

Energy and Performance Evaluation of an FPGA-Based SoC Platform with AES and PRESENT Coprocessors

Xu Guo, Zhimin Chen, and Patrick Schaumont

Virginia Tech, Blacksburg VA 24061, USA
{xuguo, chenzm, schaum}@vt.edu

Abstract. Hardware implementations of block ciphers have been intensively evaluated for years. The hardware profile, including the performance, area and power of a block cipher, only considers the block cipher as a standalone component, and does not consider it as a coprocessor in a system design. In this paper we consider system integration of AES and PRESENT crypto coprocessors, and analyze the system profile in a co-simulation environment and then on an actual FPGA-based SoC platform. Energy, performance and implementation results for both the AES- and PRESENT-based systems are presented. Our research emphasizes the need to consider energy efficiency and performance at system-level when evaluating a block cipher for real embedded systems. Simulation results reveal that the hardware/software interfaces, as the communication bottleneck, have major impact on the system performance. Experimental results further demonstrate that the PRESENT, a power-efficient lightweight block cipher with lower security level, becomes less energy-efficient than AES when system-integration overhead is included.

1 Introduction

In recent years, Field Programmable Logic Arrays (FPGAs) have had major impact on hardware/software codesign. Compared to the early frequent use as devices for rapid prototyping, FPGAs are now used for final products, thanks to their reduced time-to-market and the cost advantages of standard devices. Due to the importance of reconfigurable devices, numerous FPGA AES implementations have been published, most of which focus on high throughput rates [1, 2]. In [3], an AES design achieves a throughput of 25 Gb/s on a Xilinx Spartan-3 FPGA. This number only reflects the raw processing ability of the hardware to encrypt bits. However, FPGAs are now becoming a preferred platform for System-on-Chip (SoC). By providing hard and soft embedded processors on FPGAs, they enable on-chip integration of co-processors and processors. If we re-examine the above high throughput designs in the context of a SoC system, the communication bandwidth between system components becomes a critical design factor. For example, If we only consider an AES coprocessor that runs at 100 MHz and requires 11 clock cycles per encryption round, each round requires a 128-bit key, 128-bit plaintext and 128-bit ciphertext, then we need an

input/output bandwidth of about 3.5Gb/s. Dedicated communication hardware (e.g. direct-memory-access (DMA) chips on fixed-latency buses) may achieve this bandwidth. In many cases however, this bandwidth needs to be provided directly through the software. The bandwidth of 3.5Gb/s indeed is outside the capability of most embedded processors [4].

This shows the most optimal hardware design (in terms of performance) may not always be the most optimal solution at system level. In fact, not only the performance, but also the power or energy efficiency should be re-considered at system-level. In this paper, we consider AES and PRESENT for hardware acceleration, and using hardware/software interfaces provided with StrongARM and Microblaze processors. The results for StrongARM are estimated using cosimulation [5]. The results for Microblaze have been implemented on an FPGA board and measured using a hardware timer. In addition, power estimation was performed at system level using Xilinx XPower.

The contribution of this article is two-fold: (1) to present a system-level design flow, covering simulation up to FPGA implementation, that evaluates the performance and power consumption of a crypto coprocessor integrated in a complete system; (2) to point out that a lightweight and power-efficient cipher (PRESENT) integrated in a SoC environment may actually be less energy-efficient than a standard block cipher (AES).

The paper is organized as follows. Section 2 briefly presents the background of PRESENT block cipher. Section 3 explains the system-level design flow used in the paper and performs some analysis on the performance and power consumption under co-simulation environment. Section 4 describes the FPGA-based SoC design and illustrates the experimental results. Section 5 concludes the paper.

2 PRESENT Block Cipher

Although Rijndael has been selected by the American National Institute of Standards and Technology (NIST) as the Advanced Encryption Standard (AES) after a critical assessment, which included extensive benchmarking on a variety of platforms ranging from smart cards [6] to high end parallel machines [7], still many new block ciphers were proposed with special implementation properties, such as TEA, IDEA, Hight, Clefia, DESXL, and PRESENT [8]. In this paper, we are especially interested in comparing the AES with the newly published PRESENT block cipher, which was designed with area and power constraints uppermost in mind.

PRESENT is an SPN-based (substitution permutation network) block cipher with 31 rounds, a block size of 64-bit, and a key size of 80- or 128-bit. Fig. 1 shows the top level algorithmic description and hardware structure of PRESENT. It comprises three stages: a key-mixing step, a substitution layer, and a permutation layer. For the key mixing, simply a XOR is used. The key schedule consists essentially of a 61-bit rotation together with an S-box and a round counter (Present-80 uses a single Sbox, whereas Present-128 requires two S-boxes). The substitution layer comprises 16 S-boxes with 4-bit inputs and 4-bit outputs.

Similar S-boxes are used in both the data path and the key scheduling. The permutation layer is a simple bit transposition and can be realized by simple wiring [9].

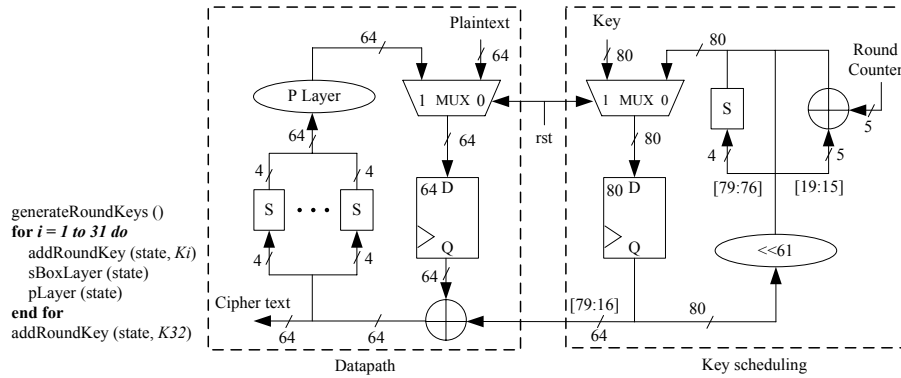


Fig. 1. Algorithmic description and hardware structure of PRESENT-80

3 System-Level Design and Analysis Using GEZEL

In order to narrow the gap between performance and flexibility, reduce the time required to complete a design and reduce the risk of errors that might result from translating a high-level prototype (e.g. C model) into HDLs, we use GEZEL to perform system-level design.

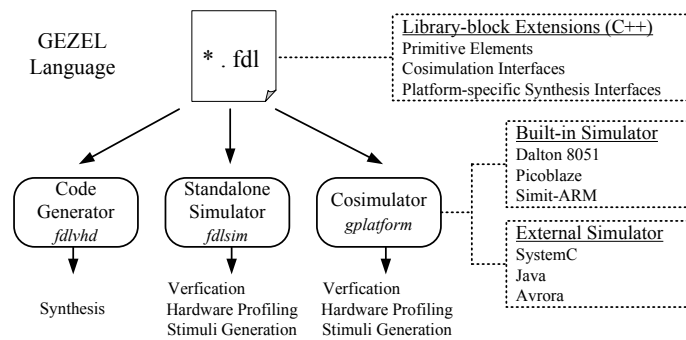


Fig. 2. Overview of GEZEL cosimulation environment

The GEZEL cosimulation environment creates a platform simulator by combining a hardware simulation kernel with one or more instruction-set simulators. The hardware part of the platform is programmed in GEZEL, a deterministic,

cycle-true and implementation-oriented hardware description language. After cycle-accurate simulation, the GEZEL description of hardware can be converted into synthesizable VHDL [10].

3.1 Hardware/Software Interfaces

There are three commonly available hardware/software interfaces: direct connection busses (e.g. Fast Simplex Link), processor local busses and general-purpose system busses (e.g. On-chip Peripheral Bus). Direct-connection buses and processor local busses are processor specific, while system busses are generic. In this research, we will only discuss the design using OPB system bus.

The OPB interface is a traditional memory-mapped interface for peripheral components. The OPB bus is a shared, variable latency bus which is part of IBM's CoreConnect specification. It is also used to interconnect soft- and hard-core processors in a Xilinx FPGA. The hardware side of an OPB interface consists of a decoder for a memory-read or memory-write cycle on a selected address in the memory range mapped to the OPB. The decoded memory cycle is translated to a read-from or a write-into a register in the coprocessor. A memory-mapped interface is an easy and popular interface technique, in particular because it works with standard C on any core that has a system bus. The drawback of this interface is the low-speed connection between hardware and software. Even on an embedded core, a simple round-trip communication between software and hardware can run into several tens of CPU clock cycles [4].

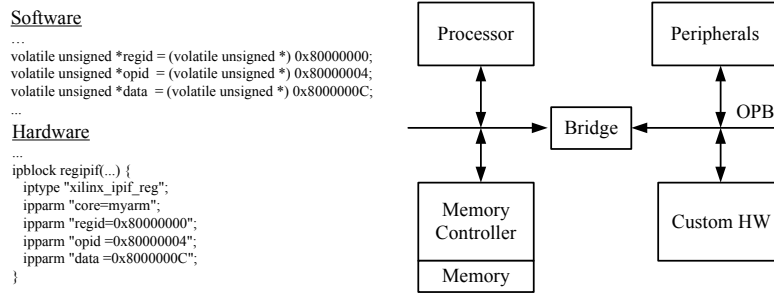


Fig. 3. Hardware/Software interface OPB

3.2 Cosimulation Based on StrongARM

Under the GEZEL simulation environment we first implement the AES and PRESENT in GEZEL based on based on Finite State Machine with Datapath (FSMD) model. A standalone simulation is then used to verify the correctness of the AES and PRESENT encryption core. Next, the AES and PRESENT cores are integrated into a coprocessor shell as follows. Three memory-mapped registers have been added: a data-input port, a data-output port, and control

port. Since the maximum data width supported by the OPB bus is 32-bit and the AES-128 and PRESENT-80 used in this paper have 128-bit, 80-bit and 64-bit ports, additional registers should be added to perform serial-to-parallel and parallel-to-serial conversions. The control shell also contains a dedicated controller that controls the operations of the hardware/software interface, which is OPB interface in our design. This controller implements the 'instruction-set' for the coprocessor, and decodes the commands sent from the software driver to the coprocessor. The final step is to write a software driver to perform a series of memory reads and writes.

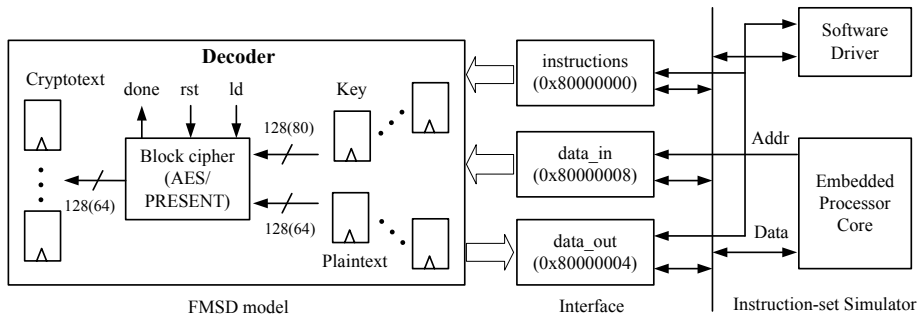


Fig. 4. Cosimulation based on instruction-set simulators

The simulation results for the AES and PRESENT under GEZEL environment are illustrated in Table 1.

Table 1. Cosimulation performance results (100 encryptions for each block cipher)

	SW	HW	HW/SW	HW	HW/SW
	cycle counts	cycle counts	cycle counts	speedup	speedup
AES-128	217,603	1,200	134,599	181.3	1.6
PRESENT-80	1,924,547	3,300	85,306	583.2	22.6

Note that the AES-128 hardware implementation is based on the core developed by Rudolf Usselman, available from OpenCores, and the software version is a 32-bit AES derived from the SSH open source package; the PRESENT-80 hardware design is based on the structure depicted in Fig. 1, and the software version was provided by one of the PRESENT authors. Both the hardware and software versions are basic implementations without specific optimization goals, and here the designs consider the encryption only with each plaintext assigned an initial key. 100 iterations for the AES and PRESENT algorithms (which transmit the plaintext and key to the cipher for each iteration) result in the above performance numbers.

Table 2. Toggle counts (TC) of standalone simulation

	TC/cycle	TC/encryption	TC/(encryption * byte)
AES-128	3,798	45,576	2,849
PRESENT-80	2,746	90,618	11,327

From Table 1, it is obvious that if we only consider the hardware acceleration the design speedup is often dramatic, but, if we consider the system integration and take into account the communication overhead, the resulting speedup can be much lower.

The GEZEL simulation environment also provides technology-independent toggle counting at Register Transfer Level (RTL), which is useful to roughly estimate the dynamic power consumption of a design.

The toggle counts collected in Table 2 only includes the AES and PRESENT encryption core when doing standalone simulation. This data can be utilized to early estimate the power- and energy-efficiency of the hardware designs of AES and PRESENT. The first column of the table indicates that PRESENT-80 is more power efficient than AES, in terms of dynamic power consumption. However, since most light-weight block ciphers, like PRESENT, are specialized cryptographic implementations for tight cost constraint applications, such as RFID tags, the energy-efficiency instead of power-efficiency should be emphasized because most of these applications are battery powered. The second column of the table reflects the toggle counts per cycle multiplying the cycle counts for each encryption, the results of which can be approximately equivalent to the energy consumption per encryption. Further, we divide the toggle counts per encryption by the number of plaintext bytes in one encryption. The obtained values can be assumed to be the energy required for the block cipher to encrypt one byte plaintext. This indicates that the PRESENT-80 might be less energy-efficient than the AES in standalone encryption mode. Table 3 illustrates the power values (by using post-place and route simulation model in XPower, which will be discussed later) obtained in standalone simulation on Xilinx Spartan-3 XC3S1000 FPGA, which well support our assumption based on toggle counts. Moreover, it reflects the relative accuracy of GEZEL toggle counting when predicting the actual power consumption of designs. Note that both the quiescent and dynamic

Table 3. Power results of standalone FPGA implementations (10 encryptions for each block cipher working at 20MHz)

	Quiescent Power(mW)	Dynamic Power(mW)	Time (ms)	Energy (mJ)	Energy/byte (μ J/byte)
AES-128	51.51	40.75	6	0.55	3.46
PRESENT-80	44.06	3.49	16.5	0.78	9.81

power values are collected from V_{ccint} , the FPGA core power supply voltage since we only consider the FPGA core power variation.

4 FPGA-Based Hardware/Software Co-design

Using the above GEZEL simulation environment we can translate the GEZEL description of the AES and PRESENT and control shells into synthesizable VHDL, which can be then added as coprocessors in the Xilinx Platform Studio (XPS) 9.1.02.

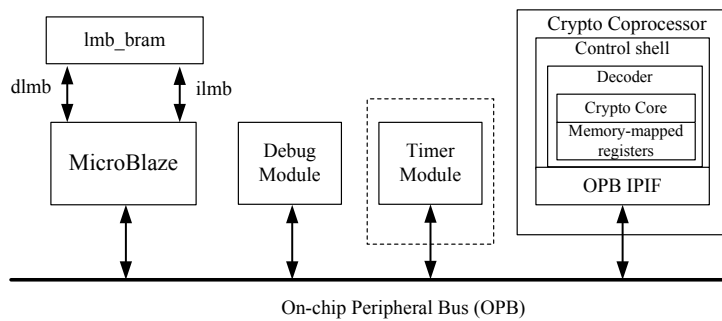


Fig. 5. FPGA-based SoC platform

The SoC system is built on a Xilinx Spartan-3E XC3S500EFG320 development board with both on- and off-chip memory. Since each on-chip memory read or write only takes 2 clock cycles compared to 22 and 23 clock cycles for off-chip memory read or write on our specific FPGA platform, we fully utilize the on-chip memory for our system design. Note that our research objective is trying to address some system integration issues or considerations for general SoC systems, and here the selection of MicroBlaze as the microprocessor and OPB as system bus is for detailed discussion.

A hardware timer module is added for measuring the speed of both crypto hardware coprocessor and the crypto software running on MicroBlaze. The timer will be removed when doing power estimation of the whole SoC system using XPower.

XPower is a commercial-off-the-shelf tool to estimate power consumption of Xilinx SRAM-based FPGAs. XPower utilizes either pre-routed or post-routed design data, and then makes a power model either for a unit or for the overall design. It considers resource usage, toggle rates, input/output power, and many other factors in estimation.

To get a good indication of the power consumed by the device using XPower, an accurate VCD file is needed. Here, we use a complete post-place-and-route, timing-accurate model built in XPS to generate the VCD file. Other system settings use the XPower default values.

Table 4. FPGA implementation areas (unit: slices)

	Crypto core	Coprocessor with wrapper
AES-128	1,877	2,097
PRESENT-80	271	460

The areas for AES and PRESENT coprocessors are presented in Table 4. The resources used for the other parts in both the systems are the same: 8Kb Block RAM, 8 slices for LMB wrapper, 99 slices for OPB wrapper, 749 slices for Microblaze and 65 slices for Debug module.

Table 5. FPGA system performance results (100 encryptions for each block cipher)

	SW	HW	HW/SW	HW	HW/SW
	cycle counts	cycle counts	cycle counts	speedup	speedup
AES-128	432,756	1,200	77,428	360.6	5.6
PRESENT-80	2,295,863	3,300	51,427	695.7	44.6

The numbers in Table 5 is for 100 iterations encryption for the AES and PRESENT, and each iteration transmits plaintext and key to the cipher. The performance improvement from using hardware/software co-design is satisfying respect to software using C codes which were compiled with -O2 optimization. However, combined with the former co-simulation results in GEZEL we see that the overhead is substantial. Take AES-128 FPGA codesign for example, 100 iterations in hardware only should take 1200 clock cycles, while we have used 77,428. This overhead factor (65X) is due to the communication with the processor and implementation of various command sequences with the encapsulated hardware. Note that the big differences in cycle counts and speedup values between GEZEL co-simulation and FPGA SoC implementation are due to the different processors (StrongARM vs. MicroBlaze) and compilers (arm-linux-gcc vs. mb-gcc).

In the former relative power estimation using toggle counts, we deduced that the more power-efficient PRESENT block cipher is in fact less energy-efficient

Table 6. FPGA system power and energy simulation results (4 encryptions for each block cipher woking at 50MHz)

	Quiescent	Dynamic	Time	Energy	Energy/byte
	Power(mW)	Power(mW)	(ms)	(mJ)	(μ J/byte)
AES-128	31.25	19.97	62.08	3.18	49.68
PRESENT-80	31.25	19.61	41.2	2.10	65.48

than AES, in terms of toggles per encryption per byte together with the standalone FPGA simulation results. When we look at this problem again in an FPGA-based SoC platform, we can find that the system with PRESENT coprocessor consumes slightly less total power than that with the AES coprocessor. However, still we can find that the PRESENT-based system is less energy-efficient than the AES-based system.

5 Conclusions

Due to the encryption speed and ease of implementation, block ciphers have been widely used in various embedded applications. Much research effort has been put on the trade-off designs on hardware implementation of block ciphers, but, we think that the hardware profile is unable to predict the performance and energy (or power) in the context of a real embedded system.

Using our design flow we can not only get some early prediction of performance and dynamic power consumption under co-simulation environment, which can help designers to refine the design at an early stage, but also get accurate performance and energy values after on-board FPGA implementation, which can help designers select the crypto coprocessor best fitted to some specific platforms. The illustrated SoC designs with AES and PRESENT coprocessors identify the hardware/software interfaces design as an important system integration issue, and address the power- and energy-efficiency evaluation issue at the system level. Our future work may focus on different hardware/software interfaces' (e.g. FSL) impact on the performance and energy- or power-efficiency of a typical SoC system with different kinds of crypto coprocessors, and possible optimization methods.

Acknowledgments. We thank Axel Poschmann for providing the C programs of PRESENT. This work was supported in part by NSF Grant No 0644070.

References

1. Chodowiec, P., Khuon, P., Gaj, K.: Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining. In: *FPGA 2001*, pp. 94–102. ACM, New York (2001)
2. McLoone, M., McCanny, J.: High Performance Single Chip FPGA Rijndael Algorithm Implementations. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) *CHES 2001*. LNCS, vol. 2162, pp. 65–76. Springer, Heidelberg (2001)
3. Good, T., Benaissa, M.: AES from the fastest to the smallest. In: Rao, J.R., Sunar, B. (eds.) *CHES 2005*. LNCS, vol. 3659, pp. 427–440. Springer, Heidelberg (2005)
4. Schaumont, P., Verbauwhede, I.: Hardware/Software Codesign for Stream Ciphers. In: *SASC (State of the Art of Stream Ciphers)*, Special workshop hosted by the ECRYPT Network of Excellence in Cryptology, Bochum, Germany (2007)
5. The GEZEL homepage, <http://rijndael.ece.vt.edu/gezel2>

6. Hachez, G., Koeune, F., Quisquater, J.-J.: cAESar results: Implementation of four AES candidates on two smart cards. In: 2nd AES Candidate Conference (AES2), Rome, Italy (1999)
7. Worley, J., Worley, B., Christian, T., Andworley, C.: AES Finalists on PA-RISC and IA-64: Implementations and Performance. In: 3rd AES Conference (AES3), NY, USA (2001)
8. Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., Uhsadel, L.: A Survey of Lightweight Cryptography Implementations. *IEEE Design and Test of Computers – Special Issue on Secure ICs for Secure Embedded Computing* 24, 522–533 (2007)
9. Bogdanov, A., et al.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhe, I. (eds.) *CHES 2007*. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
10. Schaumont, P., Ching, D., Verbauwhe, I.: An Interactive Codesign Environment for Domain-specific Coprocessors. *ACM Transactions on Design Automation of Electronic Systems* 11(1), 70–87 (2006)