Scientific Research Publishing

Journal of Computer and Communications, 2017, 5, 40-59
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Programmable SoC for an XTEA Encryption Algorithm Using a Co-Design Environment Replication Performance Approach

Mohamed H. Al Meer

Computer Science & Engineering Department, College of Engineering, Qatar University, Doha, Qatar
Email: almeer@qu.edu.qa

How to cite this paper: Al Meer, M.H. (2017) Programmable SoC for an XTEA Encryption Algorithm Using a Co-Design Environment Replication Performance Approach. *Journal of Computer and Communications*, **5**, 40-59.
https://doi.org/10.4236/jcc.2017.511004

Received: August 21, 2017
Accepted: September 17, 2017
Published: September 20, 2017

Abstract

With the rapid development of wired and wireless networks, the security needs within network systems are becoming increasingly intensive owing to the continuous development of new applications. Existing cryptography algorithms differ from each other in many ways including their security complexity, size of the key and words operated on, and processing time. Nevertheless, the main factors that prioritize an encryption algorithm over others are its ability to secure and protect data against attacks and its speed and efficiency. In this study, a reconfigurable Co-Design multi-purpose security design with very low complexity, weight, and cost, has been developed using Extended Tiny Encryption Algorithm (XTEA) data encryption standards. The paper aims to discuss issues and present solutions associated with this system, as well as compare the Co-Design implementation approach with Full-Hardware and Full-Software solutions. The main contribution that this paper offers is the profiling of XTEA cryptographic algorithm to reach more satisfactory understanding of its computation structure that leads to fully software, fully hardware, beside the co-design implementations all together, of this light weight encryption algorithm.

Keywords

FPGA, Nios II, Co_Design, PSoC, XTEA, TEA

1. Introduction

Data security is of high concern in applications where user data is exchanged, especially regarding data transmission over network channels. Radio Frequency Identifiers (RFIDs), smart meters, smart thermostats, and smart grids are good

examples of such applications. We can see these types of applications used in devices of all types including gadgets, health care devices, and environment and pollution monitoring systems.

Such devices often connect to the Internet or a trusted destination by means of a network, but this exposes them to being hacked, snooped, cloned, counterfeited, or even tracked, which may lead to the violation of user privacy. Most of these devices are small in size, inexpensive, and consume low power, but may not withstand more part area for security concerns. Since there is a growing need to ensure the security of data transmitted through such devices, many lightweight cryptographic algorithms have been developed and implemented.

Lightweight cryptography is comprised of algorithms specialised for implementation in constrained environments, such as communication carried out by RFID tag systems, wireless sensor networks (WSNs), or contactless smart bank cards. With highly limited resources found in such applications, different lightweight cryptographic protocols have emerged, and can be categorised as block encryptions such as Present, Clefia, and Katan, or stream ciphers such as Grain, Bean, and Hummingbird.

Tiny Encryption Algorithms (TEAs) and Extended TEAs (XTEAs) are two lightweight algorithms categorised as 64-bit Feistel Block network cryptographic algorithms that rely upon 32 rounds and use a 128-bit secret key. Various implementations of lightweight cryptography have been mapped to Application-Specific Integrated Circuit (ASIC) and Field-Programmable Gate Array (FPGA) devices, but unfortunately those implementations are often configured for old ASIC or FPGA families. One recently developed XTEA implementation offers acceptable security with efficient computation and use of power resources. It has been successfully implemented by FPGAs with high throughput as reported by many researchers.

In this study, it is thought that replicating an algorithm synthesised in an FPGA, where several computational devices can operate concurrently on different data chunks, will fit closely within the definition of the parallel computing paradigm. Moreover, instead of tailoring the algorithm to cope with architectural pipelining and/or performing extensive architectural optimization to reduce the processing path time, replication could be used if the required processing algorithm is of concurrent nature, *i.e.* if different computations can be carried out independently and simultaneously.

Although studies have been conducted on XTEA implementation, none have addressed replication as a concept for increasing computational efficiency. Furthermore, replication has seldom been used to increase cryptography speed using FPGAs. This thesis intends to determine the significance of XTEA Co-Design implementation as a means of conducting encryption computations. Hence, the replications and Co-Design encryption computations will be addressed in detail in this study to show how both can affect the throughput.

All designs were synthesised and implemented using Altera Quartus 14.1 and simulated using ModelSim PE II. The end designs targeted an Altera Cyclone V

FPGA. The content of this paper is organised as follows: Section 2 introduces the XTEA algorithm, Section 3 describes the tools and technologies used, Section 4 presents the built system architecture, Section 5 discusses the implementation results and performance comparison, and finally Section 6 provides a conclusion about the presented work.

2. XTEA Implementationin Configurable Hardware Logic

FPGA and ASIC implementations for cryptographic algorithms have been investigated by many researchers for several years, usually targeting Xilinx or Altera FPGAs and previous ASIC Hardware Description Language (HDL) programmable devices. In [1], the latest Xilinx Virtex 5 FPGA device was incorporated beside the Modest Altera Cyclone III series of FPGA devices to implement a Lightweight Encryption Algorithm (LEA). Similarly, in [2] two LEAs, namely Present and Hight, were analysed and implemented using a Xilinx Spartan 3 FPGA device. Yet another study [3] successfully proposed the implementation of Hummingbird, an LEA, after conducting extensive preliminary studies using the Altera Cyclone II FPGA device.

The first investigations of TEA and XTEA LEAs were [4] and [5]. Following these studies, a significant number of research investigations appeared emphasizing different techniques for hardware implementation. Some studies focused on software rather than hardware implementations of TEA and XTEA algorithms, such as [6]-[12]. A few major efforts have studied both hardware and software aspects of TEA for cost effective use of RFID applications [13] [14] [15] [16].

In [6] the author studied XTEA implementation on both the ASIC and FPGA programmable logic platforms. The FPGA implementation used Xilinx ISE 9.1 tools with Xilinx Virtex 5 and Spartan 3 FPGAs, containing the XC3S50-5 and XC3S200-5 FPGA devices, respectively, and reached a real-time encryption throughput rate of 36 Mb/s.

In various case studies [7] [8] [9], researchers identified three different VHDL architectural modification models of the main TEA algorithm, specifically, sequential (looping), parallel, and mixed implementations, but used a specific LeonardoSpectrum 0.35 um CMOS type ASIC to perform the implementations. Other XTEA studies showed that a throughput of 53 Mb/s could be achieved, and that ModelSim could be used for simulation in conjunction with the Xilinx ISE 10.1 development tool for synthesis [13]. Maximum operational frequencies of 129 MHz and 71 MHz were reported when tests were performed using the Virtex 4 and Spartan 3 FPGA devices, respectively.

Furthermore, XTEA investigations have been reported [10] [11] [12] on applications that employ RFID communication security protocols, but using different FPGA platforms compared with the previous study. For example, in [12] XTEA was recommended for RFID wireless authentication security protocols, and was actually implemented using an Altera DE2 board.

In another study, the development and validation of an RFID reader and tag modules incorporating the System On Programmable Chip (SOPC) tool with 32-bit RISC Nios II processors, was reported. This was an example of a software implementation carried out using a soft processor running code with a system response of 1.06 ms. Still another FPGA implementation can be found in [15], where an Altera-DE0 platform embedded Altera Cyclone IV FPGA device was used to implement the XTEA encryption. When the researchers compared the FPGA results to CPU tests, a 21× speedup was found.

The previously discussed studies show that XTEA can effectively be used as an encryption engine for RFID secured communication protocols.

Furthermore, ambitious experiments have been performed on an XTEA encryption algorithm using General Purpose Graphical Processors (GPUs) [16]. In this study, three computing platforms were cooperatively tested, namely a GPU, an FPGA, and a CPU. Although the FPGA outperformed the CPU, the GPU performance recorded the fastest throughput, reaching 5.3 Gb/s. The FPGA board used in this work was the Xilinx Zynq-7000 SOC ZC702 evaluation board.

3. Methods and Tools

3.1. XTEA Encryption Algorithm

The first TEA was developed by Wheeler and Needham [4] [5], who reported that with very simple operations, TEAs could contribute to the total confusion, such as XORs, logic shifts, and modulo 32-bit addition operations working on double 32-bit inputs. Table 1 illustrates pseudo code for both the encryption and decryption mentioned in (Wheeler and Needham, 1996).

As the first systematic study, it has been noted that its small code size and low storage requirements qualify it for software encryption operations, which are usually hosted by small embedded systems. Subsequently, the XTEA encryption algorithm was developed from the original TEA by the same scholars as an extension, in which it was reported as a valuable and innovative alternative for increased security when supplemented with key shuffling operations. Although XTEA is considered one of the most important lightweight algorithms, it suffers from low-round security weakness, and should be able to accommodate 32 rounds in order to accommodate high security applications.

In detail, the XTEA implements encryption using a 64-bit block split into two 32-bit halves, v0 and v1, which are input to the algorithmic routine that per-

Table 1. Pseudo code for XTEA encryption and decryption.

Algorithm 1: Pseudo Code XTEA Encryption	Algorithm 2: Pseudo Code XTEA Decryption

Sum = 0; delta = 0x9E3779B9
for i = 0 to N do
v0+ = ((v1 << 4) xor (v1 >> 5) + v1) xor (sum + key[sum & 3])
sum+ = delta
v1+ = ((v0 << 4) xor (v0 >> 5) + v0) (sum + key[sum >> 11 & 3])
end for
	Sum = 0; delta = 0x9E3779B9
	for i = 0 to N do
	v1− = ((v0 << 4) xor (v1 >> 5) + v0) xor (sum + key[sum >> 11 & 3])
	sum+ = delta
	v0− = ((v1 << 4) xor (v1 >> 5) + v1) (sum + Key[sum &3])
	end for

forms 32 rounds (Nr = 32). In XTEA, the key scheduling is modified to reflect different patterns for mixing the data and key continuously each round to add substantial confusion. Only four subkeys, each having a 32-bit length, are used, and basic addition and subtraction operations follow the modulo 232. The logic shifts are a logical left shift by four and a logical right shift by five, in addition to a simple 32-bit XOR logic operation.

The permutation functions are expressed by $f(x) = (x<<4 \oplus x >> 5) + x$, and subkey generation functions are expressed by sum + k (sum $\vee$ 3), and sum + k (sum >> 11 $\vee$ 3). Sum acts as a selector from the four subkeys k0, k1, k2, and k3 dependent on bits 0 and 1 of the sum or bits 11 and 12. The results of the permutation function and generated subkey generated are XORed and ADDed to v0 and v1. It is worth noted that the value of sum is initialized to zero prior to the start of the computation, and the value of delta is fixed to 0x9*E*3779*B*9.

3.2. Field Programmable Gate Arrays (FPGAs)

FPGAs are a recently developed technology used to synthesise any type and number of logic besides arithmetic functions. FPGAs nowadays are used to prototype algorithms and verify the solution before fabricating the final prototype into the ASIC chips. Unlike software languages such as C-C++, python, and others, FPGAs are based on HDL, such as VHDL and Verilog. Such languages have the ability to execute algorithms in parallel as compared to processors when executing instructions but sequential.

FPGAs as a type of reconfigurable hardware, can model huge sizes of mathematical algorithms that usually would be implemented by software, but with a higher density and speed, by using their large complex architectural capacity. Most FPGA devices contain one or more fabricated hard processor core (s) or else one or more soft processor core (s). Although FPGAs do not perform as fast as ASICs, the time and cost for their development are lower than for ASICs implementation, which makes them favourable in the eyes of software-hardware developers.

As the use and progression of FPGA technology has grown dramatically, especially in algorithmic realization, it has become possible for fully embedded systems to be implemented in a single FPGA chip.

3.3. NIOS II and QSys Technology

The Nios II is a synthesizable VHDL model of a 32-bit embedded-processor architecture, specifically intended to work with the Altera family of FPGAs. The processor is highly flexible and can be tailored for any design configuration, making it well-equipped for System-On-a-Chip (SOC) designs. Nios II came after its predecessor, the original Nios (Nios I), with enhancements in its architecture that make it more suitable for a range of cost-sensitive or real-time applications.

The Nios II is a Reduced Instruction Set Computer (RISC) soft-core type ar-

chitecture intended to be implemented entirely with programmable logic (FPGA) and supplemented with memory blocks originally found in Altera types of FPGAs. The Nios II processor with its soft-core architecture facilitates the design and specifications of a customised CPU core best-suited for the particular application requirements. While being designed, it is easy to change the Nios II's basic functions by adding a predefined sort of a Memory Management Unit (MMU or MPU), or by customizing certain instructions and peripherals as well. The Nios-II core is available in three configurations: the Nios II/f (fast), Nios II/s (standard), and Nios II/e (economy). According to Gartner Research[1], NIOS-II is the most widely used soft processor in the FPGA industry.

Nios II hardware designers usually use the Qsys, a system integration tool, which is now a component of the Quartus-II software development package that you can call immediately, for configuring and generating a complete Nios-II system. The Quartus 14.1 software includes Qsys, which also can be claimed as an advanced system integration tool for Nios-II soft processor system design. With Qsys, developers can construct and integrate processors, peripherals, memory controller, communication controllers, and custom intellectual property (IP) cores, using a user-friendly GUI tools. Subsequently, the Quartus-II is directed to perform the synthesis, placement, routing, and generation of the system on the selected FPGA, as well as connect the IP components with a generated system interconnect.

3.4. Wrapping Circuit Design

In accordance with the components offered by the Qsys development toolset, a custom designed I/O peripheral, specifically a hardware accelerator for the XTEA encryption algorithm, is implemented using a Finite State Machine (FSM) expressed in VHDL language. The resulting VHDL descriptive circuit fundamentally contains multiple input and multiple output ports as well as a few controls and status signals. These ports and signals need to be read from or written to using higher, but similar types of VHDL classes that can handle the typical MM Avalon interface signals and ports.

Fortunately, the Nios II processor uses the Avalon interconnect for data transfer and control to interface with any custom-made components as mentioned earlier. In addition, the Nios II system needs to convert the circuit to an IP core (a Qsys component) with adequate Avalon interface signals as well. The wrapping circuit, which needs to be designed and created, is instantiated and added to the top of the FSM circuit in order to make its IO ports compatible with the MM Avalon specifications and complete the job previously mentioned. However, this wrapping operation is usually moulded with circuits containing interfaces, buffers, output decoding circuits, and input multiplexing circuits, to assist in completing its functions.

HDL code used to wrap the XTEA circuit was successfully developed and

[1]http://www.gartner.com/technology/home.jsp

eventually synthesised while inherently instantiating the XTEA encryption engine, containing the logic required to buffer, decode, and multiplex, as shown in Figure 1.

4. Implementation

4.1. System Architecture

The system implementation is introduced on the basis of using diverse computing platforms concept proofing, which falls into three categories: the XTEA hardware accelerator implemented by VHDL first, the Full-Software implementation second, and the Co-Design implementation in conjunction with the Nios II the soft processor of the Altera Qsys EDA software third. Specifically and intentionally, the Full-Software implementation was used on the basis of offering benchmarking needed for a results comparison.

4.2. Full-Software Implementation (Nios II)

The Nios II processor is in fact a highly flexible processing tool suitable for any design configuration, even though it is mainly intended for SOC designs. Additionally, the NIOS II IDE has a GNU compiler with a C/C++ license, used to assist in the programming. The Qsys system designed to carry out the full-software tests consists of the following components as shown in Figure 2.

1) Clk and Reset: feeds the clock output and a clean reset input to all other modules.

2) Nios2 Soft-core Processor: soft processor module with a Nios II soft core processor of type E (Economy). No instruction or data caches are added. No hardware support for division and multiplication (DSP blocks) are added.

3) Memory module: interface comprised of a 12-bit address bus, 1 bit for each of enable, chip select, and write controls, in addition to 32-bit lines for read data and write data buses. Byte enable takes 4-bits to control the exchange of the 4

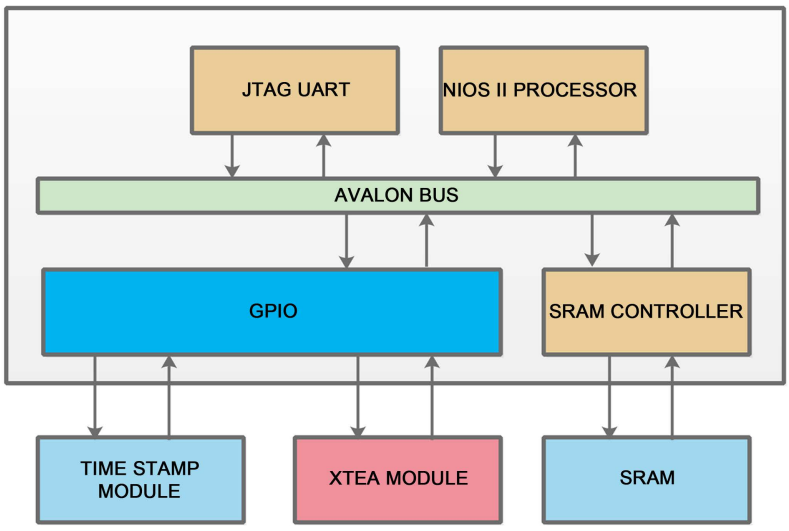

Figure 1. System design using Nios II processor.

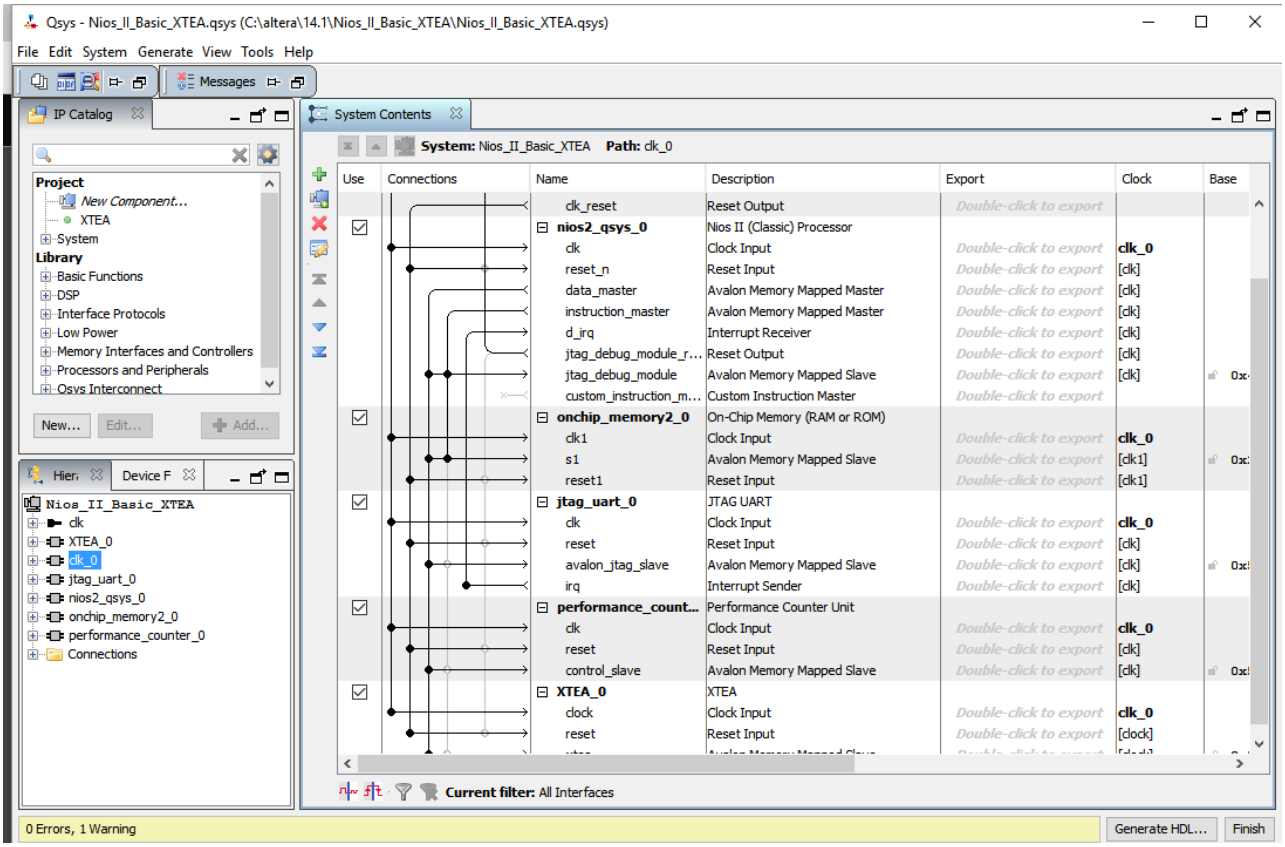

Figure 2. System generated in SOPCQSys builder.

bytes of the 32-bit data bus lines. The Altera Avalon on-chip RAM memory module has a size of 8 KB.

4) XTEA Hardware Block: hardware accelerators designed to increase the performance of the XTEA cryptographic algorithm code running on the NIOS II processor. The hardware accelerators are implemented as custom components for the Nios II processor.

5) JTAG UART: type of Altera IP that provides a means to communicate with a host PC using serial character streams between the host and the Qsys system. It is basically used for debugging purposes once needed in the Qsys system.

6) Sys ID Peripheral: Altera-based peripheral which uniquely assigns the Qsys system an ID with timestamps. The NIOS II IDE verifies the system ID before downloading new software to the system. This was introduced to ensure that the software runs on a Qsys System for which it is written and compiled.

7) Performance Counters: block of counters that can measure the execution time of selected code (cryptographic routine) by registering all times and occurrences of that section of code. This helps measure the performance of the XTEA system.

The QSys Builder automatically generates the interconnect logic to integrate the components in the hardware system. Figure 2 shows the selection of components required and the system generation for a Full-Software implementation via the Nios II EDS.

First, the integration of the QSys Builder with the Quartus software takes place. Second, the pin assignment is implemented by importing the pin assignment of the Cyclone V "5CSXFC6D6F31C6N" FPGA. Third, the system is generated within the hardware, where the Cyclone II FPGA is connected to the host computer via USB-Blaster cable.

The "C" code accurately realizes the generic algorithm for the XTEA encryption and decryption taken from the source [5], by importing the code with slight modifications to handle the reading and writing to data arrays. Finally, the code is compiled with default optimization and used for Full-Software tests.

After generating the system using Quartus II software, the produced output file is next opened by the Eclipse IDE where a C/C++ Developing GNU compiler assists in compilation and generation of executable code. A new project is created within the NIOS II IDE, and is added to the 'C' code from the XTEA algorithm in a ".C" file format. The project is then built using the "Build Project" command. After the project is built, the code is implemented on the Cyclone V "5CSXFC6D6F31C6N" FPGA using the command "Run as NIOS II Hardware".

Finally, the results for the encryption using the NIOS II IDE with a 128-bit key and 64-bit plaintext/ciphertext as the input parameters are obtained, with the output easily displayed in the NIOS II Console Window. As an example, Figure 3 shows the output displayed on the console windows of the Eclipse showing clock cycles and time required for execution of the encryption algorithm.

Upon compilation, the generated executable code reached a size of 4132 bytes including both the code and initialization data, whereas the free memory reserved for the heap stack was 3820 bytes. It is worth mentioning that the on-chip

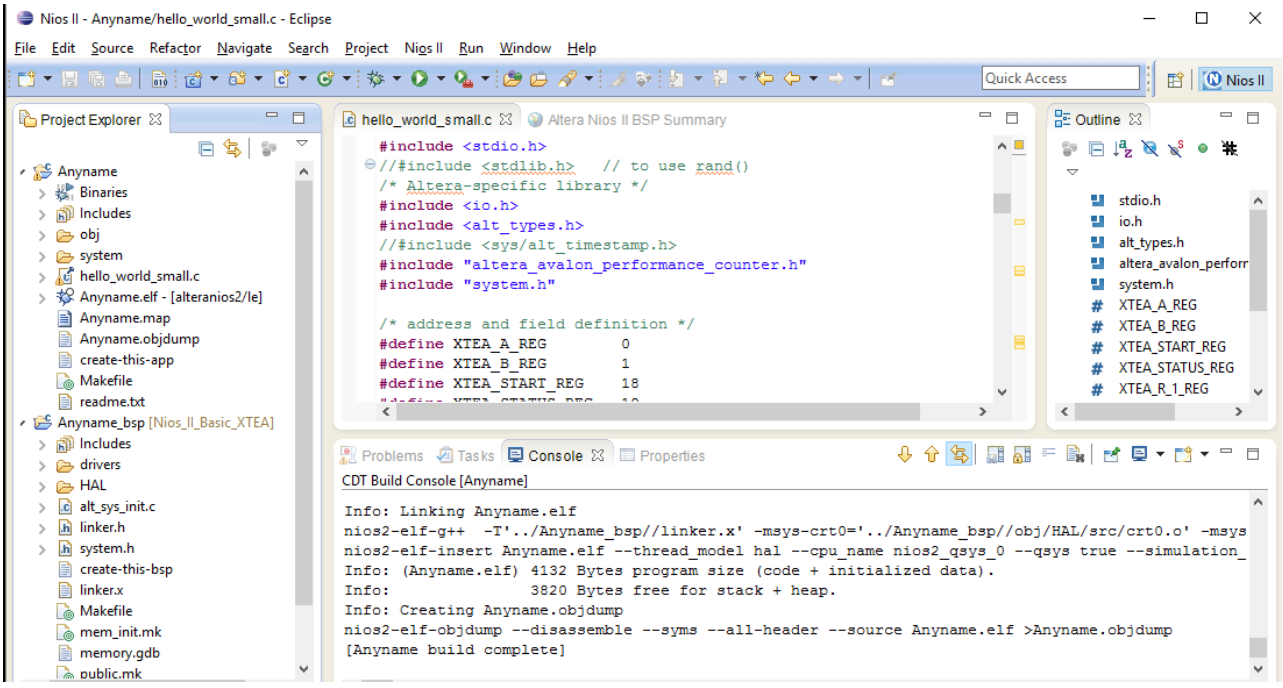

Figure 3. Console screen showing the output of the encryption program running in Nios II processor.

memory size was 8 KB starting from the address 0 × 2000 to the address 0 × 3FFF from the Nios II memory map.

This specific implementation was chosen as a reference benchmark to show how Full-Software implementation for the cryptographic algorithm compares to the Full-Hardware accelerator and to the software-hardware Co-Design.

4.3. Full-Hardware Implementation (FPGA)

The block diagram of the Full-Hardware for the system architecture is technically and fully presented in Figure 4. It shows a block diagram of the XTEA generic encryption engine known also as the encryption accelerator circuit. As shown in Figure 4, the two input data signals emerge from ports (block_in_0 and block_in_1) feeding the system with a 32-bit data source. On the other hand, the accelerator has another two output data signals ported to (v_0_out and v_1_out) offering 32-bit output sinks. In addition to those, one control input initiation signal (start) is introduced as well as one output status signal (done).

When start is set to 1, the FSM begins by taking new inputs. Consequently, the external upper level circuit (Wrapper) should place the dual 32-bit input data in the block_in_0 and block_in_1 registers, and enact the start signal for one clock cycle at least, to initiate the encryption or decryption operation. Once the encryption of the message is complete, the dual 32-bit output is latched back to v_0_out and v_1_out the output port registers, signaling the end of calculations. Accordingly and immediately, the done signal is set for one clock cycle and the computations stop.

We took the XTEA architecture as presented in the previous section and conversely implemented it as a VHDL model. The XTEA module was written in VHDL language as usual, but was compiled in the Quartus II environment used in this project. The RTL design generated from the VHDL was modelled initially using the Modelsim simulator of Mentor Graphics PE 14.1 targeting development, verifications, and functionality checking. In the last phase, the design was compiled and synthesised using the Quartus II.

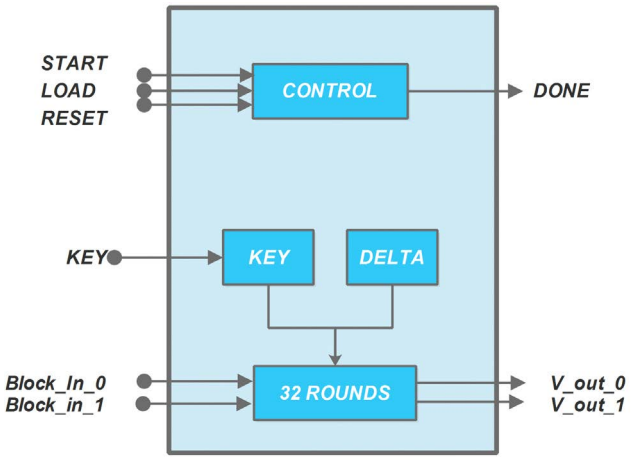

Figure 4. XTEA block diagram.

Figure 5 shows the VHDL hardware implementation model of the XTEA function. Using this configuration, the main hardware XTEA module is synthesised and replicated from 1 to 16 times within a Driver module. The Driver module is defined as a higher level VHDL-coded item that facilitates the transfer of data to the XTEA module(s) and receives the results from it (or them). In addition, it facilitates the replications of the main XTEA engine and instantiates the signals and buffers with registers related to all the replicated engines. Figure 6 shows a block diagram for four replication instances from an XTEA engine, connected together in parallel. As seen, it is assumed the feeding of the input data is committed in parallel using any outside parallel communication source giving 256 bits in parallel and resulting in 256 bits as well.

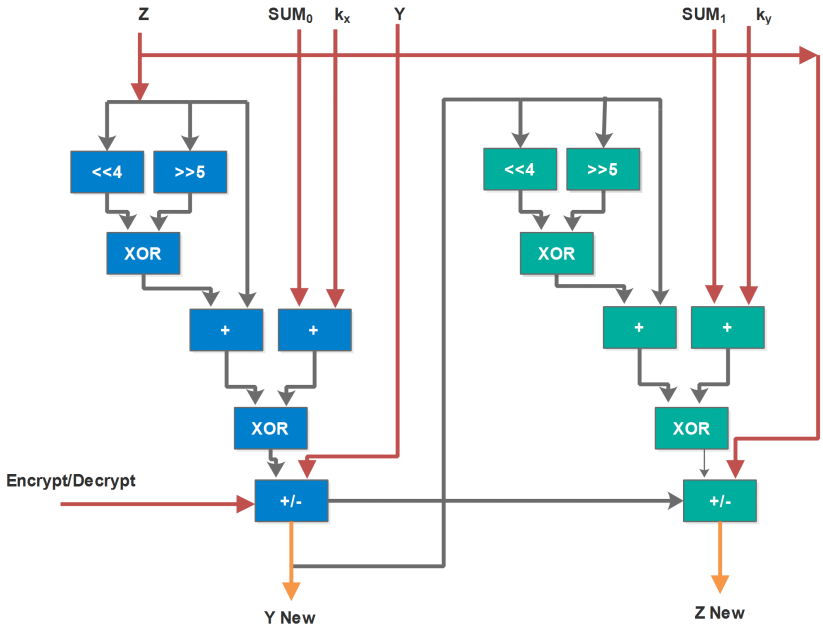

Figure 5. Hardware implementation diagram of XTEA encryption algorithm.

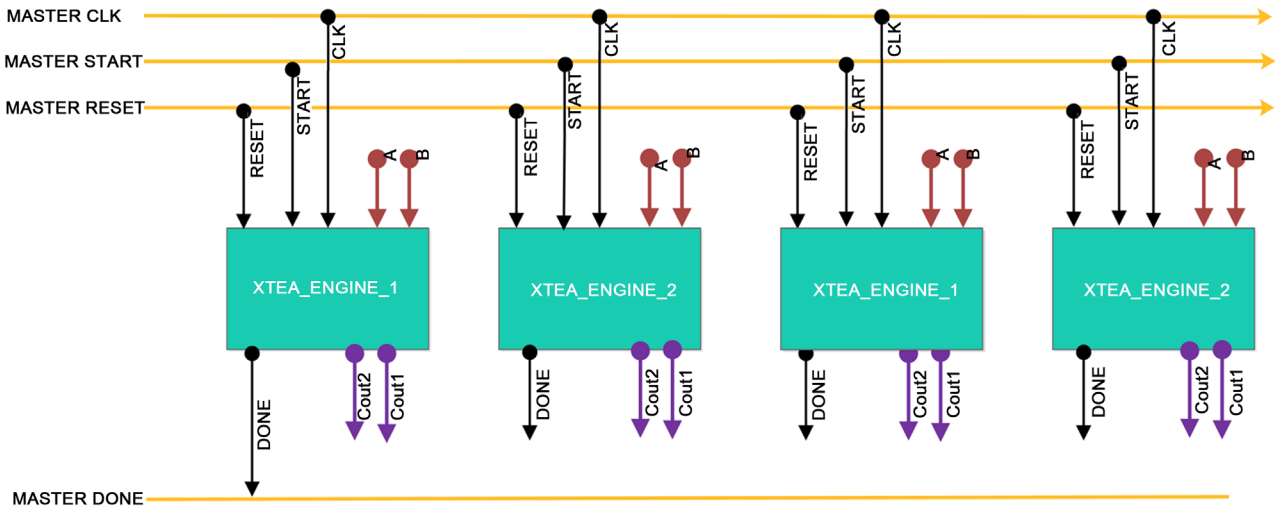

Figure 6. Full-hardware module with four replications.

The Chip Planner for the programmable chip produced from Quartus II is shown in Figure 7, displaying the relative area size of the design compared to the total area, where the highlighted portion illustrates the synthesised hardware XTEA engine with its wrapper. The two adjacent subfigures show the XTEA engine with a single synthesis, and with four synthesis replications. As can be seen, the occupied area of the design is quite small in proportion to the total area. Furthermore, the register transfer level (RTL) schematic of the XTEA for both a single and four replications is shown in the series of Figure 8 and Figure 9.

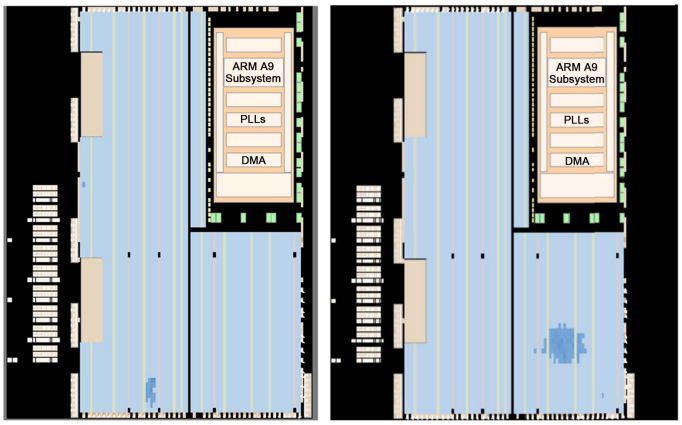

Figure 7. Chip planner diagram for one (left) and four (right) replication models.

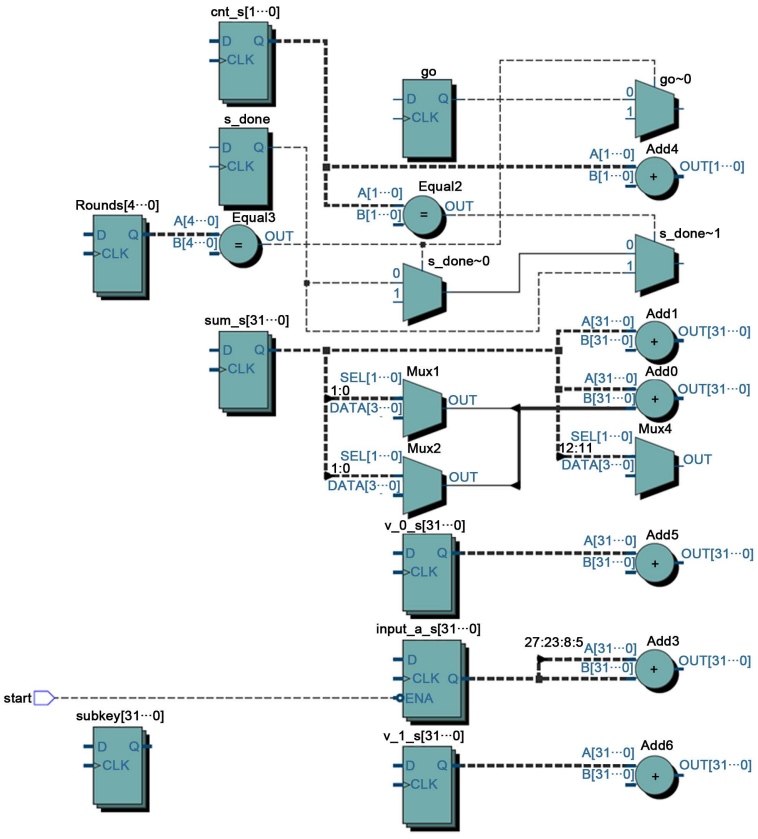

Figure 8. RTL viewer for basic XTEA encrypt circuit.

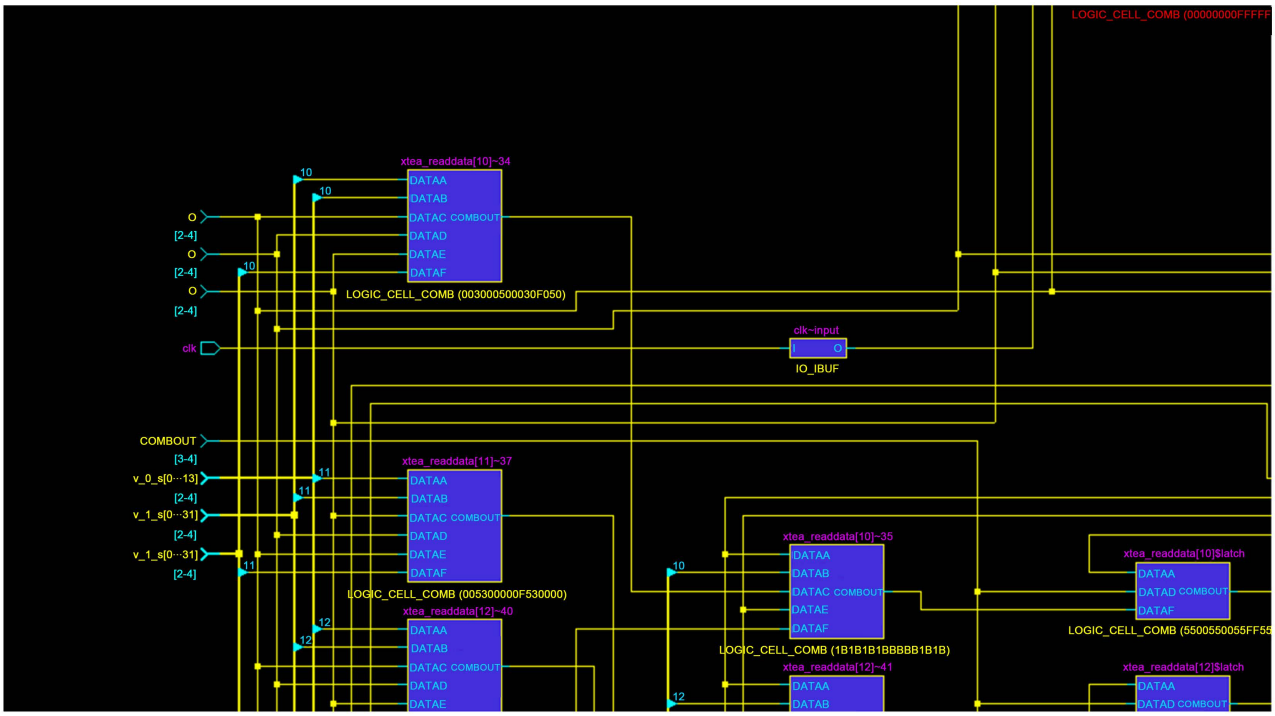

Figure 9. Partial view for technology map viwer (Encrypt plus wrapper of four replications).

Successful implementation of the Full-Hardware synthesis resulted in the highest performance as will be discussed in the results section, since the data feeding was controlled by software. The following describes the method undertaken to estimate the exact performance metrics for this configuration: from the maximum frequency given by the Quartus II synthesis, which was 200 MHz, and from basic knowledge of the 32 clock cycles needed to encrypt the message using the XTEA as well as the input message length of 64-bits, it could be determined that the throughput could be as high as 1.56 Gb/s when a maximum of 16 replications of the same XTEA engine is used.

A much greater throughput would be expected if we were to increase the number of replications (see the results section to view the throughput as a function of replications).

4.4. Co-Design Implementation

Regarding the implementation, the XTEA module was intended to be synthesised as described in the previous section, with the exception of being interfaced to the NIOS-II soft processor. C-based software is used to control the sending of blocks of test data to be encrypted or decrypted to the hardware module. The software running in the Nios II is responsible for creating from 1 to 8 K 64-bit words (64-bit formed as 32-bit v0 and v1, respectively) once, then be redirected to the XTEA hardware module. The processor sends the data to the internal registers of the XTEA hardware accelerator. Once the calculation is complete, the results are then sent to the output register and through the JTAG interface to be displayed on a prompt screen. The main objective is to have the algorithm be

computed using hardware, but the memory Read and Write operations directed to and from the hardware module to be handled by software that is offloaded from the computation process. The XTEA hardware module acts as a hardware accelerator implementing this algorithm

Figure 3 shows a snapshot of the Eclipse Editor taken while running C-based code from within the Nios II soft processor. While Figure 2 presents the Qsys screen showing the different embedded components that the Nios II processor interfaces with. Figure 10 shows an Avalon MM bus connected to four replications of the XTEA hardware module linked to the Nios II processor.

5. Results and Discussion

Figure 11 shows the hardware utilization obtained from the placement and routing report taken from Quartus II which includes: the number of Logic Elements LE, the number of Registers, and the maximum frequency, for the design of the XTEA_Wrapper circuit only. The synthesis of XTEA_Wrapper circuit utilizes from 0.4% to 4.0% of the total FPGA ALM resources and from 0.14% to 1.6% of the total FPGA Registers resources. Figure 12 provides a synthesis utilization report for the Co-Design with Nios II when synthesized with a number of replications of the main XTEA VHDL circuit. It was shown that the Nios II synthesis utilizes 2.4% of the total FPGA ALM resources and 0.82% of the total FPGA Registers resources.

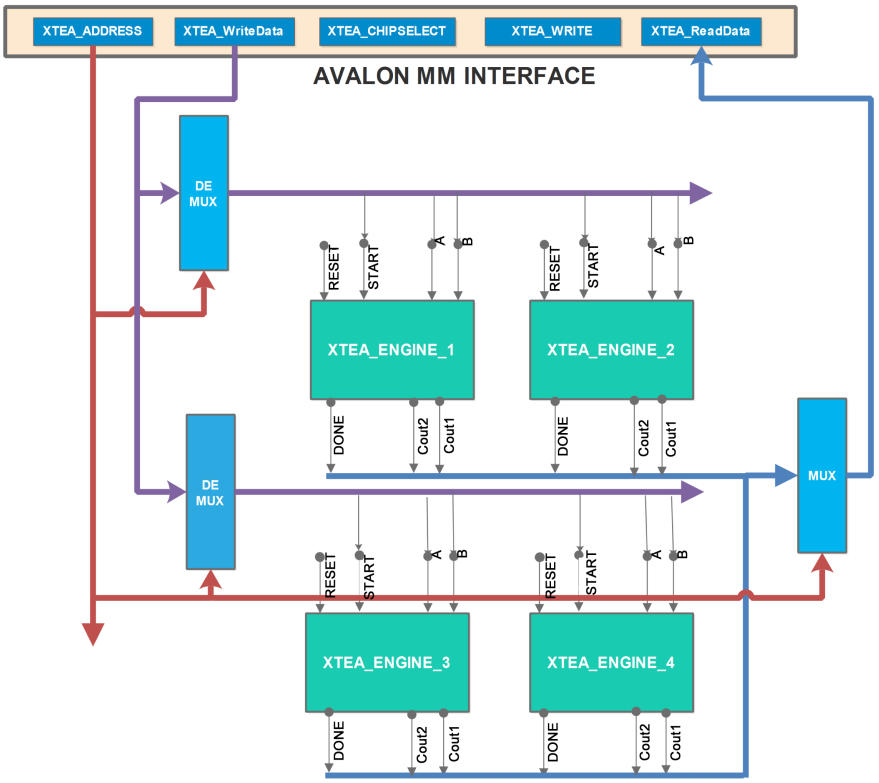

Figure 10. Avalon MM interconnect showing four replications of XTEA encryption engine.

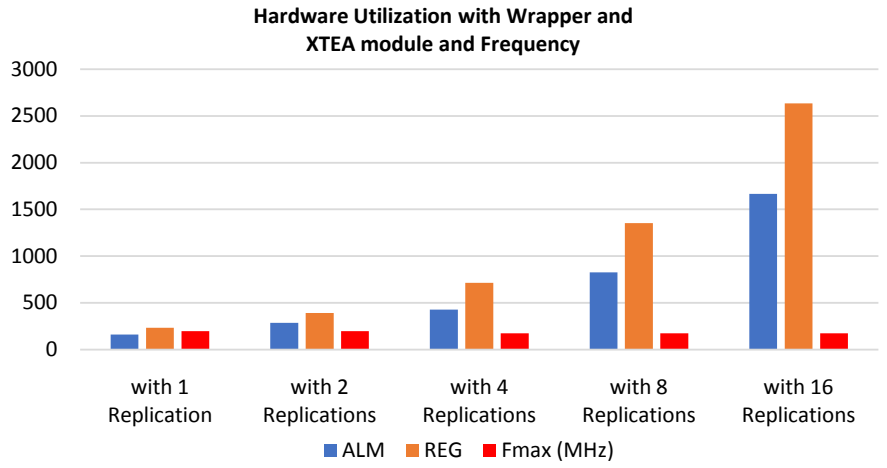

Figure 11. Hardware utilization ALMs, REGs, beside maximum frequency attained when synthesizing wrapper circuit in conjunction with Encryption module.

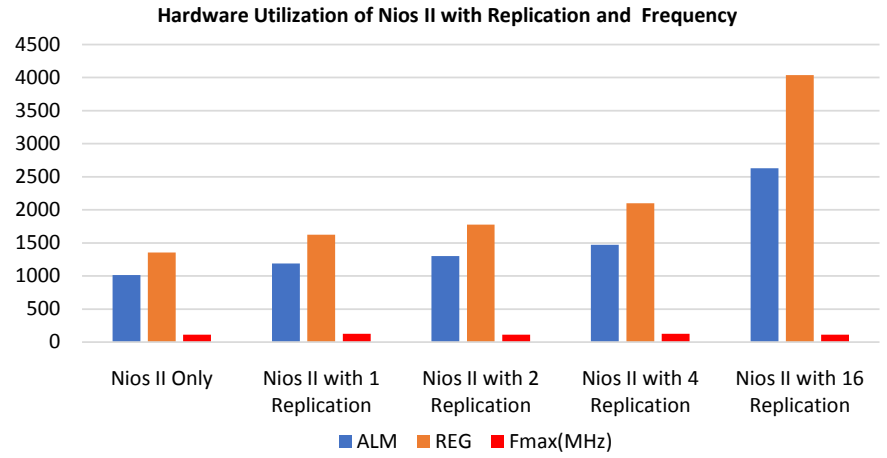

Figure 12. Hardware utilization of ALMs, REGs, beside maximum frequency attained when synthesizing Nios II soft processor in conjunction with XTEA accelerator component.

The low utilization values appear to be useful, since more programmable logic resources can be dedicated towards implementing other computationally intensive sections of the original application, or even a towards more replications to raise the total efficiency or throughput. Just to prove this theory, if 4% is utilised to implement the XTEA_Wrapper module exhibiting only 16 replications, then by utilizing full FPGA resources, a total of 400 replications could be exhibited.

Apart from the synthesis results, a performance experimentation was carried out as well. The three configurations were tested by creating arrays of data ranging from 1 block of 64-bit up to 8 K blocks of 64-bit random hexadecimal integer numbers in multiple of base-2 increments.

Namely, in the Full-Software configuration, the NIOS-II soft processor executed the XTEA encryption and decryption routines. The tested data reached 8K double words (64-bit) processed by this algorithm and the corresponding time stamp was recorded. Table 2 shows the results for this configuration, where

Table 2. Speed up with throughput of the Nios II only (full software) implementation of the XTEA encryption vs. Nios II equipped with XTEA accelerator component.

Microsoft	Looping on SW code Nios II Only CLKs with Nios II Only	Looping on HW VHDL 1 Replication × N times CLKs with Nios II and 1 Replication	Speed Up	Throughput 1	Throughput 2
1	7180				
2					
4		1215			
8	58,923	2339	25	990,581	24,954,254
16	117,649	4587	26	992,240	25,449,313
32	235,105	9083	26	993,054	25,704,283
64	470,017	18,075	26	993,462	25,833,693
128	941,043	36,059	26	992,397	25,898,888
256	1,879,489	72,027	26	993,768	25,931,609
512	3,758,785	143,963	26	993,819	25,948,001
1024	7,517,377	287,835	26	993,845	25,956,204
2048	15,034,561	575,579	26	993,857	25,960,308
4096	30,068,929	1,151,067	26	993,864	25,962,360
8192	60,137,665	2,302,043	26	993,867	25,963,386

the first column shows the number of inputs processed, the second column shows the software execution by clock count with regard to the maximum frequency of 114 MHz, and the fifth column shows the throughput (approximately 1 Mb/s).

Furthermore, Table 2 summarizes the performance of the Co-Design configuration of the Nios II processor when synthesised with a single replication of the XTEA circuit. The third column shows the clock count for this configuration, and the sixth column shows its throughput. The table also shows, in the last column, the speed-up between the two configurations. Figure 13 shows a comparison between Full-Software and Co-Design throughputs versus different block widths. The throughput of the Full-Software Nios II implementation reached 0.99 Mb/s, or nearly 1 Mb/s, while that for the Nios II with one replication model reached 25 - 26 Mb/s.

For the Co-Design configuration with M-replications, Table 3 represents the execution times for different replications ranging from 2, 4, 8, up to 16 replications. The Co-Design configuration was selected to test the configuration. In this experiment, data was sent as a double input (64-bit) to each of the instances of all XTEA replications in series, and a run was triggered once but in parallel at the end for the all XTEA instances. Finally, the time stamp for each of the activities was recorded. The first column shows the replication number, while the second shows the run clock count, bearing in mind the max frequency that this

Table 3. Speed up with Throughput of the. Nios II equipped with XTEA accelerator implementation of the XTEA encryption vs. Full-hardware of the XTEA component.

Replications	Nios II + Replications	Hardware CLKs	TP1	TP2	SpeedUp
1		128		98,000,000	
2	299	128	48,802,676	196,000,000	4
4	403	128	72,416,873	392,000,000	5
8	611	128	95,528,642	784,000,000	8
16	1027	128	113,666,991	1,568,000,000	14

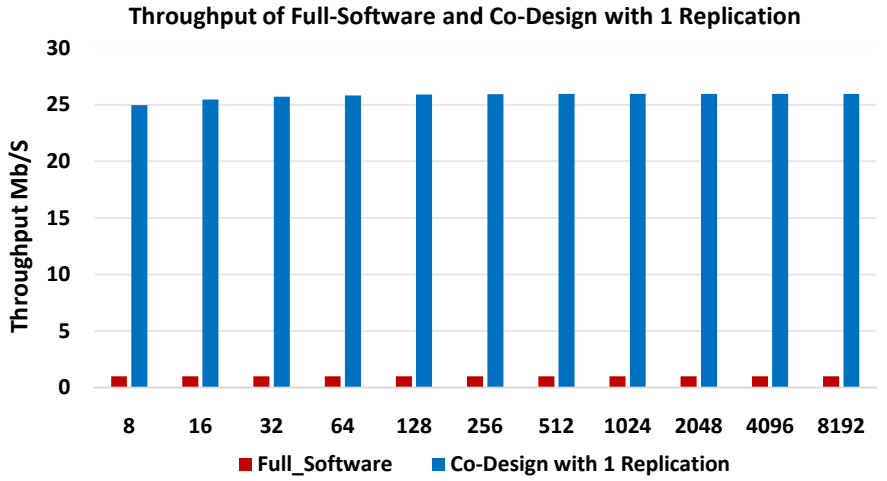

Figure 13. Throughput comparison for Full_Software and Co_Design implementations. The difference is fixed to 25 times and the throughput is fixed whatever the change of the block size operated on was selected.

configuration could operate on was 114 MHz. Figure 14 compares the throughput between the configurations of Full-Hardware and Co-Design, highlighting the different replications of the XTEA engine versus replications from 1 to 16.

Figure 15 shows a speed comparison for all of the implementations as follows: Full-Software processing arrays of 8 of 64-bit inputs (512 bits), Nios II_Co-Design with a single replication processing the same array, Nios II_Co-Design powered with eight replications, and finally Full-Hardware powered with eight replications.

As can be seen, the Nios II_Co-Design with a single hardware replication provided a throughput speed-up of 25× the Full-Software speed for the 512-bit data blocks processed. Meanwhile, the Nios II_Co-Design with eight replications provided a throughput speed-up of ~4× the speed of the Nios II_Co-Design with a single replication for the same data block processed. Finally, the Full-Hardware solution provided a throughput speed-up of ~7× the speed of the Nios II_Co-Design with eight replications. This speed-up was observed because of the ability of the FPGAs to compute streaming data via multiple instances of the datapath architecture, which means doing calculations in full concurrency.

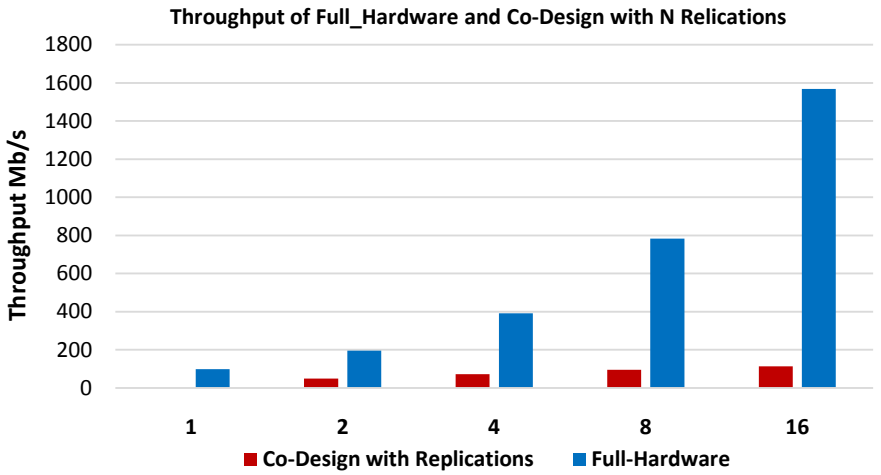

Figure 14. Hroughput comparison for Full_Hardware and Co_Design implementations with replications. As each core performs its operations in fixed 128 Clk cycles performing encryption or decryption operated on single block of 64-bit data and by knowing Fmax freuency beside the number of bytes processed by all cores. Then all that will lead to the definite calculation the throughput for the full-hardware solution.

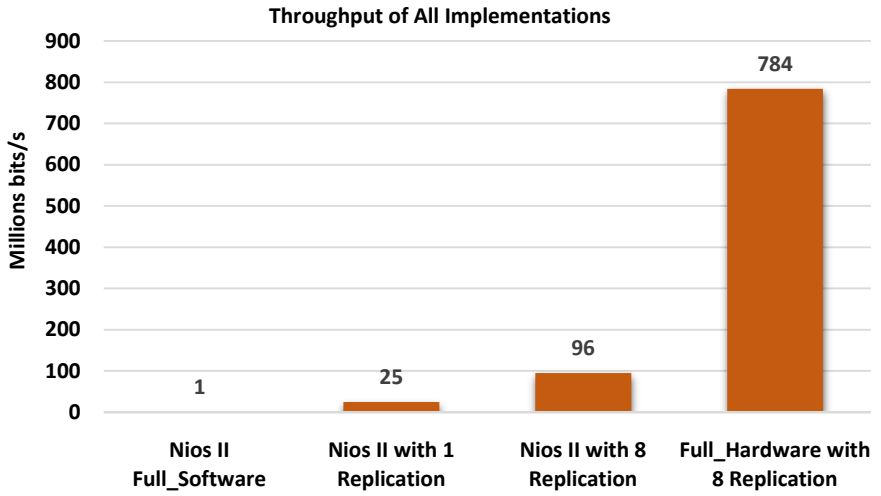

Figure 15. Throughput comparison for all the implementations.

In general, the XTEA circuit exhibited a fixed latency of 128 clock cycles when performing encryption or decryption operated on single block of 64-bit data. In addition, the placement and routing processes provided an achievable clock period of 7 ns. Those two values could be used to facilitate the calculation of throughputs as needed, such that the throughput calculations reported in this work are based on the following equation:

$$\text{Throughput} = (\text{no. of bits processed})/(\text{no. of clock cycles} \times \text{clock period}) \quad (1)$$

In summary, the Full-Hardware solution has always proved to yield a maximum encryption performance; nevertheless, utilizing multiple replications of the same encryption engine in VHDL notably raised the performance as this work shows.

6. Conclusion

In this article, the performance of the XTEA lightweight cryptography algorithm used in a soft processor CPU and FPGA is compared. We presented a methodology for interfacing an advanced XTEA in custom hardware with a system designed around a Nios II soft core processor in addition to a software alone design. Targeted hardware replications on the XTEA encryption engine were successfully used to increase the throughput. We were able to show that replications on FPGAs can add to the throughput and increase utilization. This work has outlined a co-design approach to synthesizing cryptographic algorithms of the XTEA type, but other cryptographic algorithms may be used via the same implementation approach.

Future Works

The future works could be summarized in porting other light weight cryptographic algorithms in FPGA.

Conflict of Interests

The author declares that there is no conflict of interest regarding the publication of this paper.

References

[1] Lee, D., Kim, D.C., Kwon, D. and Kim, H. (2014) Efficient Hardware Implementation of the Lightweight Block Encryption Algorithm LEA. *Sensors*, **14**, 975-994.

[2] Yalla, I.P. and Kaps, J. (2009) Lightweight Cryptography for FPGAs. *International Conference on Reconfigurable Computing and FPGAs*, Quintana Roo, 225-230. https://doi.org/10.1109/ReConFig.2009.54

[3] Arora, N. and Gigras, Y. (2014) FPGA Implementation of Low Power and High Speed Hummingbird Cryptographic Algorithm. *International Journal of Computer Applications*, **92**, 42-47. https://doi.org/10.5120/16097-5423

[4] Needham, D.J. and Wheeler Roger, M. (1995) TEA, a Tiny Encryption Algorithm. 2*nd International Workshop Fast Software Encryption*, Leuven, 14-16 December 1994, Vol. 2.

[5] Needham, R.M. and Wheeler, D.J. (1997) Tea Extensions. Report, Cambridge University, Cambridge.

[6] Kaps, J.-P. (2008) Chai-Tea, Cryptographic Hardware Implementations of XTEA. In: *International Conference on Cryptology in India*, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-89754-5_28

[7] Israsena, P. (2006) On XTEA-Based Encryption/Authentication Core for Wireless Pervasive Communication. *International Symposium on Communications and Information Technologies*, Bangkok, 18-20 October 2006, 59-62. https://doi.org/10.1109/ISCIT.2006.339887

[8] Israsena, P. (2006) Securing Ubiquitous and Low-Cost RFID using Tiny Encryption Algorithm. 1*st International Symposium on Wireless Pervasive Computing*, 16-18 January 2006, 4.

[9] Israsena, P. (2005) Design and Implementation of Low Power Hardware Encryption

for Low Cost Secure RFID Using TEA. 2005 *Fifth International Conference on Information, Communications and Signal Processing*, IEEE, 1402-1406.

[10] Khan, G.N. and Zhu, G. (2013) Secure RFID Authentication Protocol with Key Updating Technique. 22*nd International Conference on Computer Communications and Networks*, Nassau, 30 July-2 August 2013, 1-5. https://doi.org/10.1109/ICCCN.2013.6614192

[11] Khan, G.N., Yu, X. and Yuan, F. (2011) A Novel XTEA Based Authentication Protocol for RFID Systems. *General Assembly and Scientific Symposium*, Istanbul, 13-20 August 2011, 1-4.

[12] Khan, G.N. and Moessner, M.B. (2011) Secure Authentication Protocol for RFID Systems. *Proceedings of* 20*th International Conference on Computer Communications and Networks*, Maui, 31 July-4 August 2011, 1-7. https://doi.org/10.1109/ICCCN.2011.6006010

[13] Gaba, S., Aggarwal, I. and Pandey, S. (2012) Design of Efficient XTEA using Verilog. *International Journal of Scientific and Research Publications*, **2**.

[14] Hussain, M.A. and Badar, R. (2015) FPGA Based Implementation Scenarios of TEA Block Cipher. 13*th International Conference on Frontiers of Information Technology*, Islamabad, 14-16 December 2015, 283-286. https://doi.org/10.1109/FIT.2015.56

[15] Al Maashri, A., *et al.* (2016) Optimized Hardware Crypto Engines for XTEA and SHA-512 for Wireless Sensor Nodes. *Indian Journal of Science and Technology*, **9**, 1-7.

[16] Venugopal, V. and Shila, D.M. (2013) High throughput Implementations of Cryptography Algorithms on GPU and FPGA. *Instrumentation and Measurement Technology Conference*, Minneapolis, 6-9 May 2013, 723-727. https://doi.org/10.1109/I2MTC.2013.6555510