Available online at www.sciencedirect.com
SCIENCE DIRECT
Microprocessors and Microsystems 29 (2005) 317–326
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

An AES crypto chip using a high-speed parallel pipelined architecture

S.-M. Yoo[a,*], D. Kotturi[b], D.W. Pan[a], J. Blizzard[b]

[a]*Electrical and Computer Engineering Department, The University of Alabama in Huntsville, 301 Sparkman Dr, Huntsville, AL, 35899 USA*
[b]*Cadence Design Systems, Inc., Plano, TX, USA*

Received 14 September 2004; revised 10 November 2004; accepted 16 December 2004
Available online 7 January 2005

Abstract

The number of Internet and wireless communications users has rapidly grown and that increases demand for security measures to protect user data transmitted over open channels. In December 2001, the National Institute of Standards and Technology (NIST) of the United States chose the Rijndael algorithm as the suitable Advanced Encryption Standard (AES) to replace the Data Encryption Standard (DES) algorithm. Since then, many hardware implementations have been proposed in literature. We present a hardware-efficient design increasing throughput for the AES algorithm using a high-speed parallel pipelined architecture. By using an efficient inter-round and intra-round pipeline design, our implementation achieves a high throughput of 29.77 Gbps in encryption whereas the highest throughput reported in literature is 21.54 Gbps.

Keywords: Encryption algorithm; Hardware implementation; Parallel pipelined design; Throughput

1. Introduction

The number of individuals and organizations using wide computer networks for personal and professional activities has recently increased a lot. A cryptographic algorithm is an essential part in network security. A well-known cryptographic algorithm is the Data Encryption Standard (DES) [13], which has been widely adopted in security products. However, serious considerations arise for long-term security because of the relatively short key word length of only 56 bits and from the highly successful cryptanalysis attacks.

In November 2001, the National Institute of Standards and Technology (NIST) of the United States chose the Rijndael algorithm as the suitable Advanced Encryption Standard (AES) [1] to replace the DES algorithm. Since then, many hardware implementations have been proposed in literature [2–12,15–22]. Some of them use field programmable gate arrays (FPGA) and some use application-specific integrated circuits (ASIC). The advantages of a software implementation include ease of use, ease of upgrade, portability, and flexibility. However, a software implementation offers only limited physical security, especially with respect to key storage [13]. Conversely, cryptographic algorithms (and their associated keys) implemented in hardware are, by nature, more physically secure, as they cannot easily be read or modified by an outside attacker. The downside of traditional (ASIC) hardware implementations is the lack of flexibility with respect to algorithm and parameter switching. Reconfigurable hardware devices such as FPGAs are a promising alternative for the implementation of block ciphers. FPGAs are hardware devices whose function is not fixed and can be programmed in-system.

In this paper, we present an implementation of the AES block cipher with Virtex II Pro FPGA using 0.13 μm and 90 nm process technology [14]. We have exploited the temporal parallelism available in the AES algorithm. Our chip contains the same ten units, and each unit can execute one round of the algorithm. Using external pipelined design, ten rounds of the algorithm are executed in parallel in a chip. Furthermore, using internal pipelining and key exchange pipelining, our implementation operating at 233 MHz achieves a throughput of 29.77 Gbps in encryption which

* Corresponding author. Tel.: +1 256 824 6858; fax: +1 256 824 6803.
E-mail addresses: yoos@ece.uah.edu (S.-M. Yoo), dkotturi@cadence.com (D. Kotturi), dwpan@ece.uah.edu (D.W. Pan), blizzard@cadence.com (J. Blizzard).

0141-9331/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2004.12.001

is much higher than the best (in terms of throughput) implementation reported in literature.

The rest of the paper is organized as follows. Section 2 describes briefly the AES cryptographic algorithm. Section 3 explains the details of our design on the AES cryptographic chip. Section 4 compares the performance of our implementation to earlier ones. Finally, Section 5 concludes the paper.

2. The AES algorithm and previous work

2.1. The AES algorithm

The AES algorithm is a symmetric block cipher that processes data blocks of 128 bits using a cipher key of length 128, 192, or 256 bits. Each data block consists of a 4×4 array of bytes called the *state*, on which the basic operations of the AES algorithm are performed. Fig. 1 shows the AES encryption and decryption procedures.

The encryption procedure is as follows. After an initial round key addition, a round function consisting of four different transformations—byte-sub, shift-row, mix-column, and add-round-key—is applied to the data block in the encryption procedure. The round function is performed iteratively 10, 12, or 14 times, depending on the key length. The mix-column operation is not applied to the last round. The byte-sub operation is a nonlinear byte substitution that operates independently on each byte of the state using a substitution table (S-Box). The shift-row operation is a circular shifting on the rows of the state with different numbers of bytes (offsets). The mix-column operation mixes the bytes in each column by the multiplication of the state with a fixed polynomial modulo x^4+1. Add-round-key operation is an XOR that adds a round key to the state in each iteration, where the round keys are generated during the key expansion phase.

The byte-sub transformation (S-Box operation), which consists of a multiplicative inverse over $GF(2^8)$ and an affine transform, is the most critical part of the AES algorithm in terms of computational complexity. However, the S-Box operation is required for both encryption and key expansion. Conventionally, the coefficients of the S-Box and inverse S-Box are stored in the lookup tables, or a hard-wired multiplicative inverter over $GF(2^8)$ can be used, together with an affine transformation circuit.

The decryption procedure of the AES is basically the inverse of each transformation. However, the standard decryption procedure is not identical to the encryption procedure. That is, the sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is

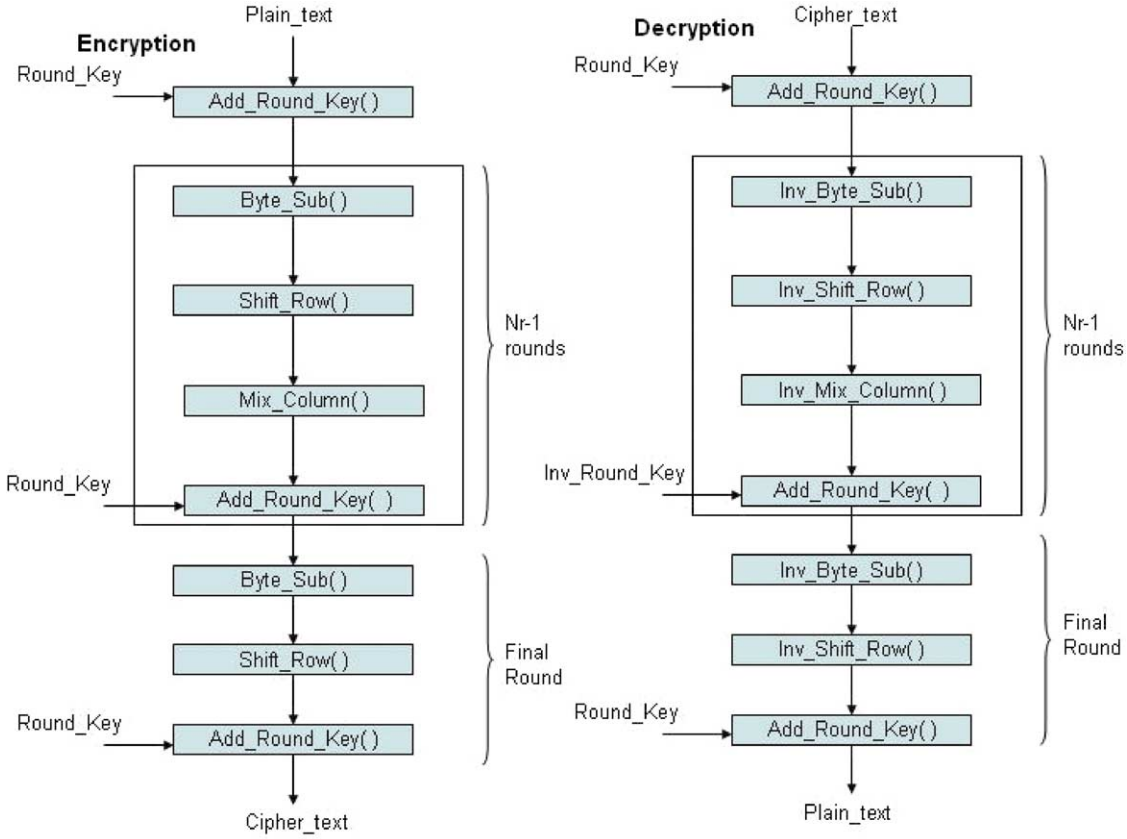

Fig. 1. The AES algorithm (equivalent version). (Nr: 10, 12, or 14 depending on key length).

the same. There is, however, an equivalent version of the decryption procedure that has the same structure as the encryption procedure. The equivalent version has the same sequence of transformations as the encryption procedure (with transformations replaced by their inverses). To achieve this equivalence, a change of key schedule is needed. In addition, two separate changes are needed to bring the decryption structure. The standard decryption round has the structure inv-shift-row, inv-byte-sub, add-round-key, and inv-mix-column. Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged. The equivalent version of the decryption procedure is shown in Fig. 1.

2.2. Previous work

There exist many presentations of hardware implementations of Rijndael AES algorithms in literature. Some of them will be briefly introduced here considering throughput. In 2001, Elbirt et al. [9] compared five candidate algorithms (including Rijndael algorithm) for AES block cipher using FPGA implementations. Here, the throughputs of Rijndael algorithm were in 187.8 Mbps ~1.94 Gbps. In 2003, many implementations are shown in literature. Verbauwhede et al. [10] presented an ASIC implementation under the throughput of 2.29 Gbps. Su et al. [2] reduced hardware overhead of the S-Box by 64% and the throughput of their pipelined implementation using ASIC was 2.38 Gbps. McLoone and McCanny [3] utilized look-up tables to implement the entire Rijndael round function under the throughput of 12 Gbps using FPGAs. In 2004, Hodjat and Verbauwhede's [8] FPGA implementation showed a high throughput of 21.54 Gbps using a fully pipelined approach with inner-round pipelining and outer-round pipelining. A brief introduction on earlier works is well written in [2]. Section 4 lists four tables comparing the performance of the presentations in literature.

3. The AES implementation using a fully pipelined design

3.1. Encryption data path—pipeline design

The goal of this implementation is to achieve the highest possible throughput. We have used the bottom-up design approach, implementing the elementary operations first before designing the final data path.

A block based top-level implementation of the design for encryption is shown in Fig. 2. Round_1 through Round_10 represent the individual rounds in the AES-128 encryption. The pipelining between each of the rounds will achieve a high performance encryption implementation. Although implementing an iterative pipelining based approach is one option, for clarity and simplicity, we have used a fully expanded implementation for all ten rounds. The data generated in each individual round is successively utilized as the input in the next round. Block level view of a single round implementation with internal pipelining is shown in Fig. 3. Exploiting the loop level parallelism which the AES offers, we determined that a simple ten stage pipelining of the top level with pipelined lower level

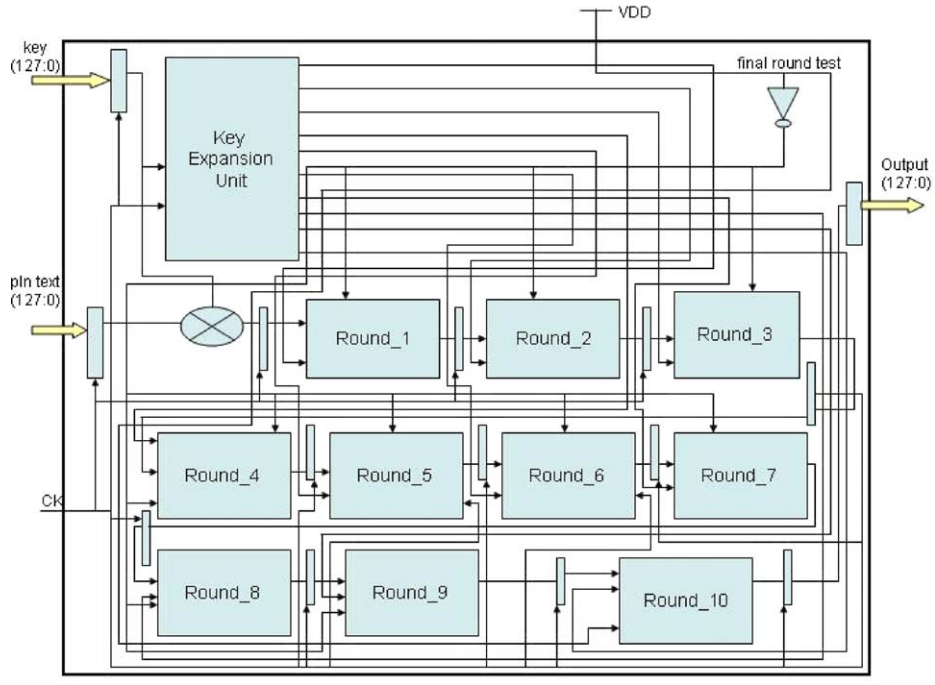

Fig. 2. A pipelined Rijndael and AES-128 encryption implementation.

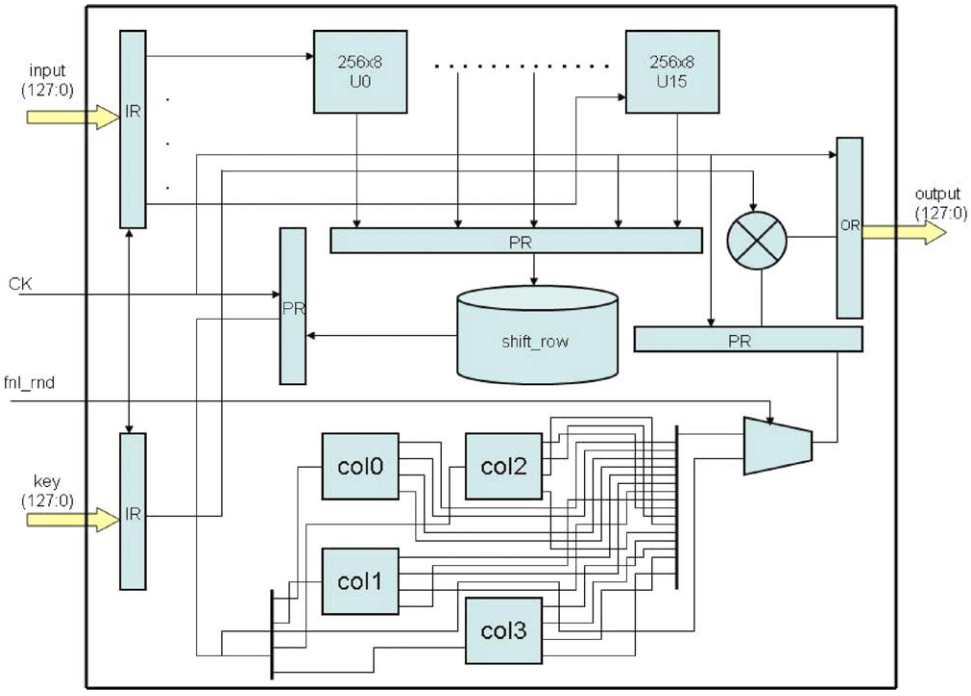

Fig. 3. A single round data block implementation.

blocks of the hierarchy is all that is needed to achieve high throughput. This is one of the easiest methods where high performance can be achieved in a very minimal amount of time thus reducing the overall design implementation cycle.

At the top-level, our pipeline design (shown in Fig. 2) is similar to [8]. There is a pipeline stage between each round, i.e. the design is fully pipelined. However, our internal (inside each round) pipeline design (shown in Fig. 3) is different from [8]. In each round, our design has three pipeline stages, one immediately after byte-sub operation, one just after shift-row operation, and the last just before data output. In each round, the design in [8] has four or seven pipeline stages, one after a byte-sub operation and three or six in a byte-sub operation.

In addition, our design has one pipeline stage (before XOR operation between Round Key Block) in key generation blocks as shown in Figs. 4 and 5. Internally, the key expansion block renders itself as a pipelined implementation between each of the key creations from Key1 through Key10. This is automatically realized by the parallel nature of the design. These additional pipelines make it possible for our implementation to obtain a higher throughput than [8].

3.2. Round key generation

For our implementation, we have chosen to use a hierarchical simultaneous key generation methodology. This is similar in approach to the fly key generation method. However, there is internal sub-pipelining for each of the sub-stages of the key creation shown in Fig. 4. This would create the key for a single round. The XST FPGA synthesis environment will be able to automatically recognize constant logic and assign constant logic ‘0’ or ‘1’ to the appropriate ‘Tie’ cells from the library. The output is the key for the next round.

Based on the AES literature [1], we have implemented the round key generation as a simple state table substitution of the 32 bits of the input key and thereby implementing XOR operations for producing the expanded 128 bits round key. Using internal pipelining would greatly reduce the minimum clock period needed to assure the correct functionality of round key generation. Hence, implementing an extremely fast block in this preliminary stage of the design itself is very much possible. With this implementation, we have achieved a maximum post synthesis clock frequency of 478.9 MHz for key generation.

Refer to Fig. 4. Here, placing the pipeline registers and loading the data using input/output registers (IR/OR) is the key to achieving high performance. IR is an input register used to load the input data. PR is the pipeline register used with intermediate data processing and OR is the output register. It can be inferred that it takes three clock cycles for the output to appear from key_in to key_out. What this would mean is that, for each of the ten rounds, to produce the key in each successive round, would require at least three clock cyles.

The Round Keys Block simply uses the Round Key Block described previously to create the round keys for all

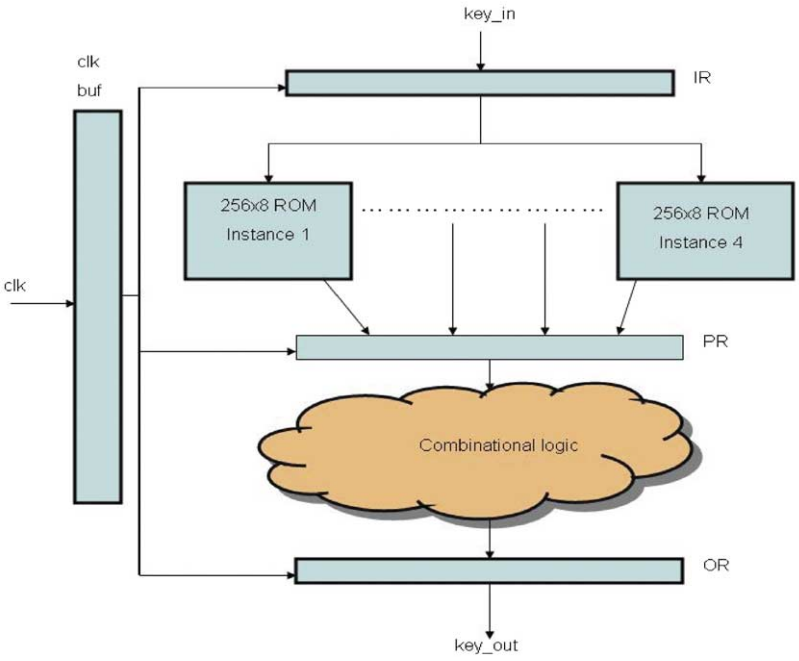

Fig. 4. A pipelined hardware architecture for key expansion.

the individual rounds. Hence, for 128 bits data/key encryption, it uses ten instances of the Round Key Block mentioned above to create all the ten round keys. Inside this block, actually each of the keys from Key1 to Key10 is created using the key expansion algorithm as shown in Fig. 5.

In this implementation with the active edge of the clock and the user key, each round key is created by instantiating the Round Key Block of Fig. 4 ten times. Any decent parallel hardware architecture offers naturally high performance. Exploiting this concept, we have used internal pipelining within each of the round key creation stages. We represent a simple view of this implementation in Fig. 5. The use of balanced internal pipelining in between stages of a parallel architecture helps in reducing the flip-flop to flip-flop clock delay. As a result, it maximizes the performance of a design while guaranteeing minimum clock speed.

3.3. S-Box design

The S-Box is an invertible function that performs two transformations: a multiplicative inverse in $GF(2^8)$ with polynomial $m(x) = (x^8 + x^4 + x^3 + x + 1)$ and an affine transformation (over $GF(2)$): $b(x) = (x^7 + x^6 + x^2 + x) + a(x)(x^7 + x^6 + x^5 + x^4 + 1) \bmod (x^8 + 1)$, where $a(x)$ is the multiplicative inverse in polynomial form.

There are two approaches in literature to the S-Box implementation, calculating the S-box values on the fly or storing the S-Box values in ROMs. In [2,8] the S-Box values are calculated on the fly using the two transformations mentioned above. This approach aims at reducing hardware

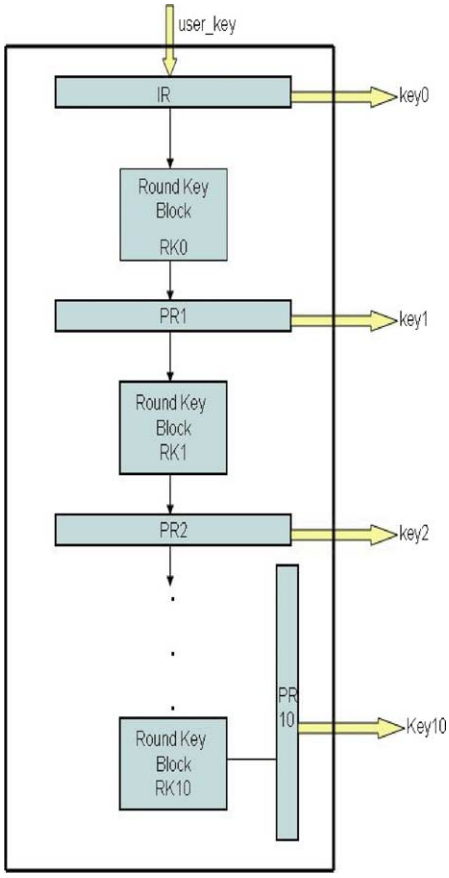

Fig. 5. Round keys block with internal pipelining.

complexity, but the approach increases the critical path delay of the encryption. In [3,10] the S-Box values are stored in ROMs. This approach aims to decrease the critical path delay of the encryption. From the original Rijndael specification, the S-box values are constant. They do not change during the encryption process. Thus, the values can be stored instead of being an on the fly calculation. We chose to store the values in a 256×8 ROM rather than calculating the S-Box values using the GF transformations.

We have used ROM Macro for the S-Box implementation. The S-Box in Rijndael contains 256 different values for all the 256 S-Box inputs. As suggested in Rijndael, this can be implemented in a look-up-table (LUT) format where for every corresponding value of input to the S-Box there is a corresponding S-Box output.

3.4. Some results and notes on encryption

The results in encryption after post placement and routing optimization are shown in Table 1. With full constraint setting and successful static timing analysis, our design has the hardware and timing characteristics as shown in Table 1.

We have used ROM rich FPGA xc2vp70 [14] for our implementation. We have selected flattening the hierarchy during post-route optimization. This is just a matter of methodology style. If transitioning to ASIC at a later stage is not the desired goal, it may not be necessary to flatten the hierarchy. In an ASIC, a flattened design is more desirable for several reasons associated with RTL-to-GDS-to-Final Sign-off design stream. Greater benefits of a flattened design may be found on any of the major CAD vendor's product manuals and user guides or for example reference flows from TSMC or Artisan. Some hardware density utilization numbers are shown in Table 2 when this architecture is implemented in an xc2vp70 device.

3.5. Decryption results

In an FPGA based flow it is usually not necessary to specify floor planning or placement constraints unless tight budgeting is necessary. We have let Xilinx Synthesis Tool (XST) [14] handle these two stages before achieving a cleanly routed design. However, for routing we have specified area constraints so that the design placement and routing is performed with minimal lengths for routes. By doing so, we can restrict XST automatically routing design components using long wires. This way we can reduce the delay contributed by the routing interconnect. In our experience with XST, we observed that XST performs a commendable job even if we simply specify a global area constraint. More information on this constraint can be found in Xilinx Constraints guide [14]. Table 3 shows the results in decryption only after post synthesis, place and route optimization and static timing analysis.

Fig. 6 provides the block level view of the design in decryption to show the top-level pipelining between each round. It should be noted that there is internal pipelining with in each of the round blocks ICRT1, ICRT2 etc.

Table 1
Hardware and timing statistics of encryption block

I/O pins	385
Block RAMs	200
Slices	7761
LUT	4884
Critical path delay	2.108 ns
Frequency	232.6 MHz
Throughput	29.8 Gbps
Registers	1351
Multiplexers	10
Min input required time	2.787 ns
Min output required time	4.363 ns

Table 2
Device utilization summary

Number of slices	5408
Number of slice flip flops	9408
Number of 4 input LUTs	4884
Number of bonded IOBs	384
Number of BRAMs	200
Number of GCLKs	1
Flatten	True

3.6. Integrated chip

We implemented an FPGA design that efficiently performs both encryption and decryption of 128-bit data and 128-bit key encryption. Fig. 7 shows the top level schematic of the integrated chip. It can function as both an encryption unit or as a decryption unit depending up on the control signals ENC and DEC. When ENC is selected, it signals the data path unit that encryption needs to be performed. Similarly, if DEC is selected, decryption will be performed. The signal RST is a global RST signal. When RST is high, the integrated chip is reset. data_in and key_in are 128-bit inputs and data_out is a 128 bits output after encryption or decryption. Post place and route static timing in the integrated chip has the hardware and timing numbers shown in Table 4.

Table 3
Post synthesis, P and R hardware and timing statistics

Number of I/Os	386
Number of 256×8-bit ROM macros	200
Number of LUTs	8283
Number of slices	6541
2-to-1 multiplexers	1
Number of clock buffers	1
Clock	4.5
Min input required time before clock	2.787 ns
Min output required time after clock	4.381 ns
Frequency	222.22 MHz

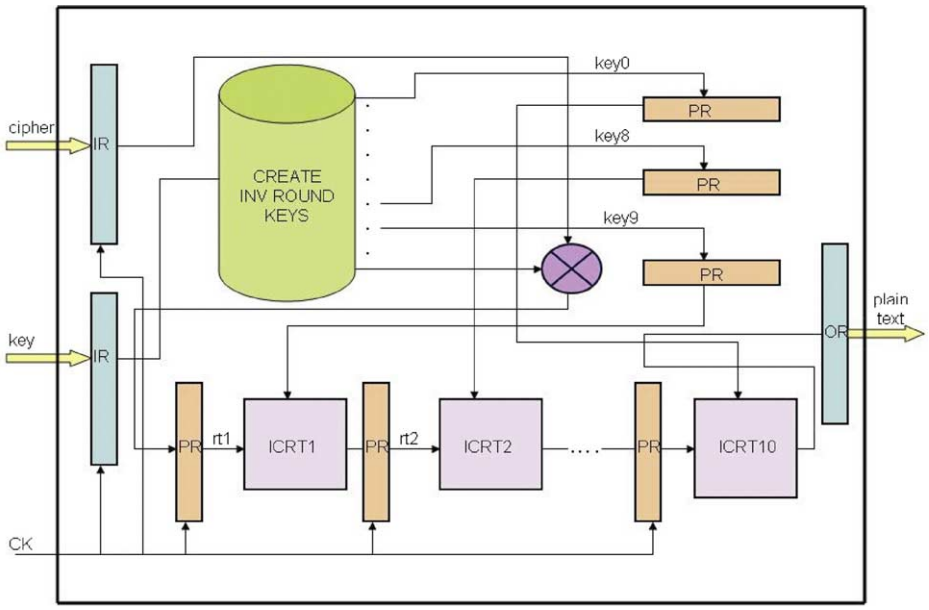

Fig. 6. Decryption implementation with pipelining between round data blocks.

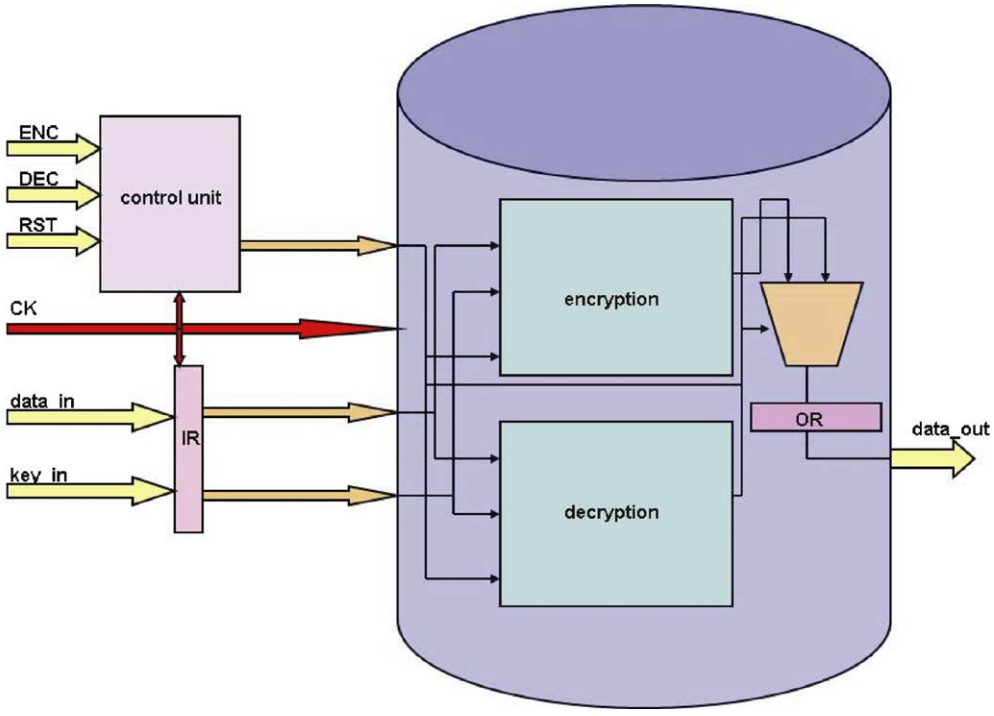

Fig. 7. Integrated chip for encryption and decryption.

After hardware usage optimization, area optimization and timing optimization, we implemented a functionally verified integrated chip that can perform encryption and decryption at a maximum clock frequency of 125.63 MHz achieving a throughput of 16.08 Gbps. Due to the integration complexities, the throughput of the integrated chip has dropped to almost half of what has been achieved with encryption or decryption as an individual unit.

Table 4
Hardware and timing statistics for the integrated chip

Number of external IOBs plus CK	387
Number of BRAMs	400
Number of slices	11,433
Number of LUTs	14,124
Minimum input required time	5.58 ns
Minimum output required time	5.26 ns
Clock	7.96 ns

Table 5
Performance comparisons in encryption only

Designers	Technology	Throughput (Gbps)	Gates or slices	Clock period or frequency	Power (mw)
[15]	0.18 μ, 1.8 V CMOS tech. ASIC	1.6	173 K gates	8 ns	56
[16]	Xilinx Virtex XCV1000	3.65	17,314	100 MHz	n/a
[3]	Xilinx Virtex family FPGA XCV812E-8	12	244 RAM	93.9 MHz	n/a
[8]	Xilinx Virtex family XC2VP20-7	21.54	84	168.3 MHz	n/a
Ours	Xilinx Virtex family FPGA XCV812E-8	18.8	200	142.8 MHz	1029
Ours	Xilinx Virtex family XC2VP70-7	29.8	200	232.6 MHz	n/a

4. Performance comparison

4.1. Evaluation metrics

When evaluating a given implementation, the throughput of the implementation and the hardware resources required to achieve this throughput are usually considered the most critical parameters. No established metric exists to measure the hardware resource costs associated with the measured throughput of an FPGA implementation. Two area measurements are readily apparent—logic gates and configurable logic blocks (CLBs) slices. It is important to note that the logic gate count does not yield a true measure of how much of the FPGA is actually being used. Hardware resources within CLB slices may not be fully utilized by the place-and-route software so as to relieve routing congestion. This results in an increase in the number of CLB slices without a corresponding increase in logic gates. To achieve a more accurate measure of chip utilization, CLB slice count was chosen as the most reliable area measurement. Therefore, to measure the hardware resource cost associated with an implementation's resultant throughput, the throughput per slice metric is used. We defined it as

$$\text{throughput per slice} = (\text{encryption rate})/(\#\,\text{CLB slices used}).$$

4.2. Device used

We used the device XC2VP70 with speed grade −7 with Virtex II Pro FPGA while [8] used the device XC2VP20 with speed grade −7 with Virtex II Pro FPGA. XC2VP70 has more ROMs than XC2VP20. Both devices are manufactured using 0.13 μm, nine-layer copper interconnection with a transistor process technology of 90 nm. Therefore, except for the differences in terms of the hardware density including the number of ROMs, these two devices perform under the same operating and process conditions.

4.3. Throughput

Tables 5–7 compare our implementation with several others very recently reported in the literature in terms of encryption only, decryption only, both encryption and decryption, respectively. We have not included some of them since their throughputs are not high because of different design methods (they are compared in [2] and [8]). It is shown that the throughput of our implementation is very high compared to others.

Table 8 compares our implementation with [8] which reported the highest throughput so far in literature. Here, we will describe the performance difference compared to [8] whose throughput is the highest reported in literature and the device they used is the same as ours. Our throughput (29.77 Gbps) is 38% higher than that of [8] (21.54 Gbps). Our throughput per slice (5.5 Mbps) is 31% higher than that of [8] (4.2 Gbps). We used fewer lookup tables (LUTs) but more number of BRAMs. Our critical path is faster than [8] but our latency is slower than [8].

Table 6
Comparison of implementations in decryption only

Designer	Technology	Clock/frequency (MHz)	Throughput	Gates or slices
[2]	0.35 μm	200	2.008 Gbps	58.43 K gates
[17]	Virtex XV2V1000	75	0.739 Gbps	4325 slices
[18]	Virtex XCV1000E	38.8	451.5 Mbps	2580
[19]	Virtex XCV 2000E	34.2	4.121 Gbps	5677
[16]	Virtex XCV 1000	28.5	3.6 Gbps	17,314
Ours	Virtex-II Pro	222.22	28.44 Gbps	6541

Table 7
Comparison of results in both encryption and decryption

Authors	Implementation	Architecture	Frequency	Throughput	Power
[17]	Encryption and decryption	Nonpipelined in Xilinx XC2V1000 37 ROMs	75 MHz for encryption and decryption	739 Mbps	N/A
[20]	Encryption and decryption	Nonpipelined 0.35 Standard cell library 135807 transistors	66 MHz for encryption, 55 MHz for decryption	844 Mbps for encryption, 704 Mbps for decryption	N/A
[16]	Encryption and decryption	Nonpipelined Xilinx XCV300BG432 2358 CLB Slices	22 MHz for both encryption and decryption.	259 Mbps	N/A
[16]	Encryption and decryption	Fully pipelined Xilinx XCV1000BG560	28.5 MHz	3.650 Gbps	N/A
[15]	Encryption only	Nonpipelined 0.18 CMOS Standard cell Library	125 MHz for encryption only	1.6 Gbps for encryption only	54 mw
[21]	Encryption and decryption	Nonpipelined Altera ACEX 1K50	19.6 MHz for encryption and decryption	61.2 Mbps	N/A
[11]	Encryption and decryption	Nonpipelined Xilinx XCV1000E	38.8 MHz for encryption and decryption	451.5 Mbps	N/A
[22]	Encryption and decryption	Partially pipelined cyclone	76.92 MHz for encryption and decryption	197 Mbps	N/A
Ours	Encryption, decryption and integrated chip	All three units are fully pipelined implementations Virtex Pro	232.6 MHz for encryption, 222. 22 MHz for decryption, 125.3 MHz for integrated chip	29.8 Gbps for encryption, 28. 44 Gbps for decryption, 16. 08 Gbps for integrated chip	2.083 w for integrated chip

4.4. Power consumption

The power consumption statistics are shown in Tables 5 and 7. Power consumption is analyzed using simple probabilistic approach. Our power consumption seems to be a bit more on the high side. Also, the ROM instances in the design tend to consume more power at a given operating voltage. However, since our objective here is optimization of the design for speed rather than low power optimization of the overall design and also since the scope of employing a better power optimization techniques is limited in an FPGA, we did not worry too much about the large amount of power consumed by this design.

Table 8
Performance comparisons between [8] and ours

	[8]	Ours
Device used	XC2VP20-7	XC2VP70-7
Number of slices	5177	5408
Number of LUTs	8285	4884
Number of BRAM	84	200
Critical path	5.94 ns	4.23 ns
Frequency (MHz)	168.3	232.6
Latency (cycles)	41	60
Throughput (Gbps)	21.54	29.77
Throughput/slice (Mbps)	4.2	5.5

5. Conclusion

In this paper we presented a hardware implementation increasing throughput for AES encryption algorithm. By using an efficient inter-round and intra-round pipeline design, our implementation achieves a high throughput of 29.77 Gbps in encryption whereas the highest throughput reported in literature is 21.54 Gbps. Therefore, we achieved a throughput much higher than any other implementations reported in the literature.

References

[1] National Institute of Standards and Technology (US), Advanced Encryption Standard, http://csrc.nist.gov/publication/drafts/dfips-AES.pdf.
[2] C.P. Su, T.F. Lin, C.T. Huang, C.W. Wu, A high-throughput low-cost AES processor, IEEE Commun. Mag. 42 (12) (2003) 86–91.
[3] M. McLoone, J.V. McCanny, Rijndael FPGA implementations utilizing look-up tables, J. VLSI Signal Process. Syst. 34 (3) (2003) 261–275.
[4] K. Gaj, P. Chodowiec, Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays, in: CT-RSA 2001, LNCS 2020, pp. 84–99.
[5] F.X. Standaert, G. Rouvroy, J.J. Quisquater, J.D. Legat, Efficient implementation of Rijndael encryption in reconfigurable hardware: improvements and design tradeoffs, in: CHES 2003, LNCS 2779, pp. 334–350.

[6] G.P. Saggese, A. Mazzeo, N. Mazzocca, A.G.M. Strollo, An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm, in: FPL 2003, LNCS 2778, pp. 292–302.
[7] K. Jarvinen, M. Tommiska, J. Skytta, A fully pipelined memoryless 17.8 Gbps AES-128 encryptor, in: International Symposium on Field Programmable Gate Arrays, 2003, pp. 207–215.
[8] A. Hodjat, I. Verbauwhede, A 21.54 Gbits/s fully pipelined AES processor on FPGA, in: IEEE Symposium on Field-Programmable Custom Computing Machines, 2004.
[9] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists, IEEE Trans. VLSI Syst. 9 (4) (2001) 545–557.
[10] I. Verbauwhede, P. Schaumont, H. Kuo, Design and performance testing of a 2.29-Gb/s Rijndael processor, IEEE J. Solid-State Circuits 38 (3) (2003) 569–572.
[11] S. Mangard, M. Aigner, S. Dominikus, A highly regular and scalable AES hardware architecture, IEEE Trans. Comp. 52 (4) (2003) 483–491.
[12] A. Satoh, S. Morioka, Unified hardware architecture for 128-bit block ciphers AES and Camellia, in: Proc. Cryptographic Hardware and Embedded Sys. (CHES), 2003.
[13] B. Schneier, Applied Cryptography, Wiley, New York, 1996.
[14] Virtex-11™ Platform FPGAs: Introduction and overview, http://www.xilinx.com/bvdocs/publications/ds031-1.pdf, 2004 (accessed on March 19, 2004).
[15] P.R. Schaumont, H. Kuo, I.M. Verbauwhede, Unlocking the design secrets of a 2.29 Gb/s Rijndael processor, in: Design Automation Conference 2002. Proceedings, 39th June 2002, pp. 634–639.
[16] N. Sklavos, O. Koufopavlou, Architectures and VLSI implementations of the AES-proposal Rijndael, IEEE Trans. Comput. 51 (12) (2002) 1454–1459.
[17] C. Chitu, D. Chien, C. Chien, I. Verbauwhede, F. Chang, A hardware implementation in FPGA of the Rijndael algorithm, in: Circuits and Systems 2002 (MWSCAS-2002) 45th Midwest Symposium, August 2002, pp. I-507–510.
[18] J.H. Shim, D.W. Kim, Y.K. Kang, T.W. Kwon, J.R. Choi, A Rijndael cryptoprocessor using shared on-the-fly key scheduler, in: ASIC 2002, Proceedings IEEE Asia-Pacific Conference, August 2002, pp. 89–92.
[19] N.A. Saqib, F. Rodriguez-Henriquez, A. Diaz-Perez, AES algorithm implementation—an efficient approach for sequential and pipeline architectures, in: Computer Science 2003, Proceedings of the Fourth Mexican International Conference, September 2003, pp. 126–130.
[20] L. Deng, H. Chen, A new VLSI implementation of the AES algorithm, in: Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference, June 2002, pp. 1500–1504.
[21] A.C. Zigiotto, R. d'Amore, A low-cost FPGA implementation of the Advanced Encryption Standard algorithm, in: Integrated Circuits and Systems Design 2002 Proceedings. 15th Symposium, September 2002, pp. 191–196.
[22] A. Panato, M. Barcelos, R. Reis, A low device occupation IP to implement Rijndael algorithm (cryptography), in: Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 20–25 (suppl.).

Seong-Moo Yoo received the MS and PhD degree in computer science from the University of Texas at Arlington in 1989 and 1995, respectively. Since September 2001, he is an associate professor in Electrical and Computer Engineering Department of the University of Alabama in Huntsville, Huntsville, Alabama, USA. From September 1996 to August 2001, he was an assistant professor in Computer Science Department of Columbus State University in Columbus, Georgia, USA. Dr Yoo is the conference chair of ACM Southeast Conference 2004, April, 2004, Huntsville, Alabama, USA. He was the co-program chair of ISCA 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2003), August 2003, Reno, Nevada, USA. Dr Yoo's research interests include wireless networks, parallel computer architecture, and computer network security. Dr Yoo is a senior member of IEEE and a member of ACM. He may be contacted at yoos@eng.uah.edu.

Deen Kotturi received the MS degree in Electrical Engineering from the University of Alabama in Huntsville in 2004. He is working as a Lead Engineer at Cadence Design Systems, Inc since 2001 till to date. His primary interests are on Interconnect Modeling in Nanometer VLSI Designing, RC reduction algorithms for efficient Extraction for Timing and Signal Integrity Analysis, Transistor/Cell level simulation based design analysis for manufacturability and VLSI architectures for high speed digital circuits. He may be contacted at dkotturi@cadence.com.

Wendi (David) Pan is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Alabama in Huntsville. He received his PhD degree in Electrical Engineering from the University of Southern California in 2002. His research interests include computer networks, image and video coding and communications, and signal processing architectures. He may be contacted at dwpan@eng.uah.edu.

John Blizzard is working as an Application Engineering Manager at Cadence Design Systems, Inc since 1996. He has focused in IC implementation and test vector analysis with a variety of semiconductor companies. Current activities are in promoting next generation EDA capabilities for synthesis and IC implementation. He can be contacted at blizzard@cadence.com.