Implementation of Cryptographic Applications on the Reconfigurable FPGA Coprocessor microEnable

H. Singpiel[1], H. Simmler[1], A. Kugel[1], R. Männer[1]
A. C. Castañon Vieira[2,3], F. Galvez-Durand[2], J. M. S. de Alcântara[2], V. C. Alves[2]

[1] *Department of Computer Science V*
University of Mannheim
B6, 26; D68131 Mannheim, Germany
(email: singpiel@ti.uni-mannheim.de)

[2] *Department of Electronics Engineering*
EE & COPPE/Federal University of Rio de Janeiro
P.O. Box 68564 21945-970 Rio de Janeiro, RJ Brazil
(email: fico@lpc.ufrj.br)

[3] *Communications and Electronics Department – GCE*
Brazilian Army Research and Development Institute – IPD
Av. Américas 28705, 23020-470, Rio de Janeiro, RJ Brazil
(email: castanon@ipd.eb.mil.br)

Abstract

Nowadays, higher demand and greater awareness on security problems lead to the study of more secure, high performance, reliable and flexible systems. To meet these demands the implementation of the Blowfish algorithm in the commercial FPGA coprocessor microEnable has been chosen to present the high performance of such FPGA based reconfigurable systems. In this paper we demonstrate, how such a system can be used to enhance the speed of cryptographic computation dramatically. We show that by using this FPGA design the Blowfish computation can be increased in speed almost by a factor of 10. The achieved results lead to the general conclusion that the use of an FPGA coprocessor is ideally suited for the execution of cryptographic algorithms regarding to execution time and flexible usage.

1. Introduction/Motivation.

In our new technological world, data security has and will become more and more a major issue. Examples are secure e-cash, e-commerce and video conferences [3]. At least there are two good reasons to use a reconfigurable system for the execution of cryptographic algorithms: Rapid Prototyping and Algorithm on Demand for the purpose of secure data transfer.

One of the most important factors in the fast-moving world of electronics industry is to shorten the time-to-market as much as possible. The usage of microEnable as a rapid prototyping system reduces the overall development time for electronic products dramatically. The emulation of a new circuit (ASIC) can start long before the chip is actually produced. The advantages you gain are the much higher speed of emulation (compared to simulation) as well as the start of the software development at the earliest possible time: reductions of the time-to-market up to several months are possible.

On the other hand, using high bandwidth and secure internet communication or broadcasting mechanisms in the near future means one must be prepared to support different kinds of cryptographic algorithms. Especially for the decryption/encryption of video data high performance systems are required. With the use of microEnable and different algorithm designs like the Blowfish algorithm, cryptographic Application on Demand becomes reality now.

In Section 1 the FPGA coprocessor microEnable is described. The blowfish algorithm is content of Section 2. Section 3 contains implementation details and results. Finally in Section 4 the overall results are discussed.

2. The microEnable coprocessor.

Co-processing is a close interaction of a high performance device – in this case the FPGA co-processor – with a host CPU in order to increase overall system performance. Effective co-processing needs a subsystem that can take over a significant portion of the algorithm, a communication path with a high bandwidth and means to describe and test the system as a whole. MicroEnable together with the VHDL and CHDL development tool-set provides all these features.

0-7695-0843-X/00 $10.00 © 2000 IEEE

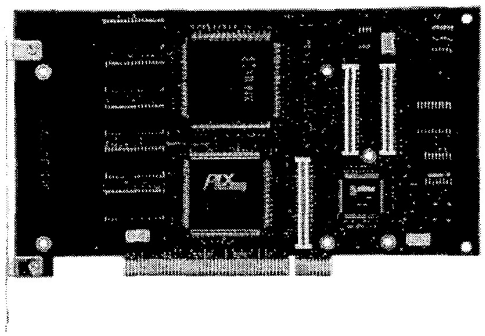

Figure 1. The FPGA coprocessor microEnable

2.1 Hardware

The microEnable is a commercial [2] FPGA processor, tightly coupled to its host computer via the PCI bus, comprising a single FPGA of the Xilinx XC4000 series and 0.5 to 2 Mbytes fast SRAM. Additionally it can carry CMC or S-LINK daughter cards. Device drivers for LINUX and WindowsNT make up to 125 Mbytes/s available to user applications via the PCI bus.

Although very flexible and powerful the basic hardware of microEnable is quite simple. It consists of the five functional units:

- one Xilinx XC4000 FPGA (from 4013 to 4085),
- one bank, either 500k or 2M Byte, of fast SRAM,
- a PCI-to-local-bus bridge PLX PCI9080,
- daughterboard connectors,
- clock and support circuitry.

The FPGA is microEnable's computing kernel. The computing power of microEnable is scalable by one order of magnitude by selecting an appropriate type of FPGA.

The RAM system is a fast buffer exclusively used by the FPGA. This memory is intended for tasks like temporary data buffer, lookup table, or coefficient storage.

The PCI interface supports the standard PCI bus of 32-bit width and 33 MHz.

A daughterboard according to either CMC or S-Link specification can be plugged onto microEnable allowing easy expansion with external interfaces or prototyping.

The clock and support circuitry of microEnable implements tasks like configuration and readback of the FPGA, a JTAG port, and provides user programmable clocks.

2.2 Software

The microEnable provides device drivers for the operating systems LINUX and Windows NT 4.0 with the latter one being more advanced.

The device drivers support master as well as slave accesses, two independent gather/scatter DMA channels and interrupt capability. In order to provide users a simple but fast programming interface, accesses to microEnable are memory mapped into user space.

Maximum throughput however can only be achieved if DMA transfers are used. Using DMA microEnable provides a performance of up to 125 MBytes/s (measured to/from a user application running in user space under WinNT 4.0 on a Pentium II/233). This is about 95% of the theoretical PCI performance! microEnable provides two DMA mode: In the standard scatter/gather DMA mode the PCI Bridge controls the transfers. In the DMA-on-demand mode, transfers are controlled via a simple handshake between the PCI Bridge and the FPGA. Using this mechanism transfer rates comparable to normal DMA rates are possible while protecting the co-processor from data over-/under-run.

2.3 Programming microEnable

VHDL is commonly used to describe logic-type FPGA designs. microEnable can be programmed using VHDL and this can be reasonable for applications where the on-board FPGA is mainly used for interfacing, prototyping or as a test-tool. To support this approach a VHDL library is available together with a set of API functions for communication and data-transfers. The VHDL library can be used with arbitrary VHDL synthesis and simulation tools. In this case the standard VHDL tools – timing simulators, testbenches, waveform displays etc. – must be used for design verification.

However VHDL supports only poorly applications that are distributed between the FPGA subsystem and the host CPU. There is no way to compile VHDL into code executable on the host microprocessor. Thus, testing of the VHDL code cannot be done by the host application itself. To efficiently realize the FPGA processor paradigm a set of tools is needed which support all aspects of hardware-software co-design. This is exactly the purpose why CHDL was developed at the University of Mannheim [1].

CHDL, considered as a hardware description language, certainly does not have all of the features available with a complete VHDL system yet. In particular, behavioral design description and exact timing simulation. However CHDL builds upon a C++ class library which implements all basic building blocks (primitives) present in a specific FPGA family, plus a library[1] of macro functions – counters, adders, multiplexers, memories etc. – which are built from the primitives, plus a simple state-machine description method. Two main features characterize the advantages of CHDL: Only one single language, C++, is sufficient to manage the whole development process. In both

[1] CHDL uses device independent higher-level macros built upon device specific primitive libraries which are currently available for the FPGA families Xilinx XC4000 and Virtex, Atmel AT40k and Lucent Orca 3T.

the application and the hardware description the features of the powerful programming language are used. Second a high correspondence between simulation and real time operation allows complex errors to be found faster and therefore decrease developing time.

3. The blowfish algorithm.

Blowfish is a symmetric variable-length key, 64-bit block cipher cryptographic algorithm developed by Bruce Schneier [5]. It has been one of the most secure secret key cryptograph algorithms. Blowfish has been also identified as a powerful cryptographic algorithm since it can satisfy two basic requirements: high immunity to attacks and relative low algorithm complexity. These two characteristics are essential for implementation of robust and fast electronic cryptographic systems.

Although a complex initialisation phase is required before any encryption can take place, the actual encryption of data is very efficient on hardware coprocessor devices.

All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

The algorithm consists of two parts: a key expansion part and a data encryption part. Key expansion converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes. It is only suitable for applications where the key does not often change.

3.1 Subkeys

Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption.

1. The P-array consists of 18 32-bit subkeys: P1, P2, ..., P18.
2. There are four 32-bit S-boxes with 256 entries each:
   S1,0, S1,1, S1,2, ..., S1,255;
   S2,0, S2,1, S2,2, ..., S2,255;
   S3,0, S3,1, S3,2, ..., S3,255;
   S4,0, S4,1, S4,2, ..., S4,255;

The exact method used to calculate these subkeys can be found in [6].

3.2 Encryption

The general algorithm is based in a Feistel network consisting of 16 rounds, see Figure 1 and the following pseudo code. The input is a 64-bit data element, X.

Divide X into two 32-bit halves: XL, XR
For i= 1 to 16
XL = XR XOR Pi
XR = F(XL) XOR XR
Swap XL and XR
End for
XR = XR XOR P17
XL = XL XOR P18
Recombine XL and XR

Function F divides XL into four eight-bit quarters: a, b, c, d.

$F(XL) = (S1,a + S2,b \bmod 2^{32}) \text{ XOR } S3,c) + S4,d \bmod 2^{32}$.

3.3 Decryption

Decryption is exactly the same as encryption, except that P1, P2, ..., P18 are used in the reverse order.

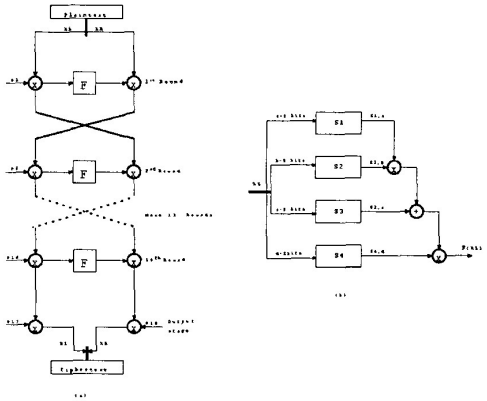

Figure 2. The structure of the Blowfish algorithm.

4. Implementation.

In this section implementation facts and details are described.

4.1 System description used for blowfish implementation

For the blowfish implementation a standard PC with a Pentium 150MHz, 512kByte L2 Cache, 64Mbyte RAM (100 MHz system bus) running Windows NT has been used. The used type of the microEnable was equipped with a Xilinx 4028EX FPGA and 512 Kbytes SRAM. For the blowfish implementation the external RAM was not used.

The access software for the microEnable and the software version (for performance comparison) of the blowfish for execution on the PC has been written in C/C++. The blowfish FPGA designs have been written in VHDL. As design entry tool and for simulation the ALDEC Active VHDL software has been used. For synthesis the Synopsys and Synplicity tools have been utilized. Finally for generating the bit files for the FPGA the Place&Route software from Xilinx Foundation M1.5 has been employed.

4.2 Algorithm adaptation

The previously described algorithm has originally been written in VHDL for an ASIC design. For implementation on the microEnable system the Blowfish design has been adapted. Both the interface has been adjusted to the VHDL interface given by the VHDL library and parts of the algorithm had to have been replaced. Due to existence of the algorithm described in VHDL, the use of the CHDL environment was not reasonable.

Three different interfaces were designed, in order to adapt the Blowfish ASIC description into the microEnable PCI bus. Two of those interface used DMA on Demand. This feature of the microEnable system improves the overall performance very much; because all data are transfers from the media to the memory are executed in only one clock cycle.

The third interface was designed to load the keys. It is very simple interface, mainly because the key length is the same as the PCI data. Therefore this kind of transfer is done straight with one data key per clock cycle.

The VHDL code was synthesized in the microEnable using a Xilinx 4028EX FPGA. This implementation resulted in a throughput of 38MB/s, approximately.

4.3 Performance results

We compared our FPGA implementation of the Blowfish cryptographic algorithm with two different implementations; all designs use 64 bits data. The first comparative implementation was software only [6], executing on a Pentium 150MHz at a rate of 8.3MB/s. The second one is an ASIC implementation [4] with a clock of 66MHz. This device with 4,620 standard cells (43,280 transistors), four 256 words × 32-bit RAM, and two 16 words × 32 bit RAM, has a throughput of 266MB/s.

Table I show the quantitative results.

	ASIC	FPGA M.Enable	PC Pentium
Clock	66 MHz	10 MHz	150 MHz
Throughput	266 MB/s	38 MB/s	8.3 MB/s

Table I - Achieved results.

5. Conclusions and Outlook.

Our implementation of the Blowfish algorithm on the FPGA coprocessor microEnable demonstrated that it is possible to have a flexible, cheap and high performance implementation of a cryptographic algorithm on standard PCs.

Compared to an ASIC implementation the performance on the microEnable system is worse. The flexible usage of system - switching application designs in a few 10 milliseconds - compensates this disadvantage in many cases.

The microEnable system is a very simple and cheap way to adapt a VHDL ASIC design to a PC platform. On the other hand efficient prototyping of ASIC designs is possible. Basic elements are high density FPGAs and a comfortable and easy to use development environment.

Furthermore this result shows that the Blowfish cryptographic algorithm implemented in the microEnable system could be used in future secure, high-performance networks based in PC platform.

References

[1] O. Brosch et al.: Simulating FPGA Coprocessors using the FPGA Development System CHDL, Proc. PACT '98 Workshop on Reconfigurable Computing, Paris (1998) 78-82

[2] Silicon Software Co.: http://www.silicon-software.com

[3] H. Feistel. Cryptographic and Computer Privacy. Scientific American, pages 15-23, Vl 228, N.5, May 1973.

[4] A. C. C. Vieira et ali. SCOB, a Soft-Core for the Blowfish Cryptographic Algorithm. SBCCI99, pp220-223, Oct 99

[5]. B. Schneier. Applied Cryptography. John Wiley and Sons, New York, NY, 1996.

[6] B. Schneier. Description of a New Variable-Length Key, 64 bit Block Cipher (Blowfish). Cambridge Security Workshop Proceedings, Spring-Verlag, pages 191-204, Dec. 1993.

[7] Synopsys, Inc. Behavioral Compiler User Guide Online Documentation, v1998.02, 1998.

[8] Synopsys, Inc. Synopsys Online Documentation, v1998.02, 1998.