico Heinrich, Marian Lüder, Ralf Joost and Ralf Salomon
University of Rostock
College of Computer Science and Electrical Engineering
Institute of Applied Microelectronics and Computer Engineering
Email:{enrico.heinrich;marian.lueder;ralf.joost;ralf.salomon}@uni-rostock.de

Abstract

Field-programmable gate arrays provide a flexible and easy-to-configure implementation platform that supports the development of tamper-proof networked embedded systems. In order to achieve a secure mode of operation many of these systems employ keyed-hashed message authentication code to provide security against intentional tampering. Since this algorithm is of a high computational complexity, this paper utilizes various hardware-software co-design techniques for its implementation and compares the results to standard software. These techniques vary in the degree of required design expertise and the degree of how (software) functionalities are implemented in hardware logic.

1 Introduction

The general view on processing systems is that they consist of two different layers, hardware and software, which have different properties with respect to speed and flexibility. Designing a particular hardware involves the logic design, wiring and routing, as well as the production of the photo masks. It thus does not come to a surprise that Nvidia's state-of-the-art Smartphone processor APX 2500, for example, took more than 800 men-years of development. Therefore, hardware is a rather static layer. As a consequence, hardware designers aim to provide general-purpose designs that are as fast as possible.

Software, on the other hand, is much slower but much more flexible, and realizes the required functionality. Any modification can be easily done in an editor; after recompilation, the software provides the new functionality.

From a security perspective, however, software is problematic, since, after its deployment, it can be quite easily altered (compromised). Backdoors, viruses, worms, and Trojan horses are typical, well-known examples for modification attacks. In order to make sure that such attacks do not cause any damage, it is common practice [11] to install some non-modifiable, tamper-proof security control hardware between the network and the actual application host.

Due to the advancements in the design of digital systems, networked embedded (real-time) systems have received recent attention. In order to realize secure network communication, the implementation of proper encryption, decryption, and authentication modules is essential. Section 2 presents a brief description of a generic embedded system as well as the keyed-hash message authentication code (HMAC) [12].

By their very nature, networked embedded (real-time) systems have significantly increased demands with respect to mobility, performance, resources, and being tamper proof. In terms of configuration flexibility and integrity enforcement, field-programmable gate arrays (FPGAs) seem to be an ideal implementation platform. FPGAs represent a piece of hardware that can be fully configured according to the designer's desires. Another advantage of FPGAs is that the core or even all functionalities can be realized in hardware, and thus provides code protection to a large extent after deployment. Section 3 presents a brief overview about this technology.

This flexibility comes at the expense of runtime performance. State-of-the-art FPGAs allow a configured (soft-core) processor to be clocked at about 50 - 200 MHz, which might turn into a problem if aiming to realize real-time systems. Thus, Sections 3.2 to 3.4 discuss three options for the implementation of the message authentication functions in custom hardware and it also discusses how to seamlessly integrate this hardware into the entire system. Section 4 discusses how the integration of custom hardware affects the system's performance. It turns out that these enhancements improve the runtime by a factor of about 11 to three hundred. Finally, Section 5 concludes this paper with a brief discussion.

2 Message Authentication at the Firewall-on-Chip

The firewall-on-chip (FoC) [13] project aims at the development of an embedded security gateway. Among

other things, this gateway provides multi-level security [10] as well as improved real-time capabilities and tamper-proofness. The implementation of interaction security assumes that (1) the two interacting systems exchange messages that are encapsulated in (IP) packages and that (2) both sender and receiver have to pass all messages through their firewall-on-chip. For the enforcement of the system's security policy, a firewall-on-chip provides various security functions, such as authentication, access control, encryption and decryption, as well as integrity protection, for the packets it sends and receives. Within these security functions, the keyed-hash message authentication code (HMAC) plays a key role, for further details refer to [12].

Keyed-Hash Message Authentication Code: The concept of the hash function is a mathematical function that maps a string of arbitrary lengths to a fixed-length string, also called checksum, hash value, or message digest. In essence, the term "Message Authentication Code" (MAC) refers to cryptographic hash functions, such as MD5 [9], SHA-1 [3], SHA-256/224 or SHA-512/384 [4], that are able to sign arbitrary messages with an electronic signature.

Keyed-hash message authentication codes (HMACs) are MACs that in addition to the actual message, consider a second parameters, a secret key that is a shared secret between the originating sender and the intended receiver.

Fig. 2 shows that an HMAC works as follows: in the first step, the HMAC-algorithm converts the key to be exactly the block size of the iterated hash by adding zero bits to the end of the key. In the next step, it generate two derived keys, the inner (K_i) and outer key (K_o), by XORing each byte with 0x36 and 0x5C, respectively.

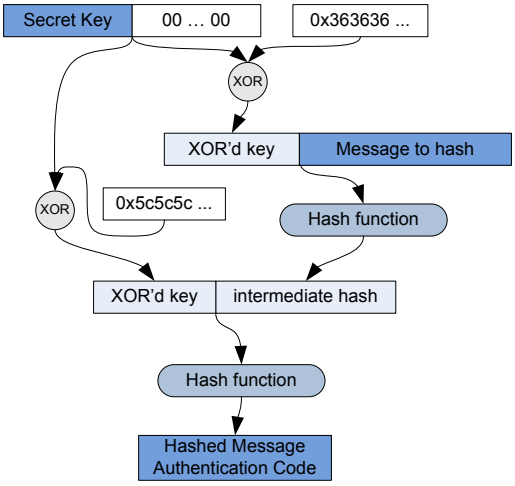

Figure 1. Keyed-Hash Message Authentication Code

After calculating the message digest that now depends on both the secret key and the actual message, the sender transmits the message along with its digest. The receiver, in turn, applies the same HMAC algorithm to both the received message and the shared secret key. The receiver considers the message correctly received, if and only if the calculated message digest matched the received one.

In mathematical terms, the digest of a message m can be defined as:

$$H\left(K_o, H\left(K_{i,m}\right)\right) \tag{1}$$

with H denoting the hash algorithm, K_i denoting the inner key, and K_o denoting the outer key.

3 Hardware-Software Co-Design Options in FPGAs

Field-programmable gate arrays (FPGAs) are general-purpose integrated circuits. An FPGA is programmed in a design process for which the designer has several options. This section briefly summarizes those ones considered in the present case study. This description is tailored towards the more experienced reader.

3.1 The Soft-Core Processor

The most and straight-forward design option constructs systems that consist of a CPU, some memory cells, as well as the required I/O ports. In the context of FPGAs, CPUs are normally called soft-core processors and are provided by all FPGA vendors as well as other sources. Normally, the soft-core processors are given in high-level hardware description languages. Thus, the system developers can tailor their processors towards the particularities of the software at hand. This includes, for example, the number of hardware registers, the width of the registers and bus systems, as well as the extension by means of other components.

Some well-known examples are the Nios and Nios II CPUs from [2], the LEON CPUs from [5], and the Microblaze CPU from [14]. The Nios II processor, for example, is shipped with the Altera Cyclone, Stratix, and Hardcopy device families. This 32-bit processor consumes about 700 – 1,800 logic elements and features up to 64 Kbytes cache, includes hardware interrupts, and hardware multiplication units.

Existing soft-core processors either execute standard operation codes or provide their own cross-compilers. This is very convenient for the developer, since existing (C) source code can be easily ported to the chosen processor. The advantages of easy (cross-) compilation and high configuration flexibility comes at the expense of significantly reduced execution speeds; the Cyclone II FPGA, for example, can clock Nios II soft-core processors at about only 85 MHz, which is about one and a half orders of magnitude slower than a standard PC.

3.2 The C2H Compiler

Altera offers an automatic software-to-hardware conversion tool, called C2H compiler. This compiler is capable of generating hardware modules based on given C

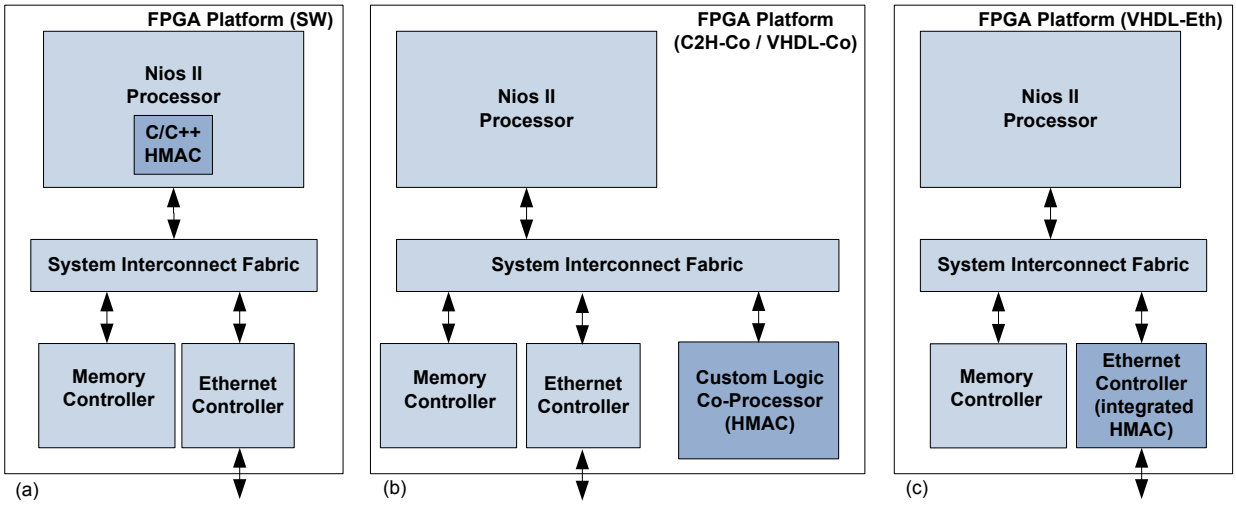

Figure 2. (a) FPGA platform with Nios II processor, (b) with a custom logic co-processor,(c) and with an integrated HMAC core in the Ethernet controller

source code. The usage of this compiler is quite easy. After choosing a C function, the C2H compiler generates a functionally equivalent hardware module and integrates it into the soft-core processor. This change in the system's hardware requires a re-synthesis of the FPGA-system; the compiler executes all relevant updates, which includes the configuration files as well as the software headers. Finally, the compiler replaces the body of the scheduled C function with a system call to the newly integrated hardware. Thus, from a software perspective no further changes are necessary. All the other parts of the software can make use of the "old" function as usual. The utilization of the hardware module is entirely "hidden" within the function.

In addition, the automated hardware generation for rather large C functions also imposes high demands on the C2H compiler: The compiler has to adhere to both data dependencies and timing constraints of the original C variables. Thus, complex C functions lead to complex hardware, i.e., hardware that requires many resources on the FPGA.

3.3 Hand-Coded Modules

An experienced hardware designer is not bound to all the restrictions that automatic tools, such as the C2H compiler, have to acknowledge. Rather, he/she can generate arbitrary functionalities at his/her own disposal. Given sufficient expert knowledge, the designer can apply almost any modification and/or extension to existing hardware modules and/or systems. Since these developments are done in a hardware description language, they require the regular design process. The integration of the manually designed modifications also requires, of course, some further software adaptations.

This paper considers the development of an authentication module, which is integrated into the existing system in two different ways. The first option realizes the module as a separate co-processor, whereas the second option tightly integrates this module into the network interface. From a system engineering point of view, the second option is much harder to realize, but offers higher performance improvements, since the communication overhead is minimal.

4 Implementation of HMAC and its Results

This section presents the architectures of four different hardware-software co-designs. The architectures differ in the amount of how much functionality has been realized in hardware as well as the resulting processing performance. The performance results have been achieved with Altera's low-cost Cyclone-II FPGA [1], which offers 33,216 logic elements.

The Nios II soft-core processor: The first, most straight forward design consists of a standard Nios II soft-core processor, which utilizes the 16MB DDR RAM as a conventional data and instruction memory. This design is presented in Fig. 2a, and serves as a reference point for all experiments presented in this paper.

In this design, the soft-core processor executes a monolithic software implementation of the entire HMAC algorithm [6]. The practical experiments have shown that this system can only be clocked at about 85 MHz. Even though this soft-core processors employs an instruction cache of 4 kilo bytes as well as embedded multipliers, the achievable throughput was as low as only 1,6 MBits/s. This is way too slow for a standard 100 MBit Ethernet connection. Furthermore, the execution of the HMAC algorithm does not leave any processing line for other task, such as encryption or other checks.

Automatically generated co-processor (C2H-Co): The first hardware alternative was developed by applying

Altera's C2H compiler to the computationally expensive `transform()`-function of the hash algorithm. The result is a separate hash module, which communicates with the soft-core processor by means of an existing on-chip communication bus. This communication bus is called Avalon. This design is presented in Fig. 2b, and denoted as C2H-Co for short.

This co-processor module accelerates the execution of the HMAC algorithm by a factor of about 11. However, this module consumes about four times as much resources as the rest of the entire system, and is a quite significant portion of the available hardware. But as has already been mentioned in Section 3.3, the C2H compiler can be used by almost any designer. The achievable throughput of 17,6 MBits/s is also not sufficient for a Fast Ethernet connection.

Hand-coded Co-Processor (VHDL-Co): This design option has realized the HMAC co-processor using the VHDL hardware description language. In so doing, the design has been tailored to given hardware architecture and has exploited some properties of the HMAC algorithm. This design approach is similar to the one already shown in Fig. 2b, and is denoted as VHDL-Co for short.

Due to the exploitation of several regularities as well as possible short cuts, this design accelerates the execution of the HMAC algorithm by a factor of almost 53, and requires only 12243 logic elements. In other words, this co-processor is about five times faster than the automatically generated co-processor and is only a half of its size. However, in order to be successful, this option requires advanced design experiences. The achievable throughput of 84 MBits/s is almost sufficient for a Fast Ethernet connection. This design alternative is the easiest way for checking the integrity of incoming packets at almost maximal Fast Ethernet speed.

Integrated HMAC Module (VHDL-Eth)[1] : Finally, the third design option directly integrates the hand-coded HMAC-module into the Ethernet controler. This tightly integrated module is placed between the Nios II processor and the network controller. This aproach avoids the data loop through the RAM and its controller. All IP packets can be checked and rejected, if necessary, before they reach the processor. This architecture is presented in Fig. 2c, and is called VHDL-Eth for short. This third design alternative yielded a four times higher performace improvement than the VHDL-Co design did. Even at a minimum frequency of 20 MHz, the HMAC core would be fast enough for a Fast Ethernet connection at full speed.

Summary and Discussion:

The presented data, depicted in Fig. 4, shows that hand-coded HMAC modules yield a speedup, which is about five to 25 times better than the ones achieved by using the C2H compiler. The data also shows that hand-coded solutions require about 50 % fewer additional logic elements. From a management point of view, these results are interesting in that the C2H compiler can be successfully used by almost any designer, where as the further improvements require substantial expert knowledge. However, if aiming at the design of networked embedded real-time systems, this additional training in a hardware description language, such as VHDL or Verilog, might be worth it. But in any case, both hardware-based implementations significantly improve the system's security, since the HMAC algorithm cannot be changed after deployment, at least not with reasonable efforts.

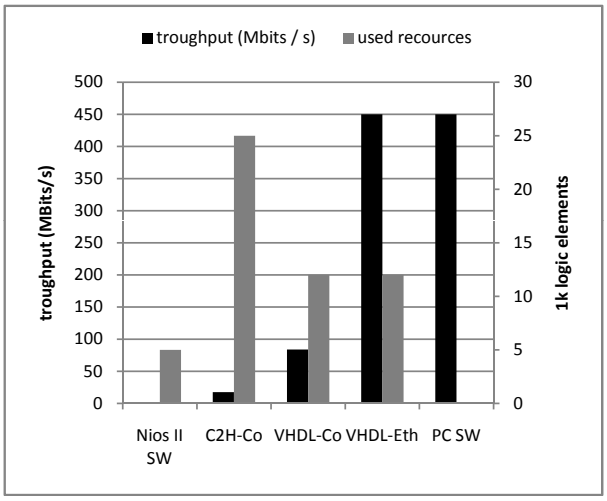

Figure 3. Throughput and resource requirements of the hardware implementations compared to the soft-core implementation

It might be mentioned that all implementations can be further improved by applying some additional optimizations as proposed by the vendors' guidelines. In addition, all hardware implementations can be further improved by using design concepts, such as parallel processing and code pipelining. However, since this paper focuses on using hardware-software co-design techniques, these design optimizations are beyond the scope of this paper.

5 Conclusion

This paper has discussed several options of how field-programmable gate arrays can improve the design of tamper-proof networked embedded systems. To this end, this paper has focused on the implementation of a security essential, the keyed-hash message authentication code (HMAC) algorithm.

From a security point of view, the main advantage of using field-programmable gate arrays is that after deployment, they cannot be altered (compromised) neither by internal nor by external attacks. From a developer's point of view, the usage of field-programmable gate arrays has

[1] These results have been obtained by means of the Modelsim simulation tool [7],[8], which provides very accurate performance figures that are normally close to actual hardware experiments. The reason for this approach was lack of time, which will be corrected in the final version of this paper.

the additional advantages that the design is almost as easy as developing a simple C program and that the designers have access to some excellent support tools.

The present case study has also discussed several options with which the designer can implement certain functionalities directly in hardware. The case study on the implementation of the HMAC algorithm has shown that the C2H compiler, which is very easy to use, yielded a speedup of about 11 but unfortunately increased the design size by a factor of about 5. By contrast, the direct usage of VHDL was able to accelerate the processing by a factor of about 50 to 280, which came at a cost of a doubled design size. This last design is able to process all incoming packets in real time.

The results of the present case study suggest that the C2H compiler provides good services if the following conditions are met: (1) A fair performance improvement by a factor of 10 to 20 is sufficient, which leads to a non real-time system in this case, (2) no VHDL expert knowledge is available, and (3) a rather instantaneous implementation result is required, i.e., short design-and-test cycle, and (4) the FPGA provides enough resources.

The implementation of the functionality by means of direct VHDL is useful under the following circumstances: (1) High performance improvements are required, (2) VHDL expert knowledge is available, (3) sufficient time for the design-and-test cycle is available, and (4) FPGA resources are plenty.

References

[1] Altera Corp. Nios Development Board Cyclone II Edition Reference Manual. Altera Document MNL-N051805-1.3, 2007.

[2] Altera Corp. Nios II Processor Reference Handbook. Altera Document QII5V4-7.2, 2007.

[3] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). Internet RFC 3174, 2001.

[4] Federal Information Processing Standards. Publication 180-2: Secure Hash Standard, 2002.

[5] Gaisler Research AB. LEON2 Processor User's Manual -XST Edition, 2005.

[6] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, 1997.

[7] Mentor Graphics Corp. ModelSim SE Reference Manual v6.3e, 2008.

[8] Mentor Graphics Corp. ModelSim SE User's Manual v6.3e, 2008.

[9] R. Rivest. The MD5 Message-Digest Algorithm. Internet RFC 1321, 1992.

[10] R. Smith. Introduction to Multilevel Security. In *Handbook of Information Security*, 2005.

[11] B. Snow. Four Ways to Improve Security. In *IEEE Security and Privacy*, pages 65 – 69, 2005.

[12] United States Federal Information Processing Standards. Publication 198: The Keyed-Hash Message Authentication (HMAC), 2002.

[13] Web Site FoC-project. http://www.imd.uni-rostock.de/index.php?id=262, 2009.

[14] XILINX Inc. MicroBlaze Processor Reference Guide. XILINX Document UG081 (v5.0), 2005.