# Efficient Tiny Hardware Cipher under Verilog

Issam Damaj
Dept. of Electrical and Computer Engineering
Dhofar University
P.O. Box 2509, Salalah 211, Oman
i_damaj@du.edu.om

Samer Hamade, and Hassan Diab
Dept. of Electrical and Computer Engineering
American University of Beirut
P.O. Box 11-0236, Beirut, Lebanon
smh22@aub.edu.lb, diab@aub.edu.lb

**KEYWORDS**
Gate Arrays, Cryptography, Algorithms, Hardware Design

## ABSTRACT

Embedded hardware security has been an increasingly important need for many modern general and specific purposes electronic systems. Minute security algorithms with their expected low-cost and high-speed corresponding hardware realizations are of particular interest to fields such as mobile telecommunications, handheld computing devices, etc. In this paper, we analyze and evaluate the development of a cheap and relatively fast hardware implementation of the extended tiny encryption algorithm (*XTEA*). The development will start by modeling the system using finite state machines (*FSMs*) and will use *Verilog* hardware description language to describe the design. Minimizing the chip area will be our primary target rather than the construction of a multi-way massively parallel implementation with its expected high-speed and large silicon area. Many hardware design tools are used to try reaching the best possible optimized syntheses. The targeted hardware systems are the reconfigurable *Altera's Stratix II* and *Xilinx Virtex II Pro* modern field programmable gate arrays (*FPGAs*).

## INTRODUCTION

Security of information has become a main issue in the ever evolving world of small mobile devices such as personal digital assistants (*PDAs*) and cell phones. In such minute devices, the fight over high performance and low power consumption, besides security, are primary targets. However, a great deal of assistance in creating low-power and high-speed cores, comes from the simplicity of the selected algorithm for embedding as a hardware component.

Many encryption algorithms are now available in the market (Kelsey et al. 1996), and the selection of a specific one is dependent on the relatively tight constraints in small devices. The selected algorithm should be small, relatively secure, with a proven history of overcoming possible well known attacks on it. The Tiny Encryption Algorithm (*TEA*) (Wheeler and Needham 1994), and hence its successor the Extended-TEAs (*XTEAs*) (Needham and Wheeler 1997; Russell 2004; Kelsey et al. 1997; Moon et al. 2002) are among the best choices available for the above taut requirements and to be implemented in the research in hand.

Other requirements are still of no less important than the issues of performance and power consumption; these include the ease of modifiability, upgradeability and reuse of the designed security components. The type of hardware circuits to be used for implementing the developed cores, largely affects the above modifiability properties. Here we propose reconfigurable computers; more specifically field programmable gate arrays (*FPGAs*), as a possible solution with their famous property of programmability to satisfy the addressed need for modifiability.

*FPGAs*, nowadays are important components of reconfigurable systems; they have shown a dramatic increase in their density over the last few years. For example, companies like *Xilinx* and *Altera* have enabled the production of *FPGAs* with several millions of gates, such as in *Virtex-II Pro* and *Stratix-II FPGAs*. The versatility of *FPGAs*, opened up completely new avenues in high-performance computing. These programmable hardware circuits are aided with various co-design tools and flexible design methodologies to form a powerful paradigm for computing.

The traditional implementation of a function on an *FPGA* is done using logic synthesis based on *VHDL*, *Verilog* or a similar *HDL* (hardware description language). These discrete event simulation languages are rather different from languages, such as *C*, *C++* or *JAVA*. Many *FPGA* implementation tools are primarily *HDL*-based and not well integrated with high-level software tools. Furthermore, these *HDL*-based *IP* (intellectual property) cores are expensive and they have complex licensing schemes. In the presented designs, the hardware implementations are carried out under *Verilog*, employing different co-design tools. The targeted *FPGA* systems are *Altera Startix II* and *Xilinx Vertix II Pro*. The hardware design tools involved in this project are *Altera's Quartus*, *Xilinx ISE*, *Mentor Graphics HDL Designer*, *Leonardo Spectrum*, *Precision Synthesis* , and *ModelSim*.

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

## THE TINY ENCRYPTION ALGORITHM

In cryptography, the Tiny Encryption Algorithm (*TEA*) is a block cipher notable for its simplicity of description and implementation (typically a few lines of code). The cipher was initially presented by (Wheeler and Needham 1994). *TEA* operates on 64-bit blocks and uses a 128-bit key. It has a Feistel structure with a suggested 64 rounds, typically implemented in pairs termed cycles. It has an extremely simple key schedule, mixing all of the key material in exactly the same way for each cycle.

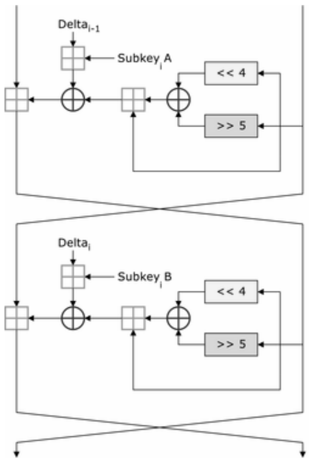

Figure 1. A single XTEA round with its internal computational constructs. The crossed square for the sum, crossed circle for an XOR, >> for a right shift, << for a left shift.

*XTEA* is a symmetric block cipher designed to correct weaknesses in *TEA*. Like *TEA*, *XTEA* is a 64-bit block Feistel network with a 128-bit key and a suggested 64 rounds. Several differences from *TEA* are apparent, including a somewhat more complex key-schedule and a rearrangement of the shifts, XORs and additions (Hong et al. 2003; Ko et al. 2004). Figure 1 show the block diagram of an XTEA single round.

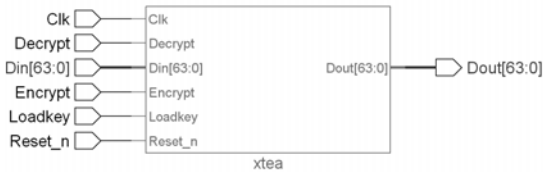

Figure 2. Block Diagram generated from the top model by Leonardo spectrum

High-speed hardware implementations of the *XTEA* under *VHDL* were suggested in (Ghazzawi et al. 2006). The power consumption of the *XTEA* were studied in (Kelsey et al. 1996) and compared with results for the *RC5* algorithm.

## XTEA HARDWARE

In Figure 2, the block diagram of the created chip is shown. The 128-bit key is input through the *Din* pins entered 64 bits at a time. The same *Din* pins are used to enter the plaintext (or ciphertext in the case of decryption) 64-bits at a time, while *Dout* pins are used for outputting the ciphered (or plaintext in the case of decryption). The remaining pins are for clocking (*Clk*) and control signals, moreover, loading the key (*Loadkey*), enabling encryption or decryption (*Encrypt* or *Decrypt*), and resetting the system (*Reset_n*).

The development of the *XTEA* core is started by creating a finite state machine (*FSM*) with four possible states. The system will be initially in its *IDLE* state till the control signals are received. The transition that takes you from the *IDLE* state to *BUSY_KEY* state is controlled by the external event *Loadkey*. The state *BUSY_KEY* is responsible for inputting the key. After finishing the key inputting process the system returns automatically to its *IDLE* state. The system will undergo a transition to its *BUSY_ENC* (encryption state) or *BUSY_DEC* (decryption state) according to the transitions controlled by the events expected on the *Encrypt* and *Decrypt* pins. The system continuous operation is done by returning to the IDLE state on finishing the encryption or the decryption is shown in Figure 3.

A single *XTEA* round is to be repeated 32 times. A sequential version on the round level would mean the creation of circuit corresponding for a single round, then the output is fed-back to the circuit to become the input of the following round; this is to happen 32 times. A different degree of parallelism could be reached if the designer decides to unroll the loop to construct a fully-pipelined network of rounds. In this paper, we show a sequential version avoiding any resources replication and accordingly any additional expected increased area, cost, and power consumption. In Figure 4, a circuit block diagram was obtained using *HDL Designer* from *Mentor Graphics*. In Figure 4, the feedback wires are clearly shown going from the *eb1* block to the *eb2* main block. The *eb2* block contains all the inputs and outputs, and synchronized by a master clock. The second block *eb1* contains continuous assignment statements for the main functions (or core) of the encryption (decryption) algorithm. The output generated by *eb1* is fed-back to *eb2* synchronized by the master clock.

As shown in Figure 1, an *XTEA* round implementation requires the construction of the following computational elements:

- Addition and Subtraction modulo 32.
- Bitwise XOR
- Shift left and right operations

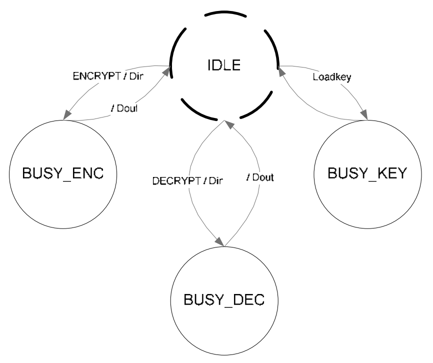

Figure 3. Finite State Machine

## PERFORMANCE EVALUATION

For the purpose of analysis we present first the results of testing the design using *Altera's Quartus* tool, where we build the simulation cases graphically. From the performed simulation, the number of clock cycles needed to complete a single encryption or decryption process is 68 cycles (key loading 2 cycles, encryption/decryption 32 * 2 cycles, and wait states 2 cycles). A pipelined version will, with no doubt, enhance the performance by decreasing the process total time, but as a quid pro quo for silicon area.

In Figure 5, we show the waveforms of testing the encryption process. The *reset* signal was activated at first to insure that all the registers are cleared before starting any operation, note that the asynchronous reset is active low once. After that the *Loadkey* control was activated for two clock cycles to store the selected key that will be used in the next process.

Now, the module is ready to either encrypt or decrypt. One can distinguish between these two by the control signals provided as an input to the system. After engaging the *Encrypt* signal, the system will enter the *BUSY_ENC* state, and will finish the encryption after 64 clock cycles. You can notice that the output is available at that time and that the system is returned to the *IDLE* state waiting to the next control signal to operate.

In Figure 6, we depict the waveforms of decryption state transition testing. The encryption and the decryption processes are quite the same in architecture, but the decryption operates in a reverse manner on the data, and uses a subtractor rather than an adder. This test shows the transition of the state from the *IDLE* to the *BUSY_DEC*, the output will be available after 64 clock cycles.

In Figure 7, we show the simulation for testing the key loading process. In order to reduce the number of IOs used, the key was distributed over the *Din* input pins. Knowing the fact that the key length is 128 bits while the Din is only 64 bits, we need two clock cycles to load the key to its internal register inside the *FPGA*. This will cost using the *Loadkey* control driving the system to the *BUSY_KEY* state. The *BUSY_KEY* state will need two clock cycles to exit and return to the *IDLE* state again.

The number of IOs needed in this project is fixed and can easily be calculated from the module's event list, this number is equal to 133 IOs. The next step of assessment is to map the developed design onto different *FPGA* systems comparing the use of resources in each case. The synthesis tools used in this comparison are *Xilinx XST*, *Mentor Graphics Precision Synthesis RTL and physical 2004*, *Altera's Quartus*, and *Leonardo Spectrum 2004*. Both, *Precision Synthesis* and *Leonardo Spectrum* are vendor free third party tools developed to synthesize popular *FPGAs*. In Table 1, we show the different findings of compiling the developed *XTEA* design to *Altera's Stratix II FPGAs* with three different sizes. In Table 2, we show the findings after compiling the design to *Xilinx Virtix II Pro FPGA*. The chart in Figure 8 shows clearly the maximum speed of 134 Mbps achieved by mapping our sequential small-sized design onto the Virtix II Pro *FPGA*.

It is clear from the results shown in this section that the aim of obtaining a relatively small area has been achieved. The advantage of having a small design with a small occupied area had its impact on speed, where a maximum speed of 134 Mbps was achieved with the suggested sequential design. A similar design achieved only a speed of 16.8 Mbps in (Ghazzawi et al. 2006), but was enhanced by eliminating the large area occupied by the controller part and accordingly reducing the propagation delay and the number of clock cycles. The enhanced design in (Ghazzawi et al. 2006) reached a speed of 800 Mbps; more manual designs where offered expecting speeds above 1 Gbps.

The tiny *XTEA* for sure is not comparable to any of the powerful ciphers like the *AES* finalists, but it had the following summarized advantages:

- Small expected hardware silicon area.
- It is relatively secure enough with a number of rounds above 16.
- Fast enough to accompany other projects.
- Low power consumption.

Based on the comparison done between various synthesis tools, the following is concluded:

- For a small application like the development of the *XTEA*, no big improvement is gained by selecting one synthesis tool over the others.
- Choosing different FPGA device from the same family will not affect largely the amount of occupied area.

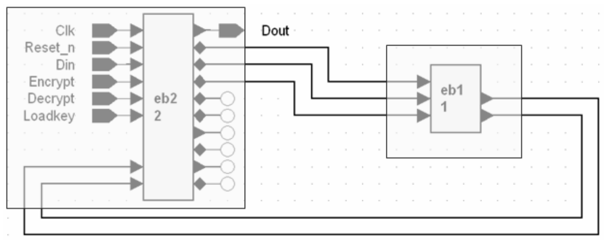

Figure 4. Block Diagram of the Implemented Sequential System

Increasing the depth of investigation concerning accelerating the *XTEA* would lead us, however, to parallel processing including pipelining. Thus, generic reasoning about the parallelization of the algorithm in hand is a possible extension for the proposed work, besides investigating the correctness of various parallel hardware implementations. Developing correct hardware leads to the adoption of a formalization framework. Through this formal mathematical framework, different parallel designs could be generated systematically, using provably correct rules of refinement. An example of such a framework is the *Bird-Meertens* Formalism (BMF) by which one generates data parallel programs from abstract specifications using the skeleton approach. The essence of this approach is to design a generic solution once, and to use instances of the design many times for various parallel architectures. Another frame work starts by formulating an algorithm by a generic formal functional specification step and generates parallel programs described in a concurrency framework - *CSP* (Communicating Sequential Processes). Through such developments, our implementations will benefit from the advances in the area of hardware/software co-design to generate efficient hardware *XTEA* circuits.

The generation of an efficient hardware solution for the *XTEA* would with no doubt satisfy the need for speed and efficiency. The parallelized designs could easily be mapped to various parallel architectures such as clusters, grids, *FPGAs*, complex programmable logic devices (*CPLDs*), dynamically reconfigurable systems (the *MorphoSys* (Bagherzadeh et al. 1999), etc. The availability of such systems with different sizes and speeds obliges us to study the parallelization of our algorithm not only in its sequential or pleasantly data-parallel version, but also with different degrees of parallelism. This will include reasoning about the use of pipelined blocks, partially sequential blocks, etc. Again, the parallelization is to be n a systematic parallelization framework which is done in a straightforward manner.

## CONCLUSION

The research presented in this paper is motivated by the need for low-power, fast, tiny, and cheap hardware security cores. We have presented a sequential, small-sized, relatively fast implementation of the extended tiny encryption algorithm (XTEA). The best achieved synthesis was by mapping the design onto a Virtex Pro II FPGA with a tiny area and a speed of 134 Mbps. The mapped design employed 32 rounds, although 16 rounds are assumed to be secure enough. Many extensions of the work in hand are present. The extensions could include the formal development, for the sake of correctness, of the XTEA and its successors the XXTEA and block XTEA. Multi-way massively parallel implementations are expected to increase the throughput at the expense of silicon area.

## REFERENCES

Bagherzadeh N. Kurdahi F. Singh H. Lu G. Lee M. and Filho E.1999. “MorphoSys: An integrated reconfigurable system for data-parallel computation-intensive applications.” IEEE Transactions on Computers.

Ghazzawi W., R. Saraeb, and I. Damaj. 2006. “Hardware Development of the Extended Tiny Encryption Algorithm,” in the ACS/IEEE International Conference on Computer Systems and Applications, Dubai/ Sharjah, United Arab Emirates (Mar).

Hong S., Deukjo Hong, Youngdai Ko, Donghoon Chang, Wonil Lee, and Sangjin Lee. 2003, “Differential cryptanalysis of TEA and XTEA.” In *Proceedings of ICISC 2003*, 2003b.

Ko Y., Seokhie Hong, and Wonil Lee. 2004. “Related key differential attacks on 26 rounds of XTEA and full rounds of GOST.”In *Proceedings of FSE '04*, Lecture Notes in Computer Science, Springer-Verlag,

Kelsey J., Bruce Schneier, and David Wagner.1997. “Related-key cryptanalysis of-WAY, Biham-DES, CAST, DES-X NewDES, RC2, and TEA.” *Lecture Notes in Computer Science*, 1334: 233-246.

Kelsey J., Bruce Schneier, and David Wagner. 1996. “Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES.” Lecture Notes in Computer Science, 1109: 237-251.

Moon D., Kyungdeok Hwang, Wonil Lee, Sangjin Lee, and Jongin Lim. 2002. “Impossible differential cryptanalysis of reduced round XTEA and TEA.”*Lecture Notes in Computer Science*, 2365: 49-60.

Needham R. M. and David J. Wheeler.1997. "Tea extensions." Technical report, Computer Laboratory, University of Cambridge(Oct).

Russell M. D. 2004. “Tinyness: An Overview of TEA and Related Ciphers.” http://www.users.cs.york.ac.uk/~matthew/TEA/TEA.html

Wheeler D. J. and R. M. Needham.1994. “TEA, a tiny encryption algorithm.” Fast Software Encryption, Leuven, LNCS 1008, , pp. 363-366.

**Issam W. Damaj** received his Bachelor of Engineering (B.Eng.) in Computer Engineering from Beirut Arab University in 1999 (with high distinction), and his Master of Engineering (M.Eng.) in Computer and Communications Engineering from the American University of Beirut in 2001. He was awarded his Ph.D. degree in Computer Science from London South Bank University, London, United Kingdom in 2004. Currently, he is an Assistant Professor of Electrical and Computer Engineering and the chairperson of the Department of Electrical and Computer Engineering, Dhofar University, Oman. His research interests include hardware/software co-design, embedded systems design, reconfigurable computing, parallel processing, and software engineering.

**Samer Hamade** is a graduate student at the American University of Beirut. He is a student in the Faculty of Engineering and Architecture, Department of Electrical and Computer Engineering. His major is Computer and Communications Engineering.

**Hassan B. Diab** received his B.Sc. (with Honors) in Communications Engineering from Leeds Metropolitan University, U.K. in 1981, his M.Sc. (with Distinction) in Systems Engineering from the University of Surrey, U.K. in 1982, and his Ph.D. in Computer Engineering from the University of Bath, U.K. in 1985. Dr. Diab is a Professor of Electrical and Computer Engineering at the Faculty of Engineering and Architecture, American University of Beirut (AUB), Lebanon and has over 22 years of experience. He has 116 publications in internationally refereed journals and conferences. His research interests include cryptography on high performance computer systems, modeling and simulation of parallel processing systems, embedded systems, reconfigurable computing, simulation of parallel applications, system simulation using fuzzy logic control, and the application of simulation for engineering education.

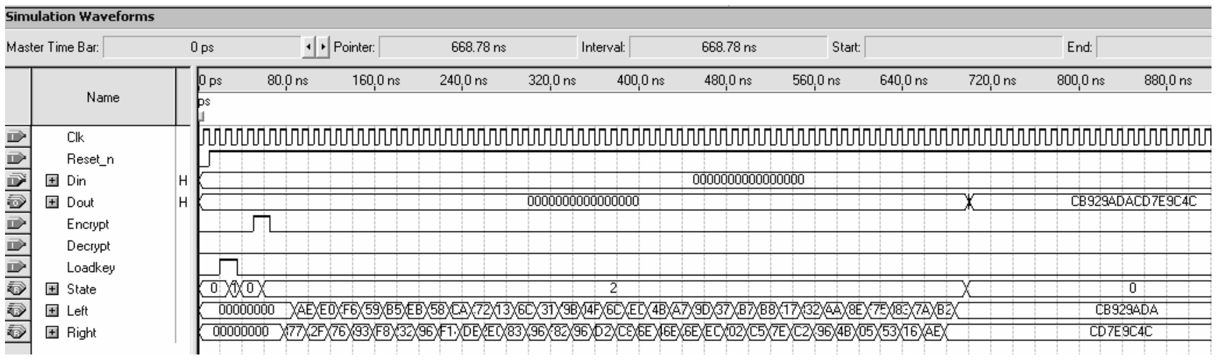

Figure 5. Test Case: Encryption Testing, Input Key: 0x0, Plaintext: 0x0, Ciphertext: 0XCB929ADACD7E9C4C

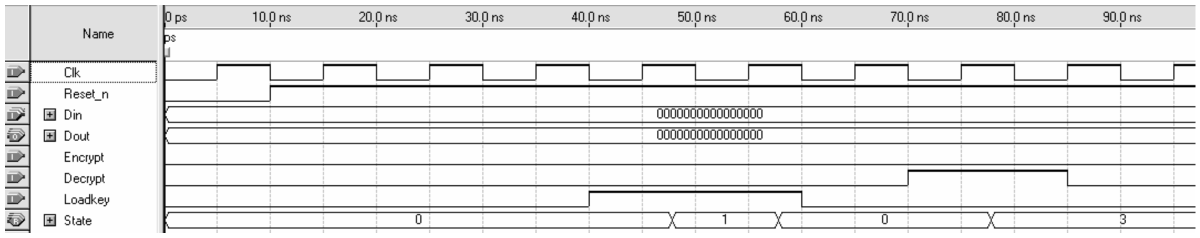

Figure 6. Test Case: Decryption State Transition

Table 2. Results of Mapping the Developed Design to *Xilinx Vertix II Pro FPGA*

| **Devices** | **Leonardo Spectrum** | **Xilinx XST** | **Precision Synthesis** |
|---|---|---|---|
| **Device Number:** | | **2VP2fg256** | |
| Number of IO: | | 133 out of 140 ( utilization = 94.3% ) | |
| CLBs Slices (408 available) | 294 | 393 | 539 |
| Maximum Frequency in MHz | 91.2 | 120.3 | 142.4 |
| Speed in Mbps | 85.83 | 113.22 | 134 |
| LUTs (2816 available) | 588 | 634 | 624 |
| Slice Flip Flops (2816 available ) | 298 | 307 | 305 |

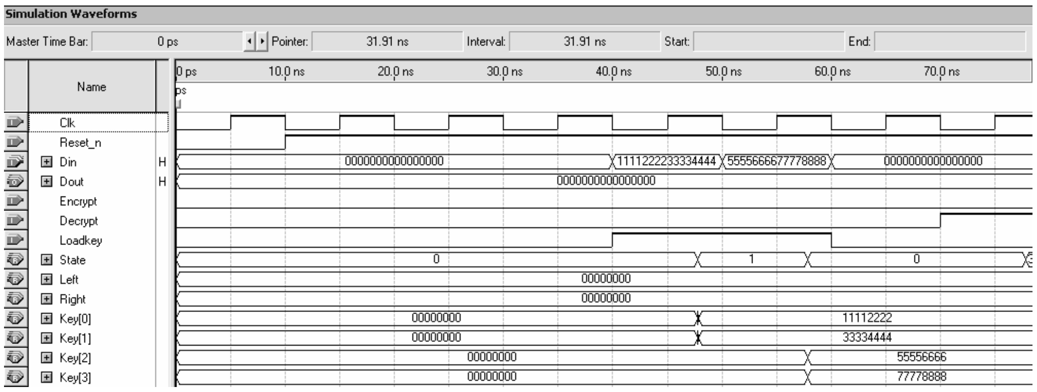

Figure 7. Test Case: Key Loading.

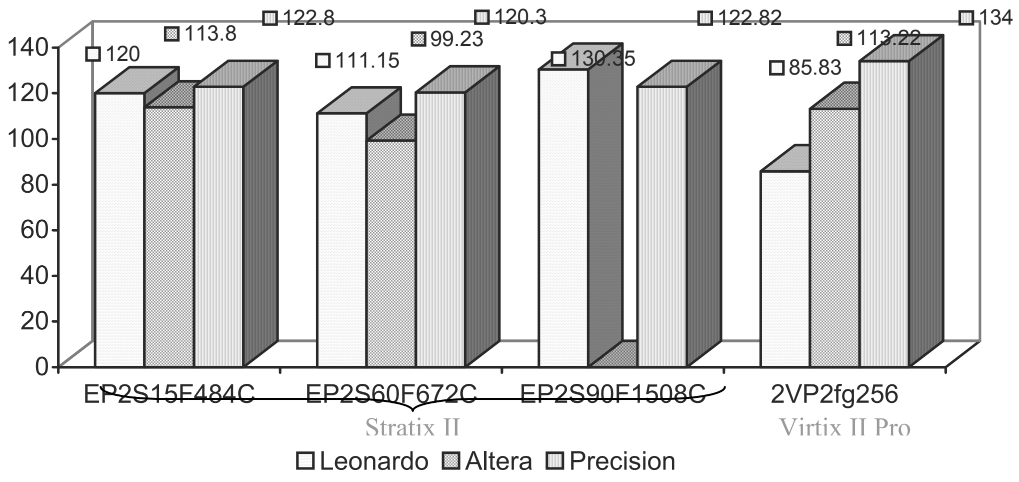

Figure 8. Speeds in Mbps of the Developed Implementations.

Table 1. Results of Mapping the Developed Design to *Altera's Stratix II FPGAs* with Three Different *FPGA* Sizes.

| **Devices** | **Leonardo Spectrum** | **Altera Quartus** | **Precision Synthesis** |
|---|---|---|---|
| **Device Number:** | | **EP2S15F484C** | |
| Number of IO: | | 133 out of 343 (utilization = 38.78% ) | |
| LUTs used (12480 available) | 526 | 573 | 539 |
| Maximum Frequency in MHz | 127.5 | 120.95 | 130.5 |
| Speed in Mbps | 120 | 113.83 | 122.82 |
| Registers used (14410 available) | 297 | 297 | 305 |
| **Device Number:** | | **EP2S60F672C** | |
| Number of IO: | | 133 out of 493 (utilization = 26.98%) | |
| ALUTs used (48352 available) | 526 | 573 | 539 |
| Maximum Frequency in MHz | 118.1 | 99.23 | 120.3 |
| Speed in Mbps | 111.15 | 93.39 | 113.22 |
| Registers used (51182 available) | 297 | 297 | 305 |
| **Device Number:** | | **EP2S90F1508C** | |
| Number of IO: | | 133 out of 903 (utilization = 14.73%) | |
| ALUTs used (48352 available) | 526 | NA | 539 |
| Maximum Frequency in MHz | 138.5 MHz | NA | 130.5 MHz |
| Speed in Mbps | 130.35 | NA | 122.82 |
| Registers used (51182 available) | 297 | NA | 305 |