jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 1, PP. 21–26
Manuscript received December 31, 2011; revised March 2012. DOI: 10.2478/v10177-012-0003-9

Whirlpool SoPC Implementation – Hardware/Software Co-Design Example

Kamil Krawczyk, Paweł Tomaszewicz, and Mariusz Rawski

***Abstract*—The aim of this work was to design a System on Programmable Chip (SoPC), that implements the Whirlpool Hash Function (WHF) algorithm. An assumption of the project was to use an embedded soft-processor NIOS II controlling the whole system, which functionality was extended by a custom logic in order to improve the used algorithm efficiency. This paper presents the Whirlpool Hash Function realized in several SoPC configurations, which differ in implementation complexity and performance.**

***Keywords*—FPGA, SoPC, soft-processor, NIOS, custom instruction, custom component, hashing function.**

I. Introduction

HASHING functions are specific type of one-way cryptographic functions, that are able to process finite length data and produce small fixed size output called hash. Because for a specific data the produced hash is unique, the hashing functions are widely used for the purpose of data integrity verification and message authentication schemes. Like most of the cryptographic functions, the hashing functions perform heavy and complex computations on large amounts of data [1]. For that reason the throughput of cryptographic functions is an important factor and often such functions must be realized in dedicated embedded systems. Such type of dedicated system was used for the purpose of this paper in order to implement one of the hashing functions – the Whirlpool hashing function.

Embedded systems are the specialized processor systems operating under the real-time conditions. This kind of designs are realized as a hardware/software systems, where software is responsible for functionality and allows easy modification, while hardware solutions are used to increase the performance of complex or heavy computational subroutines [2]. Nowadays microelectronics' technology, especially new solutions in FPGAs (Field Programmable Gate Arrays), allows to implement microprocessor and subsystems on one integrated circuit [3]. This type of solution is called System on Programmable Chip (SoPC). SoPC is a great tool allowing to conduct designing in an easy and efficient way, even of hardware/software co-synthesis systems.

This paper presents the SoPC implementation of the Whirlpool hash function realized in a few system configurations, which differ in software/hardware co-synthesis proportions. Each following realization offloads software tasks to hardware, improving whole system performance. Results of these realizations are compared with prepared software implementations of the Whirlpool function performed on system based on general purpose processor.

This work was partly supported by the Ministry of Science and Higher Education of Poland – research grant no. N N516 418538 for 2010-2012.

K. Krawczyk is with the Department of Electronics and Information Technology Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland (e-mail: K.Krawczyk@stud.elka.pw.edu.pl).

P. Tomaszewicz and M. Rawski are with the Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland (e-mails: {ptomasze,rawski}@tele.pw.edu.pl).

II. The Whirlpool Function

The Whirlpool is a one way hash function, which algorithm was based on a substantially modified Advanced Encryption Standard (AES). This function takes a message up to 2^{256} bits (10^{64} terabytes, in length) and returns a 512-bit hash message [4].

In the Whirlpool function algorithm there can be indicated tree main stages of message processing (Fig. 1): *padder* stage, *W cipher round* stage, and *message compression scheme* stage [1]. In the first stage, based on Merkle-Damgård strengthening, to a message '1' bit followed by a variable number of '0' bit is added, is such a way that message can be divided into equal 512 bits blocks, where in last 256 bits of last block, the length of original message is stored. During ten rounds of the *W cipher round* stage each padded block is transformed into 8×8 byte arrays and undergo specific bit operations in four internal stages: (γ) the non-linear, (π) the cyclic permutation, (θ) the linear diffusion and (σ) key addition stage. During the non-linear γ stage the individual bytes are remapped to a different value. The cyclic permutation π stage performs a cyclic rotation of columns of the matrix, where each column is moved downwards by the index value of shifting column. The linear diffusion θ stage is the operation of multiplication the input data matrix by a matrix generator. The key addition in Whirlpool function simply means performing of 512-bit XOR operation between hash state and specially generated key [1], [4].

After all ten rounds of the *W block* stage in the last step a compression is performed, where according to the Miyaguchi-Preneel hashing scheme a following next blocks of data are given under XOR operation with their hashed form and previously obtained hash state, so as the result the 512 bit hash is obtained [1]. Exact Whirlpool's function algorithm and mathematical model is widely described in [4].

III. System on Programmable Chip

A Field-Programmable Gate Array (FPGA) is a type of logic chip that can be programmed with use of specific Hardware Description Languages (HDL). The HDL allows to define FPGA-based system components in ANSI C like source codes, which can be synthesized into the hardware logic.

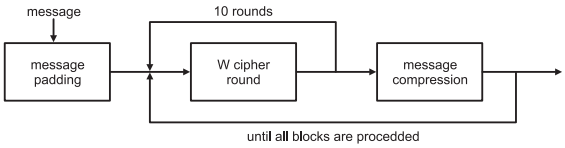

Fig. 1. High level block diagram of Whirlpool.

The FPGA circuits possess the same advantage of possibility to implement specialized systems like in ASIC (Application Specific Integrated Circuit) technology. However, due to fully programmable structure at the same time the FPGAs allow reducing costs of designed realizations and redesign systems even after production process [5].

One of the greatest advantages of using the FPGA structures is that, they allow to use IP Cores (Intellectual Property Cores), which are the defined in HDL standardized system components, that can be used among different FPGA-based devices. One of the most advanced IPs that can be used in the modern FPGAs are the soft-processor cores – the models of a specific general purpose processors (CPUs) allowing the FPGA structures to perform C/C++ applications [5]. There is also possibility to add IP cores defined by user with non-specified architecture restrictions, that can handle any desired tasks.With use of special computer-aided design tools (CAD) IP Cores can be used to design digital circuit in a FPGA chip, what results in creation of the dedicated fully functional system (System on Programmable Chip) embedded in programmable chip [6]–[8].

For the purpose of described in this paper the Whirlpool function embedded SoPC realizations, the Altera's FPGA-based development board DE2-70 was used [9]. With use of the Altera SoPC Builder and Quartus II CAD tools, on single FPGA structure there were embedded some components creating together a fully functional embedded computer system [7], [8]. The main elements used in designed system were the NIOS II 32 bit soft-processor, 32MB SDRAM controller and communication serial interface protocol JTAG-UART [10]. The DE2-70 board is equipped in the 50 MHz system clock. However, with use of the phase locked loop (PLL) available on the Altera Cyclone II FPGA chip it was possible to increase the system clock frequency to the 100 MHz, which supplied all of the designed SoPC system components. Optionally in couple implementations there were added some user defined components, which were able to perform partially or in whole the Whirlpool algorithm in the dedicated hardware.

IV. Software Realization (PC & SoPC)

The Whirlpool function like most of a cryptographic functions, was designed for hardware implementation [4]. This algorithm is mainly based on substitution and permutation operations, which are the bit operations, so the effective realization of this algorithm is not easy to be implemented on systems using just general purpose processors [4], [11]. However two fully software realizations of the Whirlpool function were prepared and performed on PC's general purpose processor (Intel Core 2 Duo CPU L7500 2×1.6 GHz) and on SoPC system equipped with the NIOS processor. First

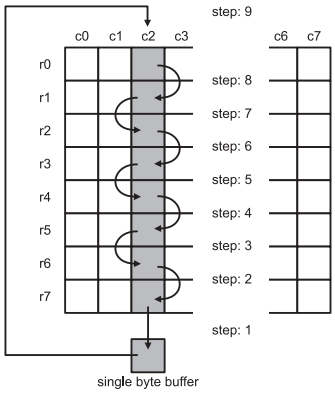

Fig. 2. Scheme of bytes shifting during the cyclic permutation stage.

realization was performed with most optimized C program version of the Whirlpool function, which code was found on the Whirlpool function creators page [4]. Software optimization of this Whirlpool function implementation was based on look-up tables containing results of conversions of some portions of algorithm, which gives better efficiency in exchange for greater memory usage. Second software realization was performed with C program developed on the basis of the Whirlpool hash function hardware compression scheme, that can be found in [1], [4]. In this software implementation all cryptographic calculations were performed according to the original Whirlpool function algorithm operations translated to the ANCI C sequential codes. Such prepared program consists of a large number of nested loops, that are not efficiently performed in GPU. For example the cyclic permutation stage is based on shifting bytes in loop separately in each column (Fig. 2).

V. Hardware Accelerated Whirlpool Realizations

In previous section it was mentioned that user defined logic can realize any desired tasks. According to that assumption the user can design a logic, which aim is to cooperate with the soft-processor in order to increase its efficiency. Such logic is called *a hardware accelerator*, or more precisely: *the hardware accelerator supporting software solution* [6]. The Whirlpool hash function SoPC implementations, described in this section, were realized with use of such hardware accelerators supporting the NIOS processor in computations.

There are two main types of hardware accelerators that can be used in the SoPC systems equipped with the NIOS processor. First type is called a custom instruction – user logic that is built within the NIOS processor logic in order to extend its instructions set (Fig. 3). This type of hardware accelerator connects to the NIOS' Arithmetic Logic Unit (ALU), what limits a functionality of the custom instructions [6]. These accelerators do not have direct access to the other SoPC system components and they can possess only two 32-bit inputs and single 32-bit output. On the other hand, such implementation of the custom instructions brings an advantage in the way of using these accelerators [6]. This type of user logic can be easily embedded in the NIOS processor due to the SoPC

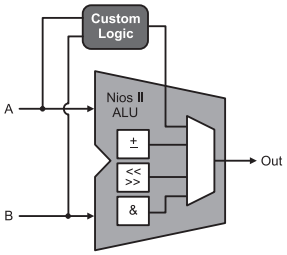

Fig. 3. Custom instruction.

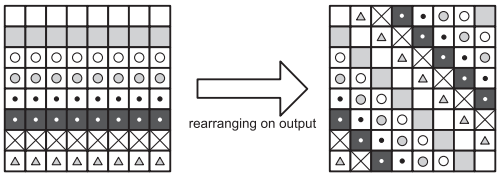

Fig. 4. Single cycle cyclic permutation function operation realized in custom instruction.

Builder tool, which allows for generation of a simple macro functions, through which the software can call user defined hardware instructions [12].

Custom components are the second and much more advanced type of hardware accelerators, which can be defined as a user defined SoPC system standalone peripherals. Custom components can possess any architecture and functionality limited only by the size of a FPGA structure. One of the greatest advantages of this type of hardware accelerator is that, like any other peripherals they can work in parallel to other SoPC system components including the CPU and other user defined logic. Moreover, custom component can be designed in such a way, that it can perform multiple parallel operations inside of it and with any number of other SoPC system components. However, such hardware accelerator requires additional complex arbitration logic, that allows for communication between designed component and other system peripherals [6]. Moreover, in order to control such user logic from the software level some specific control program (driver) must be written. Development of such drivers is simplified due to Hardware Abstraction Layer (HAL) routines [13].

Designing FPGA-based SoPC system with user defined custom peripherals requires knowledge about Avalon interconnection and interface system [14]. This interconnection system, called also an Avalon bus, specifies easy-to-understand port connections and protocols used between master and slave components. Master components are components with master ports, which initiate bus transfers, while slave components are components with slave ports, which only accept transfers initiated by the masters [5], [14]. It is possible that components possess both master and slave ports. The Avalon bus supports multiple masters and slaves, which interact each other with supervision of arbitration logic.

A. WHF Realization Implementing the Custom Instructions

The above described custom instructions and custom components were used in two different hardware accelerated WHF realizations. For the purpose of these realizations the software implementation of the original Whirlpool hash function algorithm was analyzed in order to find the bottlenecks of this algorithm. It was discovered that parts of software, where bottlenecks were found were located in the *W cipher block* – mainly every round stage where bit operations were performed there was a bottleneck.

For the purpose of the first realization implementing the hardware accelerators, all the *W block round* software functions ((γ) the non-linear, (π) the cyclic permutation, (θ) the linear diffusion and (σ) key addition stage) were replaced by custom instructions embedded in the NIOS processor . Due to such design all functions, where bit operations were performed, could be efficiently realized in hardware. For example, the cyclic permutation (π) function, that in software was realized with use of large amount of nested loops, could be implemented in hardware in such a way, that proper custom instruction just after obtaining some block of data was able to return this data blocks with properly rearranged bytes (Fig. 4).

Moreover, each of the W block functions was easy to implement in custom instructions. The non-linear stage γ could be implemented as a simple logic using very small substitution tables, which outputs were mixed with proper processed byte's bits according to the algorithm. The linear diffusion stage θ was realized as a simple shift and add logic mixing all bits in the processed block of data. Finally the key addition stage σ was realized as a 512 bit adder equipped with a small register file, that stores the Whirlpool rounds key constants. Beside *W cipher block* functions rest of the Whirlpool function algorithm was implemented in software, which due to special automatically generated opcode could use designed custom instructions like a typical software functions [1], [4]. Each custom instruction could be called by using following macro code template:

```
result = CUSTOM_INSTRUCTION_ID(selectorN,
  dataA, dataB);
```

Parameters $dataA$ and $dataB$ are the two 32-bit input values. Parameter $selectorN$ is a 8-bit signal value that can determine different custom instruction internal operations.

B. WHF Realization Implementing the Custom Component Accelerator

In the second designed hardware accelerated realization, a created custom component was used, that implemented the whole Whirlpool hash function algorithm in a dedicated hardware, called *WHF accelerator*. In this realization the software role was limited only to initiation of the WHF accelerator and indication data which should be processed. The WHF accelerator was designed in such a way, that after initiation it was able to read indicated data directly from the system memory. When hash was ready the WHF accelerator was able to interrupt the processor in order to signal the end of data processing and in follow the NIOS CPU was able to download the hash result directly from the WHF component's internal registers.

C. WHF Accelerator – Architecture and Operation

Because the WHF accelerator peripheral was designed as a dedicated non-standard logic, some explanations to its architecture and operations are needed. The WHF accelerator is a set of co-operating, working in parallel modules visible for the SoPC system as a single component (Fig. 5). There can be distinguished four sub-modules: component's control, master read, padder and W block module. Additionally there can be distinguished modules used as a functional blocks processing the data according to the Whirlpool function algorithm (padder and W block modules) and modules, which implement the arbitration logic (control and master read modules). The control module mediates between the WHF accelerator and the NIOS soft-processor. This module is responsible for accelerator initiation and for informing the NIOS about component's status. Also the control module is equipped with the register files storing the hash result. The read master module is responsible for transactions controlled by the WHF accelerator, which results in reading directly from the SDRAM memory to the accelerator the required data to be processed by the Whirlpool function. The padder and W block modules contain logic responsible for data processing according to the algorithm. It is important to indicate that the master read module is equipped with the Altera FIFO (SCFIFO) buffer used for the purpose of storing read, but not processed by the padder module data [15]. Due to use of such buffer the data acquisition process could be smoothly performed.

The whole designed WHF custom component works in such a way, that when NIOS initiates the WHF accelerator it passes to the accelerator the base address and length of a desired data. Based on this information the WHF accelerator proceeds with 16-bit reads from SDRAM to the WHF component's internal FIFO buffer. The FIFO buffer holds data for padder, which organizes them into 64 byte arrays. In follow prepared block of data are passed to the W block module, which simultaneously operates on all of the 512 bit data processing them ten times in all round stages ((γ) the non-linear, (π) the cyclic permutation, (θ) the linear diffusion and (σ) key addition stage) (Fig. 6) [1]. All round stages were using the logic similar to that used in the WHF realization using the custom instructions. When hash is ready, the control module sends $done$ signal interrupting NIOS, which can read result directly from internal registers of WHF accelerator. Like it was mentioned, all cooperating modules work together in parallel. So in the same time the control modules supervise the WHF accelerator, the master read performs memory reads and the padder and W block modules perform the data processing.

The WHF accelerator operates in parallel to the other SoPC system components including the NIOS processor, so the software can perform other tasks while waiting for result of hardware accelerator. For the same reason the control of the user defined component from the software level requires use of some specific methods. Because the CPU performs operations in sequence, the software must contain proper loops in order to be able to check component's status from time to time. Like in the case of custom instructions, the communication between the NIOS processor and the Whirlpool Accelerator periphery

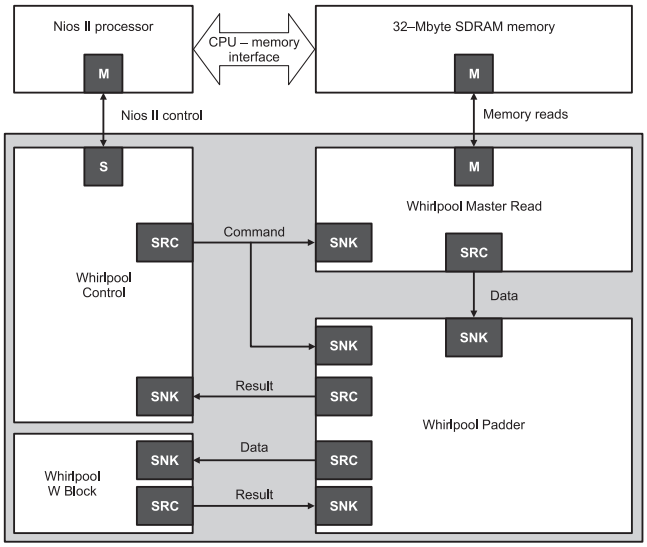

Fig. 5. Whirlpool function hardware accelerator.

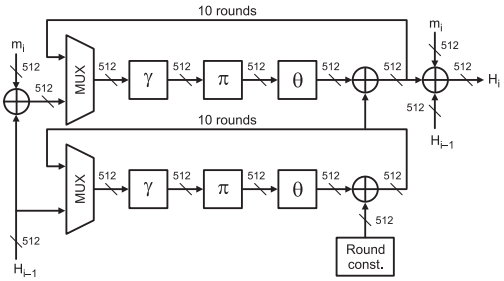

Fig. 6. Diagram of the implementation of the Whirlpool W-Block module.

depends on the HAL input/output macros [13]. The software that controls designed WHF accelerator was involving only two types of mentioned HAL macro commands:

```
IOWR_32DIRECT(COMPONENT_BASE_ADDRESS,
 OFFSET, DATA);
IORD_32DIRECT(COMPONENT_BASE_ADDRESS,
 OFFSET, DATA);
```

The command $IORW\_32DIRECT$ means 32-bit write transaction from the NIOS processor to the component and $IORD\_32DIRECT$ means 32-bit read transaction from the component to the NIOS processor. Parameter $COMPONENT\_BASE\_ADDRESS$ indicates to which SoPC component the read/write operation is performed. The $OFFSET$ can be considered as a identifier of one of the operations, that periphery can perform. $DATA$ parameter is used to pass 32-bit value of data or memory address. Created WHF software depending on $OFFSET$ value was controlling the custom periphery, which after decoding passed instruction was able to return its status, was initiated to read indicated data, or was requested to return a ready hash result.

D. WHF Accelerators – Important Observations

Both hardware accelerated realizations have been burdened with some drawbacks. In the realization using the custom instructions there was a problem caused by port architecture,

which as standard is limited only to two 32 bit inputs and one 32 bit output. In the case of hash function where big block of data must be passed to custom logic, it was necessary to write a ANSI C control program, which constantly writes and reads small portions of data to/from custom instruction . To pass and read single 64-byte block of data to the custom instructions, there have to be done 16 read/write function calls. Each such read and write supervised by processor wastes time and does not allow for great increase of performance. Also due to the same reason, the custom instructions had to be extended with functions, that composed portions of data into 64-byte blocks and decompose them into the 32-bit words.

In the case of design involving the custom component as a hardware accelerator, the main problem was a complexity of the arbitration logic that was responsible for synchronization of all transactions between accelerator and SoPC system. Also the created C program had to be carefully considered in order to optimize proper communication between NIOS processor and WHF accelerator.

VI. Results

In Table I the results of different Whirlpool hash function realizations, described in previous sections, are presented. Column *Realization ID* indicates the realization type:

- *PC_opt/SoPC_opt* – PC/SoPC optimized software realization
- *PC/SoPC_soft* – PC/SoPC not optimized software realization (based on original WHF algorithm)
- *SoPC_CI* – SoPC WHF accelerated by Custom Instructions
- *SoPC_WHF* – SoPC WHF accelerated by Custom Component (*WHF Accelerator*)

In the *C code* column there are values of C code size after compilation of software part realized in the embedded systems. The system resources usage is presented in the *FPGA Resources* column, and achieved realization efficiency expressed as transfer speed was presented in the last column. Each realization was tested on 64 kB data iterated 1000 times.

For the purpose of the *PC_opt* and *SoPC_opt* realizations, the most optimized ANSI C software implementation of the Whirlpool algorithm was used. Thus, obtained results of these realizations show the maximum performance, that can be achieved by the software implemented Whirlpool function executed in given system, where processor was not supported by any functionality extension method. The PC realizations were performed in order to obtain the results of the implementation of the Whirlpool algorithm performed on the general purpose CPU, which possess rich architecture and very fast clock. Thanks to these realizations, it was possible to check, if the NIOS processor-based systems, implemented in FPGAs supplied by lower clock frequencies, will ever achieve the higher efficiency of performing given task, than the high power general purpose processor. The *SoPC_CI* realization implementing the custom instructions and *SoPC_WHF* realization created with use of custom component, were developed in order to improve the performance achieved by the software implemented Whirlpool function performed in the *SoPC_soft* realization. Process of creating of each realization was characterized by varying levels of difficulties in their implementation. Moreover, the way of implementation of each method of increasing the functionality of the NIOS processor, affected the results achieved by each realization.

Analysis of the results achieved by testing the SoPC realizations allows to notice, how with the increasing degree of the hardware acceleration, the efficiency of whole implementation improves. The pure SoPC software realizations are characterized by low efficiency simce it performs complex calculations on large blocks of data in a CPU supplied by very low clock. With additional dedicated logic improving the efficiency of the complex calculations realized in the hardware, the size of C program decreases, but the FPGA resources usage increases, leaving much less space where additional functionality can be implemented. The SoPC realization using the WHF accelerator delivers much better results than the optimized software WHF realization performed on PC equipped with advanced CPU. However, the *SoPC_WHF* realization uses a lot of the system resources despite the fact that its functionality is limited only to perform the WHF calculations.

In the *SoPC_CI* realization each W block was implemented in the separate custom instruction. The software, during each of the W block rounds, performed large amount of hardware function calls, what took much of the execution time. For that reason, despite of offloading to the hardware the most bottleneck parts of the Whirlpool algorithm, the *SoPC_CI* realization does not bring any spectacular results.

VII. Summary

Modern FPGA technology, the Hardware Description Languages and advancement CAD tools allow to design in easy way fully functional embedded systems performing the desired tasks in most efficient way. In this article, based on the Whirlpool hash function algorithm, the methods of hardware acceleration of the software solutions in the embedded SoPC systems were presented, along with discussion of their advantages and disadvantages. Moreover, it was shown how efficiently the complex and heavy computational cryptographic function can be implemented in the FPGA-based embedded system. All realizations presented in this article were different in implementation complexity and proportion of the software to hardware functionality, which allowed thoroughly to examine both the process of creating embedded system as well as the impact of various configuration changes on the obtained results.

Very important conclusions resulting from this work are related to the methods of hardware acceleration of the software solutions. The hardware method of expanding the functional capabilities of the NIOS soft-processor allow to obtain a huge increase in efficiency in implementations supported by them. The resulting increase in performance compensates the low frequency of clocks used in the FPGA devices. It was shown that, depending on type of the hardware accelerator used for the software solutions, and depending on the volume of the functionality of the software-implemented operations elevated to hardware, different performance growth can be achieved, that is combined with different level of consumption

TABLE I
RESULTS OF REALIZATIONS OF SYSTEMS PERFORMING THE WHIRLPOOL FUNCTION

Realization ID	System Clock [MHz]	Code [kB]	FPGA Resources #LC	Transfer [Mb/s]
PC_opt	1600	–	–	102
PC	1600	–	–	0.96
SoPC_opt	100	43	4478 (7%)	0.31
SoPC_soft	100	28	4478 (7%)	0.03
SoPC_CI	100	16	6985 (10%)	2.48
SoPC_WHF	100	11	17597 (26%)	165

of the FPGA resources. With more and more functionality of the system moved to hardware , an increase in resource consumption can be observed, as well as significant increase of the realization efficiency. For this reason in the limited resources realizations, the use of the NIOS processor extension methods is associated with maintaining an appropriate proportion between resources usage and performance increase.

REFERENCES

[1] P. Zalewski, "FPGA design and performance analysis of SHA-512, Whirlpool and PHASH hashing functions," Master's thesis, Rochester Institute of Technology, 2008.

[2] S. Heath, *Embedded Systems Design, Second Edition*, 2nd ed. Newnes, 2002.

[3] J. O. Hamblen and T. S. Hall, "Using system-on-a-programmable chip technology to design embedded systems," *International Journal of Computers and Their Applications*, vol. 13, no. 3, pp. 142–152, 2006.

[4] P. Barreto and V. Rijmen, "The WHIRLPOOL Hashing Function," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2003. [Online]. Available: http://www.larc.usp.br/ pbarreto/WhirlpoolPage.html

[5] F. Plavec, "Soft-core Processor Design," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, 2004.

[6] Altera Corporation. (2008, Jun.) Hardware Acceleration and Coprocessing. edh_ed51006.pdf. [Online]. Available: http://www.altera.com/literature/hb/nios2/

[7] ——. (2010, Dec.) SOPC Builder User Guide. ug_sopc_builder.pdf. [Online]. Available: http://www.altera.com/literature/ug/

[8] ——. (2010, Dec.) Embedded Design Handbook. edh_ed_handbook.pdf. [Online]. Available: http://www.altera.com/literature/hb/nios2/

[9] Terasic Corporation. (2010) DE2-70 User manual. DE2_70_User_manual_v109.pdf. [Online]. Available: http://www.terasic.com.tw

[10] Altera Corporation. (2010, Dec.) Nios II Processor Reference Handbook. n2cpu_nii5v1.pdf. [Online]. Available: http://www.altera.com/literature/hb/nios2/

[11] P. Kitsos and O. Koufopavlou, "Whirlpool hash function: architecture and vlsi implementation," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 893–6, 2004.

[12] Altera Corporation. (2011, Jan.) Nios II Custom Instruction User Guide. ug_nios2_custom_instruction.pdf. [Online]. Available: http://www.altera.com/literature/ug/

[13] ——. (2011, Jul.) Guidelines for Developing a Nios II HAL Device Driver. an459.pdf. [Online]. Available: http://www.altera.com/literature/an/

[14] ——. (2010, Aug.) Avalon Interface Specifications. mnl_avalon_spec.pdf. [Online]. Available: http://www.altera.com/literature/manual/

[15] ——. (2010, Sep.) SCFIFO and DCFIFO Megafunctions. ug_fifo.pdf. [Online]. Available: http://www.altera.com/literature/ug/