A Simple Autonomous Reconfigurable Cryptographic Node

Oswin Horvath, Carlos Macián and Sylvain Stanchina

Abstract— **Current communication systems usually lack the flexibility to perform vast functional changes without expensive (in time, money and complexity) upgrades. The use of programmable logic in communication equipment provides the mixed advantages of software-like flexibility and high performance. A less well studied capacity of Programmable Logic Devices (PLDs) is the possibility of building fully autonomously reconfigurable systems that can adapt to changing needs and conditions. In this paper, an architecture for a simple autonomous reconfigurable node is presented that takes advantage of these features. To demonstrate its feasibility a prototype was implemented and tested for the case of Security Gateways in a VPN environment. Results show that a completely autonomous reconfiguration is possible and compatible with performance in the Gbps range.**

Keywords— **Reconfigurable Computing, Security Gateway, AES**

I. Introduction

In recent years, the performance of Integrated Circuits (ICs) — especially in the form of microprocessors — increased exponentially, while at the same time the price/performance ratio of ICs has dropped exponentially[1]. As a result, it became economically attractive to implement the central functionalities of network nodes in specialized hardware to satisfy the high bandwidth requirements posed by today's communication networks. There is one serious drawback when using static hardware, though: Lack of flexibility. To overcome this drawback, reconfigurable logic can be used, giving two main advantages:

Firstly, the requirements a system faces change over time, and a system based on reconfigurable logic can easily adapt itself accordingly. Consider as an example the processing of audio and video streams: The development of improved en- and decoders for such streams is the topic of many research projects and since there is a high need for better codecs in the market, the number of different standards that must be supported by a hardware solution increases quickly. Instead of having to manufacture a new ASIC to replace the old one in the multimedia processing systems whenever a new standard arises, it is much more convenient to update the existing system by loading a new configuration into a reconfigurable device that is used for the processing. And not only the new functionality can be added earlier to the system since the whole time-consuming ASIC manufacturing process can be omitted, but also the functional change itself takes only seconds.

Corresponding author: Carlos Macián. The authors are affiliated to the Institute of Communication Networks and Computer Engineering (IKR), University of Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart. E-mail: {horvath, macian, stanchina}@ikr.uni-stuttgart.de. Homepage: www.ikr.uni-stuttgart.de/en/~{macian, stanchina}

The other main advantage when using reconfigurable logic lies in the better usage of resources it offers: Looking at the multimedia example, in most cases only a limited number of codecs will be actually in use at any given time. In an ASIC-based solution, all codecs that must be supported have to be constantly present in the system, regardless whether they are actually used. Using a reconfigurable solution, a much smaller IC can be applied, e. g. only able to hold three different en-/decoder pairs simultaneously, thus using less chip area and power, an aspect increasingly important in today's power-hungry devices. Performance can benefit from using reconfigurable logic, too: When e. g an IC that is able to hold three en-/decoder at a given time has to process multimedia streams all using the same codec, the IC can instantiate three en-/decoder pairs supporting this codec, (ideally) tripling its computation power.

To fully exploit the potential of systems using reconfigurable logic, these systems additionally have to be able to *autonomously* reconfigure themselves, for the following reasons:

Firstly, not only the system should adapt to changing requirements, but this adaptation has to be done as quickly as possible in many applications — consider e. g. the aforementioned multimedia scenario again. However, if the system cannot react autonomously to changing requirements, but instead has to inform an external instance which then decides and initiates the reconfiguration process, an additional delay is added. In addition, the underlying communication network must be capable of transporting the messages exchanged between configuring and configured systems even when applying worst-case conditions: If the network gets overloaded — or non-functional for any other reason —, the configured system may become inoperative when requirements change although no error occurred within the reconfigurable system itself.

Secondly, autonomous reconfiguration allows easy synchronization between different reconfiguration events. Consider the case where many nodes of one network have to be reconfigured at exactly the same time — e. g. for the purpose of migrating from IPv4 to IPv6 where because of the huge number of nodes in many IP-based networks, manually reconfiguring all of them in short time is next to impossible. Autonomous reconfiguration provides an alternative in this scenario: Prior to the scheduled configuration time, short messages are given to the autonomous re-

configuring nodes telling them the exact time when to reconfigure.

In the next section, a simple though flexible architecture for an autonomous reconfiguring node will be introduced. In section III, a Security Gateway will be presented as case study for this architecture and in section IV a prototypical implementation of such a gateway will be described. Section V presents the main tools used for implementing the prototype, which will be further evaluated in section VI. Finally, a brief overview over previous work and a short summary of this paper will be given in sections VII and VIII, respectively.

II. Architecture

When designing the reconfigurable node architecture proposed in this section, care was taken to ensure that the architecture can be easily ported to a wide range of platforms and that the configuration process can be initiated by both an external controlling node as well as the configured node itself. The result is shown in fig. 1. The node consists of Node Control Logic (NCL) connected to an OAM (Operation, Administration, Maintenance) network via an OAM Interface (OAMI), Application Specific Logic (ASL) connected to a data network via one or more Network Interfaces (NI) and a Configuration Controller (CC) which can reconfigure both the NCL as well as the ASL and is connected to the OAM network, too, in order to be able to access the Configuration Data Base(s) (CDB).

The *NCL* — typically implemented as a microcontroller system — handles all control related tasks such as establishing new connections, reacting to external requests coming from other networks or users (using the OAM network to receive commands and send status informations) as well as controlling and supervising the other units of the node, i. e. the ASL and the CC as well as the various interfaces. The *ASL* consists of the main data processing units of the node and communicates with one or more data networks in order to receive/send the data that should be/has been processed. It can be composed of several independent modules that perform a variety of tasks, e. g. media-transcoding, protocol processing, encryption, etc. To add the reconfiguration ability to the node, a CC is integrated into the system, which locally stores several different configurations and can use any of them to reconfigure the NCL, the ASL or both. The CC can receive commands to do so from the OAM network (*externally triggered reconfiguration*) as well as from the NCL (*internally triggered reconfiguration*). New configurations are loaded on the CC from an external *CDB* via the OAM network.

III. Case Study: Security Gateways

The concept of Virtual Private Networks (VPNs) provides a way to connect two[1] or more distant lo-

[1]for simplicity, this discussion will concentrate on the case where only two local networks are connected to each other

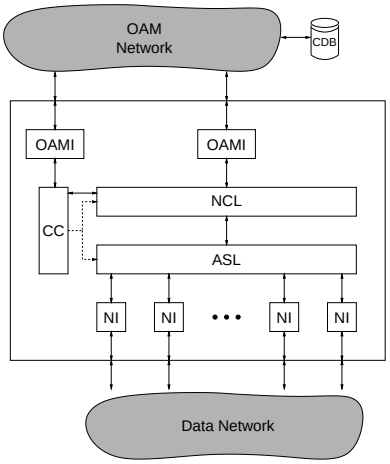

Fig. 1. Node Architecture

cal networks, merging them into a virtual common one. Packets sent from a client in one local network to a client in the other local network are transmitted using a third — typically public and insecure — network, e. g. the Internet. Packets from the LANs are encapsulated in a packet using a format supported by the intermediate network. To provide a secure transmission, data is not transmitted unmodified but instead encrypted when leaving the source network and decrypted when arriving at the destination network. This is the task of the sending and receiving Security Gateways which in doing so build a so called "tunnel" through the intermediate network. The Security Gateways also decide whether a packet has to be sent to a remote network at all or to a client in the same network. The whole process is transparent to the clients, only the routers know the real network structure. Fig. 2 shows a VPN using Security Gateways based on the described architecture — note that since the en/-decrypter performs no controlling tasks all of its functionality is part of the ASL.

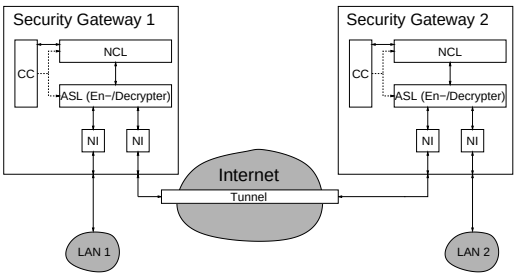

Fig. 2. A Virtual Private Network

Such a Security Gateway is especially suited to serve as a case study for the proposed architecture for the following reasons: Since secure encryption algorithms require a large number of calculations, implementing more and more of the corresponding functionality in hardware is highly desirable to increase performance of the Security Gateway. On the other hand, the encryption algorithm is likely to change over time. There are three main reasons for this: Firstly, the local network should be able to form a VPN with nu-

merous distant networks each one possibly using a different encryption scheme. While there are attempts to standardize the forming of VPNs across the Internet (namely IPSec), a variety of cipher algorithms will still be used, e. g. due to different demands regarding level of security vs. throughput. Secondly, a cipher may be vulnerable to a newly discovered cryptographic attack, rendering it instantly useless without prior warning. In this case it is imperative that the algorithm can be changed with minimum delays but also with a downtime of the server as short as possible. Lastly, it is advantageous to use encryption algorithms optimized for specific applications. For example it is possible to build a generic block en-/decrypter supporting all possible key and block sizes, but an en-/decrypter supporting the one specific key/block size combination that is used in the VPN at a given time will be in most cases more efficient considering speed as well as chip area. As was shown in the introductory section, reconfiguration offers an economical way of being able to use numerous different algorithms.

In addition, being able to *autonomously* reconfigure and thus being able to react quicker to changing requirements is specifically beneficial for a Security Gateway, since this way every time a new security association is established, the corresponding cipher can immediately be instantiated in a fully automatic way.

The encryption standard chosen for the implementation of this case study is the Advanced Encryption Standard (AES, [2]). The main reason for this choice was the high importance that AES will achieve in the future, since it has recently been approved by the NIST as the official successor of the Data Encryption Standard (DES). DES was discontinued because of security issues primarily concerning its short maximal key length of 56 Bit. The algorithm used in AES is Rijndael[3], a symmetric iterative block cipher[4] supporting block and key sizes of 128, 192 and 256 Bit, though only a block size of 128 Bit is applied when using AES. The basic principle of an iterative block cipher is to repeatedly apply a series of mathematical operations (a single such series is called round transformation or round) to the block that has to be en-/decrypted[2]. All rounds consist of the same types of operations executed in the same order, although their actual parameters may be different each round — e. g. the last operation in every Rijndael encryption round is a bitwise exclusive-or between the block to be encrypted and a so called subkey (various different subkeys are derived from the cipher key), but the chosen subkey changes every round.

IV. Prototype Realization

This section will describe the prototypical realization of the Security Gateway, first introducing the Universal Hardware Platform (UHP) on which it was realized [5], then giving an overview over the prototype, and finally concentrating specifically on the configuration mechanism.

A. Universal Hardware Platform

The UHP consists of up to nine different (one base, eight daughter[3]) boards, each one having one FPGA from Altera's Apex 20K series [6] and a set of Configuration EEPROMs of Altera's EPC device series[4]([7], see subsection IV-C) mounted on it. In addition, every board features an independent clock as well as four "Mezza"-slots and a serial port, all connected to the board's FPGA. On the Mezzas additional boards can be mounted to expand the functionality of the main board. The daughter boards connect to the base board via two independent buses.

B. Structural and Functional Description

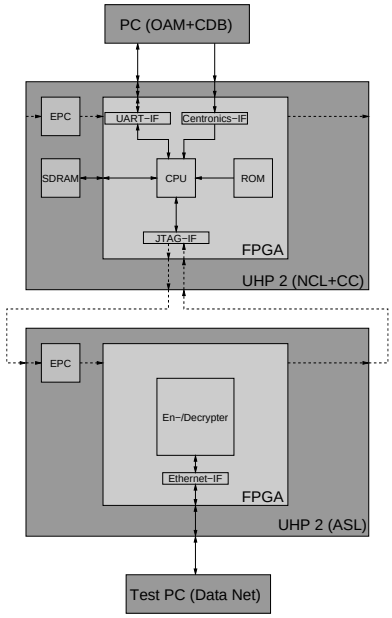

Fig. 3. Implementation

The complete implementation of the case study is shown in fig. 3. NCL and CC are integrated into one microcontroller system running a single program, the whole system being implemented on a single UHP 2. An ordinary PC is used to emulate the OAM network including a controlling external node and the CDB, while another Test PC emulates the Data Network. The en-/decrypter module is implemented on a separate UHP 2.

The reason that NCL and CC are merged is to enable them to share the OAMI and additional resources such as memory and CPU — this option can be used in most systems, since reconfiguration will occur relatively seldom and thus it is reasonable to use the resources of the CC to enhance the performance of the NCL when the functionality of the CC is not needed. The Apex 20K[5] on the NCL/CC board con-

[2] in most such ciphers — including Rijndael — there are slight modifications to this scheme

[3] called "UHP 1" and "UHP 2", respectively

[4] in the following they will be referred to as "EPCs"

[5] Altera EP20K1000ECB652-1

tains a Nios CPU, ROM and various interfaces. Small boards mounted in the Mezza slots are used to carry a Centronics and a JTAG port as well as 64 MB of SDRAM. The UART- and a Centronics-interface form the OAMIs of the system. The former is used for bidirectional communication to receive messages and send status informations, while the latter was added to provide a fast way of receiving large amounts of data, a task the UART-interface is not suited for because of its low bandwidth. On the other hand, the Centronics interface cannot replace the UART-interface since its implementation only supports unidirectional communication. To enable the microcontroller system to configure PLDs, a JTAG interface was added (see IV-C).

A bootloader is stored in the ROM to load the main program via the Centronics port after a reset. The reason this mechanism was chosen is that this way the program running on the system can be easily exchanged because the ROM content does not have to be updated for this purpose — since the ROM is part of the FPGA, this would require the whole FPGA to be reconfigured — and the ROM would consume most of the FPGA's available memory cells in this case.

The main program's task is to load and manage configurations, control the reconfiguration and communicate with the OAM network. In this simple implementation, configuration-related commands are given as ASCII characters via the UART-interface and configurations are loaded via the Centronics-interface.

Note that the presented prototype actually does not *autonomously* reconfigure itself, it only reacts to commands given via the UART-interface. This approach was taken because this work focuses on the reconfiguration itself, not on the application-specific conditions that may trigger the reconfiguration. However, modifying the system to support fully autonomous reconfiguration is a trivial task, since from the standpoint of the CC, there is no fundamental difference between externally and internally triggered reconfiguration: The only required change consists in modifying the main program, so that it does not react to messages coming from the UART-interface but instead observes some internal variables or messages from the NCL to decide when to reconfigure the ASL.

No modifications had to be done to the PC. It is connected to the NCL/CC via a standard serial and a standard parallel cable, data is sent to and received from the serial port by using a terminal emulator running on the PC, while data is sent to the parallel port with a simple file copy command.

The en-/decrypter module features the en-/decrypter core as well as a Fast Ethernet-interface (utilizing an Ethernet Transceiver[6] mounted on a board which is plugged into one Mezza slot) and is contained in the Apex 20K[7]. En- and decrypter use a block and key size of 128 Bit and thus require 10 rounds to process one block of data[3]. They are realized using a structure for the data path proposed in [8] (inner-loop pipelining with two stages). The structure of the encrypter is shown in fig. 4, the decrypter is very similar and for

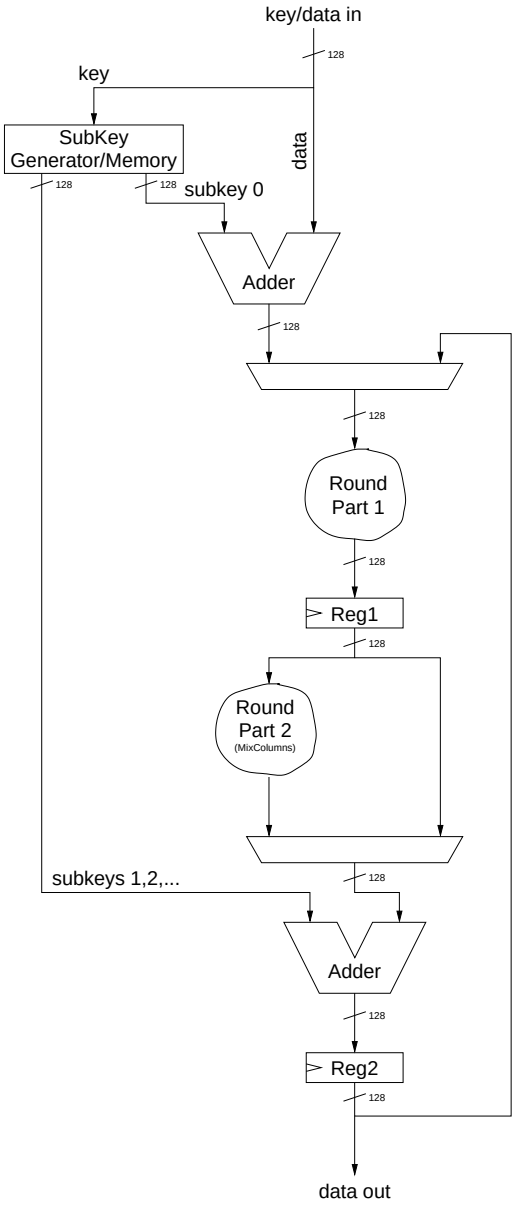

Fig. 4. Encrypter Structure

this reason not presented. Keys can be changed during operation without needing to reconfigure, the corresponding subkeys are calculated in-system. Of the five block cipher modes of operation proposed in [9], this implementation only supports Electronic Code Book Mode, i. e. every single block is en-/decrypted independently of the others. However, this mode is vulnerable to codebook attacks, since a given block/key combination always leads to the same output block. The modes Cipher Block Chaining, Cipher Feedback and Output Feedback avoid this disadvantage, in that their output depends not only on the output block of the current input block but also on those of the previous input blocks. However, this property renders these modes unsuitable for a pipelined implementa-

[6] AMD Am79C875 NetPHY-4LP Low Power Quad 10/100-TX/FX

[7] Altera EP20K400CB652C7

tion like the one used in this work. Instead, Counter Mode, the fifth mode of operation, can be chosen in future extensions as a way of improving security while at the same time maintaining high performance: In this mode, the Rijndael algorithm is not applied directly onto the input blocks but instead onto an series of internally generated constantly changing counter values and the results are then exclusive-ored to the input blocks to produce the output blocks.
Though in principle it would be possible to process two data blocks in only 20 cycles (10 rounds times 2 pipeline stages) when using the presented structure, this implementation uses 21, since new blocks are not fully concurrently shifted in while the old ones are shifted out. Therefore, if f_{max} denotes the maximal possible clock frequency, the maximal throughput rate is[1]:

$$r_{max} = 128Bit \times \frac{2}{21} \times f_{max} \quad (1)$$

C. Configuration Mechanism

A simple way of configuring Programmable Logic Devices rests on utilizing the IEEE 1149.1 protocol and bus[10]. Originally designed to allow the testing of integrated circuits via a standardized serial test bus (the "JTAG bus"), many PLDs from vendors like e. g. Altera and Xilinx can additionally be configured using the JTAG bus. To connect a device to the bus, it has to be included in a serial chain (called "JTAG chain") of devices which starts and ends at the JTAG controller ports (see fig. 5). This way, an arbitrary number of devices can be tested/configured by a single controller.

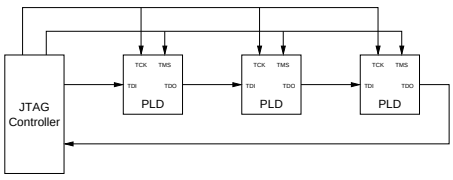

Fig. 5. Sample JTAG Chain

In this prototype, the CC posseses an JTAG port for configuring the ASL and uses JAM STAPL[8] programs as configuration data format. JAM is an interpreted language resembling BASIC and developed by Altera as a way to specify signal patterns on the JTAG bus — this becomes possible because besides usual instructions for e. g. manipulating variables in memory, there are some special ones defined that cause the interpreter to directly access the JTAG port of the system running the interpreter. JAM has become a JEDEC standard (JESD-71) and JAM programs for the purpose of configuring can be generated by many PLD vendor tools.

Configurations are normally not used to directly program the FPGA, instead they are loaded in advance via the JTAG bus onto the EPCs. Afterwards, the downloading of the new configuration to the FPGA can be triggered at any time by the CC. This way, the time between reconfiguration request and completion is shortened to a few milliseconds, since using the EPCs to reconfigure the FPGA is orders of magnitude faster than directly using JTAG, though transmission of configuration data to the EPCs via JTAG is still slow. For this reason, a caching mechanism should be used: The configurations most likely needed in the future are loaded onto the EPCs prior to any reconfiguration request, so that they can be applied in very short time afterwards, while configurations less likely to be used stay in the CC's large memory where they can still be used for programming the FPGA, however with a heavy time penalty. But even in this case, the downtime of the Gateway still consists of only a few milliseconds, since normal operation of the FPGA is sustained until the final stages of the reconfiguration process.

V. Tools

For the implementation of the NCL/CC, Altera's Excalibur-Nios-Kit was used [11]. This package consists of numerous soft cores commonly needed to design a computer system, tools to integrate these and custom components into one embedded system and a software development kit based on the GNU development tools which allowed the main program running on the NCL/CC to be written in C. To be able to use JAM-Programs, Altera's JAMPlayer[9] was integrated in the main program. Soft cores used are CPU "Nios", bus "Avalon" and the UART-interface. Additional components were designed using HDL-Designer Pro 2001.5b [12], namely the JTAG-, Centronics- and Ethernet-Interfaces as well as the AES en- and decrypter. All logic was simulated using ModelSim V5.6 [13], synthesized using Leonardo Spectrum V2001_1b [14] as well as placed and routed by Altera's Quartus 2.0 [11] which was also used for timing simulations.

VI. Evaluation

A. Validation

To ensure proper operation of the reconfigurable Security Gateway, following scenario was used (refer to fig. 3 for the test setup): At first, the AES en- and decrypter configurations were transmitted via the Centronics port to the Security Gateway and stored in the NCL/CC's memory. Then, after the Gateway was signaled to configure the ASL as an encrypter, the Test PC sent a key K followed by a series of data blocks D to the Gateway, and at the same time received and stored the (encrypted) data blocks E that came back. After that, the Gateway was signaled to configure its ASL as a decrypter and the Test PC sent K again, but this time followed by the previously stored Data Blocks E. As one would expect, the blocks that were now received were identical to the original series of blocks D, showing that the reconfiguration process

[8] Standard Test And Programming Language

[9] a free JAM interpreter [11]

TABLE I
RESOURCE USAGE

Entity	Logic Elements	RAM Bits	Device
NCL/CC	3661/38400 (9.5%)	83968/327680 (25.6%)	Altera EP20K1000ECB652-1
Encrypter	4997/16640 (30.0%)	79872/212992 (37.5%)	Altera EP20K400CB652C7
Decrypter	5725/16640 (34.4%)	79872/212992 (37.5%)	Altera EP20K400CB652C7

worked and the decrypter performs the inverse operation of the encrypter.

Conformance of the en-/decrypter with the AES was separately validated by sending series of keys and blocks and comparing the received blocks with known correct results.

B. Performance

As soon as the correct next configuration is loaded into the EPCs, the FPGA can be configured in a few milliseconds. Contrarily, the direct reconfiguration of the FPGA via the JTAG bus can take — depending on the FPGA type and number of devices in the JTAG chain — up to eight minutes. Thus, this second reconfiguration method only makes sense as backup solution and the EPCs should always be used as an intermediate step.

A clock rate of 50MHz was used for the whole En- and Decrypter-Module since the Ethernet Transceiver mounted on the same UHP requires a clock with this clock frequency. However, since the maximal clock frequency is $f_{max} = 102\,MHz$ for the encrypter and $f_{max} = 86\,MHz$ for the decrypter, throughput rates of more than $1.2\,\frac{GBit}{s}$ and $1.0\,\frac{GBit}{s}$ can be obtained according to eq. 1. Since the implementation was primarily designed for the purpose of testing the Configuration Controller, only a short time was taken to optimize the design. However, since the critical path lies in the data path and can easily be shortened by adding more pipeline stages, further throughput improvements are possible. The maximimum throughput rates could not be verified, though, since the maximum throughput rate of the Fast Ethernet Interface is too slow to allow transmitting enough data blocks per time unit to and from the en-/decrypter.

The resource usage (used resources/total available resources) of the NCL/CC and the en-/decrypter (including the Ethernet interface) is denoted in table I.

VII. PREVIOUS WORK

The Rijndael algorithm has already been implemented using both ASICs[15] as well as FPGAs[8], [16]. However, these realizations concentrated on performance evaluation and optimization and didn't consider the issues connected to reconfiguration.
A thorough examination concentrating on the use of FPGAs in reconfigurable systems can be found in [17]. However, its emphasis does not lie in autonomously reconfiguring systems.

VIII. CONCLUSIONS

In this paper, a simple architecture for an autonomous node was presented, introducing the concept of an embedded Configuration Controller. Systems applying this proposal can quickly adapt themselves to changing conditions and thus are able to increase both performance and resource usage. As a case study, the practically important application of an autonomous reconfiguring cryptographic node in a Virtual Private Network was examined, implemented and validated.

REFERENCES

[1] John L. Hennessy and David A. Patterson, *Computer Architecture - A Quantitative Approach*, Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, third edition, 2002.

[2] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication (FIPS PUB) 197, Nov. 2001.

[3] J. Daemen and V. Rijmen, "AES proposal: Rijndael," NIST AES Proposal, June 1998.

[4] Lars R. Knudsen, "Contemporary block ciphers," in *Lectures on data security: modern cryptology in theory and practise*, vol. 1561 of *Lecture Notes in Computer Science*, pp. 105–126. Springer-Verlag, Berlin Germany, 1998.

[5] "A Universal Hardware Plattform (UHP) for rapid prototyping of network nodes," IKR, University of Stuttgart, Internal Report, Sept. 2002.

[6] "Apex20K Programmable Logic Device Family Data Sheet," Feb.2002, Ver.4.3, www.altera.com/literature/ds/apex.pdf.

[7] "Configuration Devices for SRAM-Based LUT Devices Data Sheet," Dec.2002, Ver.12.2, www.altera.com/literature/ds/dsconf.pdf.

[8] Kris Gaj and Pawel Chodowiec, "Comparison of the hardware performance of the AES candidates using reconfigurable hardware," in *The Third Advanced Encryption Standard Candidate Conference, April 13–14, 2000, New York, NY, USA*, Gaithersburg, MD, USA, 2000, pp. 40–56, National Institute for Standards and Technology.

[9] National Institute of Standards and Technology (NIST), "Recommendation for block cipher modes of operation," NIST Special Publication 800-38A, 2001.

[10] "IEEE standard test access port and boundary-scan architecture," IEEE 1149.1.

[11] "Altera homepage," www.altera.com/.

[12] "HDL Designer Pro Homepage," www.mentor.com/hdldesigner/.

[13] "Modelsim Homepage," www.model.com/.

[14] "Leonardo Spectrum Homepage," www.mentor.com/leonardospectrum/.

[15] Patrick R. Schaumont, Henry Kuo, and Ingrid M. Verbauwhede, "Unlocking the design secrets of a 2.29 Gb/s Rijndael processor," in *Proceedings of the 39th conference on Design automation*. 2002, pp. 634–639, ACM Press.

[16] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists," in *The Third Advanced Encryption Standard Candidate Conference, April 13–14, 2000, New York, NY, USA*, Gaithersburg, MD, USA, 2000, pp. 13–27, National Institute for Standards and Technology.

[17] S. Hauck, "The roles of FPGAs in reprogrammable systems," in *Proceedings of the IEEE*, April 1998, vol. 86, pp. 615–638.

[18] Jinghuan Chen, Jaekyun Moon, and Kia Bazargan, "A reconfigurable fpga-based readback signal generator for hard-drive read channel simulator," in *Proceedings of the 39th conference on Design automation*. 2002, pp. 349–354, ACM Press.