Implementation and Comparative Analysis of AES as a Stream Cipher

Bin ZHOU, Yingning Peng
Dept. of Electronic Engineering,
Tsinghua University,
Beijing, China, 100084
e-mail: zhoubin06@mails.tsinghua.edu.cn

Kris Gaj
Dept. of Electrical and Computer Engineering
George Mason University
Fairfax, Virginia, 22030, USA
e-mail: kgaj@gmu.edu

Zhonghai ZHOU
Ocean University of China College of Marine Geo-science
Sea instrument measuring appliance research institute
of Shandong province academy of science
Qingdao, China, 266001
Email: zhouzhonghai2001@163.com

***Abstract*— Advanced Encryption Standard (AES) is the current encryption standard adopted by U.S. government and plays an important role in cryptograph systems. In this paper, AES was transferred into a stream and variant compact architectures are studied. On-the-fly key scheduling schema is also used. Pure logic based and distributed RAM based S-Boxes are both implemented for the purpose of best speed and area. Pipelined architecture is also studied to achieve a better throughput. Different memory schemas are navigated, including 2-bank distributed RAM, 2-bank block RAM, shift-register in LUT, 1-bank registers and dual-port memory. 8-bit, 32-bit, 64-bit Datapath versions are implemented to get the best throughput/area ratio. The whole design is targeted to Xilinx Spartan 3 FPGAs. The 32-bit architecture had a maximum clock frequency of 50.0 MHz and used 341 slices on the Spartan-3, a throughput of 118.5 Mbps. The pipelined 32-bit architecture had a maximum clock frequency of 125.1 MHz and used 422 slices on the Spartan-3, a throughput of 296.49 Mbps. The results show that our implementation has a good potential to fit to stream cipher requirements.**

Index Terms: AES, Stream Cipher, Compact AES, Pipeline

I. INTRODUCTION *(HEADING 1)*

AES is the current encryption standard by the U.S. government. It is used world-wide and intensively studied. It was announced by the National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) [3]. It shows many good characteristics, such as strong security level, fast computing and flexible architecture, which makes it the most popular algorithm used in symmetric key cryptography. Usually, AES is considered as a block cipher, which operates on a fined, larger number of bits to provide a bigger throughput and higher security. Meanwhile, stream cipher is another kind of symmetric key cipher, where cipher text is a function of the current and all preceding blocks of plaintext. Stream cipher is usually used in limited resource environment, such as cell phones, network stream media, wireless network and mobile devices and so on. It has a relatively smaller key size and could be implemented in a more efficient way. As a well-known alternative, by feeding back its key stream, block cipher could be adopted as a stream cipher. So in this paper, we use Counter Mode (CTR) AES to make it as a stream cipher.

As a main contribution to stream ciphers, in 2004, the ECRYPT stream cipher project (eSTREAM) [3] was launched to introduce new stream ciphers and introduce a wide range of systems. The candidates are: DECIM, Edon80, F-FCSR, Grain, MICHKEY, Moustique, Pomaranch and Trivium. In [2], they were compared with each other in terms of performance, throughput and area. And there is some work done on the AES as a stream cipher. In [1], the AES was implemented on a small FPGA using an application specific instruction processor; in [2] and [5], a compact architecture is introduced, using the data path widths equal to 64-bit, 32-bit, and 8-bit.

II. AES STREAM CIPHER ARCHITECTURE

A. Standard Iterative AES Architecture

AES is based on round operations and key scheduling functionality. Standard AES has a 128-bit block size and three variable key size (128-bit, 192-bit and 256 bit) with the round repeated 10, 12, 14 times, respectively. In this paper, we focus on 128-bit key size. The round contains four operations: ShiftRows, SubByte, MixColumn and AddRoundkey. Fig. 1 shows the basic diagram of AES iterative operation.

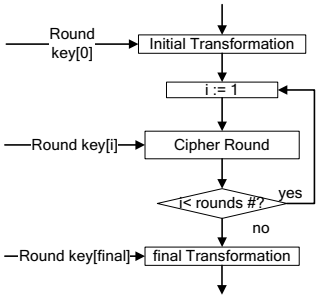

Figure 1. Diagram of basic iterative AES [4]

The cipher round of encryption is shown in Figure 2. It operates on a "state" block of 128 bits, which is usually

978-1-4244-4520-2/09/$25.00 ©2009 IEEE

organized as a 16x8 array. The minimum datapath we could achieve is 8-bit.

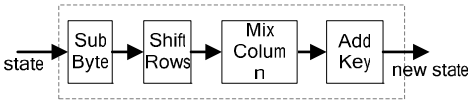

Figure 2. Encryption round

B. Mode Selection

The AES Counter (CRT) mode, Output Feedback (OFB) mode and Cipher Feedback (CFB) mode are suitable for stream cipher. However, a stream cipher has restricted resources. Pipeline should be added to gain speedup. Intermediate stages could be added to CRT, OFB and CFB modes. As shown in Fig. 3, it is easy to add pipeline stages to CRT mode. The next stage $IS_{i+1} = IS_i + 1$ is predictable and the pipeline could run without waiting for the intermediate states.

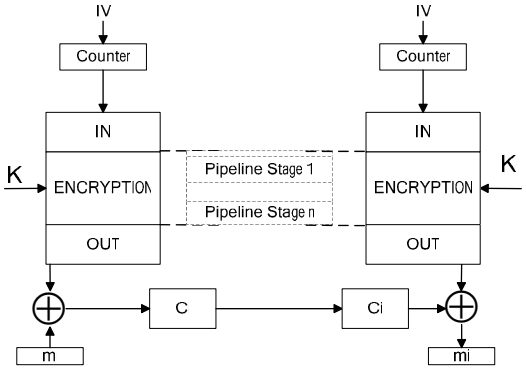

Figure 3. Adding Pipeline stages to CRT mode

However, things are different for OFB and CFB modes when adding pipeline stages. The next state value IS_{i+1} will depend on the last stage value IS_i, which will slowdown the whole process rather than speed it up. So we choose CRT mode as our implementation subject.

C. On-the-fly Key Scheduling

The on-the-fly key scheduling will generate the key with the round operation goes, which will save resources and provide more flexibilities.

32-bit on-the-fly key scheduling [4]

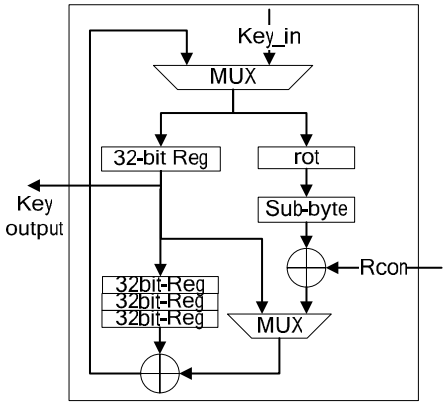

Figure 4. 32-bit on-the-fly key scheduling

Fig. 4 shows how the 32-bit key scheduling works. First, the initial keys are fed to a register and a shift register. Then when the fourth key word comes, it goes through the Rotation and SubByte operation, XOR with the first key word, generates the fifth key. After that, the following 3 key words are generated by XOR the other key words.

III. COMPACT AES ARCHITECTURE

A. 128-bit datapath full mode[4]

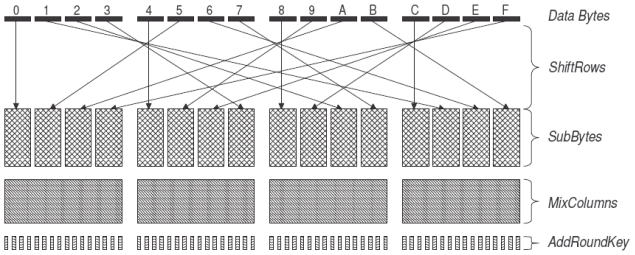

Figure 5. Basic 128-bit AES operation round

From Fig.5, we can explore a great parallelism in SubByte, MixColumns and AddRoundKey. Only the ShiftRows operation expands to all the bytes. The simple starter is to store all the intermediate results into a memory and then try to reuse the SubByte, MixColumns and AddRoundKey components.

B. 32-bit Compact Architecture

The 32-bit compact architecture is a naturally folded one. Since the MixColumn will take 32-bit input and output, it's nice to feed it with a 32-bit word. The input and output memory should remain 128-bit to hold all the intermediate results. The ShiftRows operation is performed by reading the input from different input memory positions.

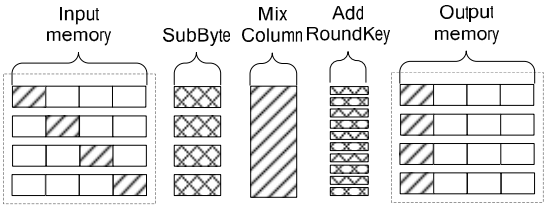

Figure 6. 32-bit Compact round

The following steps will perform the whole round with 4 clock cycles [4].

1. Read input bytes: 0, 5, A, F; execute SubByte, MixColumns and AddRoundKey; write results to the output at locations: 0, 1, 2, 3. this step is highlighted in the Fig. 6.
2. Repeat input bytes: 4, 9, E, 3; write to: 4, 5, 6, and 7.
3. Repeat input bytes: 8, D, 2, 7; write to: 8, 9, A, B.
4. Repeat input bytes: C, 1, 6, B; write to: C, D, E, F.

Output now becomes input for the next step.

C. 8-bit Compact Architecture

The 8-bt compact architecture is much similar to the 32-bit. The MixColumn operation requires at least 32-bit word, so a buffer register is added before it. Since each clock cycle, only one byte operand is fed to the SubByte, there's only one SubByte module needed. As a result, one-byte adder could be shared within four bytes.

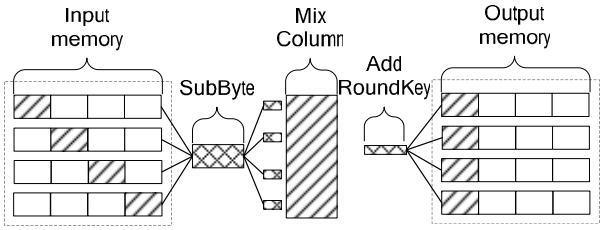

Figure 7. 8-bit Compact Architecture

D. Shared Memory Compact Architecture

In above architectures, the input memory and output memory should be swapped after single iteration. This will introduce more resource usage and more complexity in control logic. By observing that the memory position will be free after all the data are fed to the processing unit, we can reuse them. Fig. 8 shows how to connect the output back to the input memory. By doing this, the output could share the input memory and save half of the total memory usage. This will add complexity to the control logic especially to the memory address part.

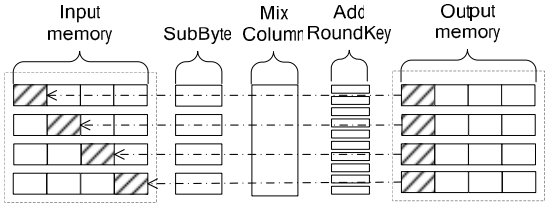

Figure 8. Shared Memory Compact Architecture

And Fig.9 shows the memory address transformation. It has a very nice characteristic that the address could return to its original after 4 clock cycles, which is exactly the time duration of one round.

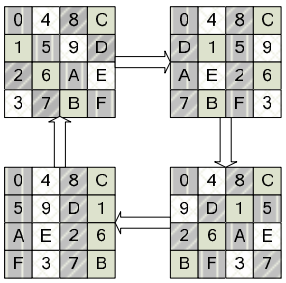

Figure 9. Address transformation of shared memory

E. Pure Logic S-box

S-box in AES is a diffusion transformation composed of a multiplicative inverse in GF(2^8) and an affine transformation. It has an equation as follows:

$$s' = MX \bullet (X^{-1} \bullet s)^{-1} \quad (1)$$

Here the MX is the affine transformation matrix and X^{-1} is the transformation matrix to GF(2^8). All the multiplication is in GF(2^8). Fig.10 shows the SubByte operation.

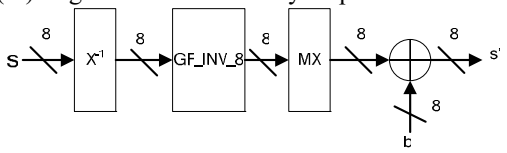

Figure. 10. S-Box Operation for encryption

And the multiplication inverse in GF(2^8) could be decomposed of operations into a sequence of operations in GF(2), which only consists simple XOR gate (addition) and AND gate (multiplication). Thus, the entire inversion in GF(2^8) can be decomposed into a logic circuit composed of XOR and AND gates only [4].

F. Pure logic vs. Distributed RAM S-Box Implementations

In Xilinx Spartan 3 FPGAs, one LUT could be used as a 16x1 single port distributed RAM. And 1 slice has 2 LUTs. If we want to get 4 S-Boxes, the whole number of slices are:

$$4\ SBox = 4 \times 256 \times 8bits = 4 \times [16 \times (1\ LUT \times 8)] = 256\ slices$$

And Fig. 11 shows the real implementations from Synplicity Tools. The distributed RAM implementation requires more slices than the pure-logic.

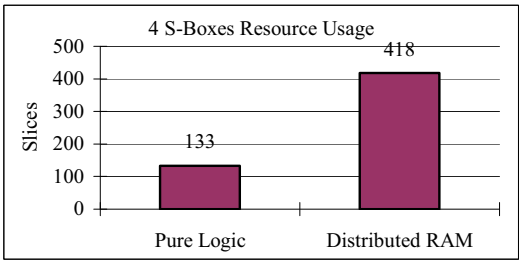

Figure 11. Comparison between pure-logic and distributed RAM

IV. Different Memory Schemes with Different Datapath

Different memory schemas combined with different data paths could make different resource utilizations and throughputs, which needs to pay more attentions on. In this section, we propose different banks of memories and the partial MixColumn operations and compare the resource usage and reach to the best designs.

A. 1-Bank Distributed Memory with 32-bit Datapath

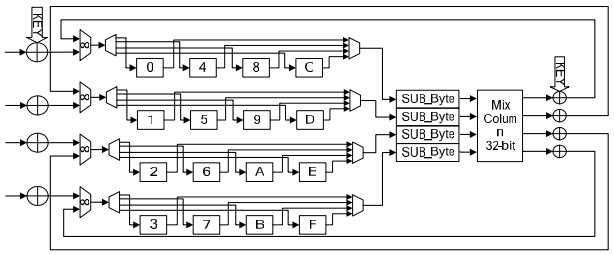

Figure. 12 2-bank shift register memory with 32-bit datapath

There's only one memory bank needed in this schema. The intermediate results are fed back to the same position of the last read-out data. Memory works as individual registers and the logic part updates the state every clock cycle. The advantage here is half of the memory could be saved and there's no need to swap the input and output memory. But this will add complexity to the controller, which needs to generate a set of new addresses, as shown in Fig. 12.

B. 1-Bank Distributed Memory with 8-bit Datapath

This architecture reduces the 32-bit datapath to 8-bit. It has the common 1-bank advantages and disadvantages. However, it could leverage both the memory and the datapath. Nevertheless, we could put more efforts on reducing the datapath and digging more sharing out.

C. A Very Compact Datapath Design

When looking deep into the AES key scheduling and encryption round operations, they involve three types of operations: multi-accumulation, SubByte and XOR. All of these operations manipulate the memory contents. So simply connecting these three to the memory and controlled by a special purpose instruction set will theoretically finish the whole AES functionalities. Fig. 14 shows the very compact architecture. The first line is multi-accumulator, which could perform MixColumn operation in 16 clock cycles. The second line is SubByte. The last line is XOR, which could perform on two different 8-bit values inside the memory and write the results back.

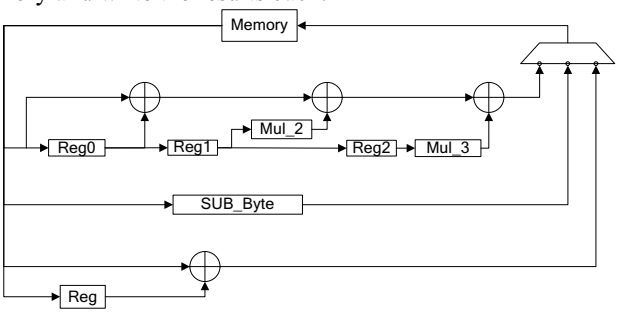

Figure 14. A very compact datapath design

D. Resource Utilization Comparison

The smaller Datapath means smaller resources, as shown in Table I. However, smaller Datapath will require more clock cycles to process one round. The 64-bit could process one whole round in 2 clock cycles, while 8-bit will require 16 clock cycles.

TABLE I
RESOURCE UTILIZATIONS

Architecture			Memory	S-Box*	Mix Column	2:1 MUX	XOR
Bank #	Data path	SBox Share					
2	64		256*8	16	2	128	64
	32		256*8	8	1	64	32
	8		256*8+32	2	1/4	104	8
	64	√	256*8	8	2	192	64
	32	√	256*8	4	1	96	32
	8	√	256*8+32	1	1/4	112	8
1	64		128*8	16	2	64	64
	32		128*8	8	1	32	32
	8		128*8+32	2	1/4	72	8
	64	√	128*8	8	2	128	64
	32	√	128*8	4	1	64	32
	8	√	128*8+32	1	1/4	80	8

*the key scheduling S-Boxes are counted.

V. IMPLEMENTATIONS

The designs are following basic algorithm state machine (ASM) plus Datapath and controller scheme. First, a functional simulation model was build and the behavior is verified by ModelSim again the test vectors. Then, the design was implemented in a synthesizable VHDL and Xilinx XST was used to synthesis the whole design. The tools used are shown in table II.

TABLE II
IMPLEMENTATION TOOLS

Design Step	TOOL
VHDL Simulation	ModelSim SE 6.2b
FPGA Synthesis	Xilinx XST, Synplicity
FPGA Implementation	Xilinx ISE 9.1
Target FPGA	Xilinx Spartan 3

Xilinx SRL components are used to implement the shift registers and LUT is used for distributed RAM. The other optimization methods include retiming and FSM recoding.

A. Result Matrix for Spartan 3

TABLE III
RESULTS FOR SPARTAN3

S-Box	Data path	Bank #	Area (slices)	Max Clock (MHz)	Data rate	Max Throughput (Mbps)	Throughput /Area (10^3)
RAM	8	2	689	117.60	0.73	85.84	124.59
RAM	32	2	728	95.026	2.91	276.53	363.85
RAM	64	2	1191	100.588	5.82	585.42	491.54
logic	8	2	437	75.622	0.73	55.20	126.33
logic	32	2	669	58.367	2.91	169.85	253.88
logic	64	2	958	62.655	5.82	364.65	380.64

Table III shows the smaller the Datapath, the smaller the throughput and area are. Bigger Datapath has better throughput/area ratio. This is because the controller logic will consume a bigger portion of resources for smaller data path. And pure logic S-Box implementations are better in terms of area, but have lower throughput.

B. Different Memory Schemas Implementation Results

TABLE IV
RESULTS FOR SPARTAN3 WITH DIFFERENT MEMORY SCHAMAS

S-Box	memory	Bank #	Area (slices)	Max Clock (MHz)	Data rate	Max Throughput (Mbps)	Throughput /Area (10^3)
Logic	distributed	1	510	52.142	2.91	151.73	297.52
Logic	Shift register	2	511	55.374	2.91	161.14	315.34
Logic	distributed	2	667	54.10	2.91	157.43	236.03
Logic share	distributed	1	341	50.00	2.37	118.50	347.51
Logic share	distributed	4	422	125.1	2.37	296.49	702.58

The results in table IV show that sharing S-Boxes could get much smaller area with slightly lower frequency. However, by adding pipeline and taking advantage of the wasted LUT positions, the max clock frequency could be bumped up to 125.1MHz.

C. Comparison with other AES implementations

Table V presents the results comparison with other implementations. Our design without pipeline could achieve a relatively high throughput with smaller area. Pipelined design could get the best throughput and throughput/area ratio. The area keeps small, showing that this design could achieve a nice balance between performance and resources.

TABLE V
RESULTS COMPARISON WITH OTHER IMPLEMENTATIONS

	Our Design	Out design (pipelined)	Good& Benaissa[10]	Chodoviec &Gaj[10]	Rouvroy et. al[11]
Clock Freq.	50.0	125.1	67	60	71
Datapath	32	32	8	32	32
Slices	341	422	264	522	1231
Throughput	118.5	296.49	2.2	166	208
Throughput /Area (10^3)	347.5	702.58	8.3	318	169

D. Results Analysis

The results show that the AES could be implemented as a stream cipher in different ways. AES shows a good potential to be used as a stream cipher. However, the resources used by AES are bigger due to its complex design. This will need more efforts to simplify the design.

VI. CONCLUSION AND FUTURE WORK

In this paper, the compact design of AES as a stream cipher is introduced. On-the-fly key scheduling is used. Pure logic and distributed RAM based S-Boxes are implemented and compared; showing pure logic has smaller area while distributed RAM has higher speed. Different memory schemes are compared in term of resource utilization and throughput. Compact 8-bit and 32-bitdatapath versions are also implemented to find the best architecture satisfying stream cipher requirements. A very compact datapath with only 1 S-Box, 1 multi-accumulator are proposed to get extremely small area. By sharing S-Boxes between key scheduling and encryption round, we could achieve 118.5Mbps throughput and 341 slices area without pipeline. With pipeline, the numbers become 296.49Mbps and 422 slices. The results show AES has a good potential to be fit to stream cipher.

Future work includes the comparative analysis of AES with the eSTREAM candidates, more FPGA architectures and more compact modes will be researched on.

REFERENCES

[1] Tim Good and Mohammed Benaissa, "AES as Stream Cipher on a Small FPGA," Circuits and Systems, 2006. ISCAS 2006. Proceedings.

[2] David Huang, Mark Chaney, Shashi Karanam, Nich Tom and Kris Gaj, "Comparison of FPGA-Targeted Hardware Implementations of eTREAM Stream Cipher Candidates", CA: Wadsworth, 2008, pp. 123–135.

[3] eSTREAM Project, http://www.ecrypt.eu.org/stream/.

[4] Kris Gaj and Pawel Chodowiec, "FPGA and ASIC Implementations of AES", chapter of "Cryptographic Engineering", Springer US,2008

[5] NIST, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)", Federal Information Processing Standards Publication 197, November 26, 2001

[6] Network Working Group, R. Housley, "RFC 3686 - Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", Internet RFC/STD/FYI/BCP Archives, January 2004.

[7] Pawe_l Chodowiec and Kris Gaj, "Very Compact FPGA Implementation of the AES Algorithm", Cryptographic Hardware and Embedded Systems -- CHES 2003: 5th.

[8] H. Lipmaa, P. Rogaway, and D. Wagner, "CTR-Mode Encryption, Comments to NIST concerning AES Modes of Operations", Symmetric Key Block Cipher Modes of Operation Workshop, 2000.

[9] Kris. Gaj, "AES implementations in software and hardware", ECE746 Lecture, GMU.

[10] T Good, M Benaissa –"AES on FPGA from the Fastest to the Smallest", CHES, 2005 - Springer

[11] G. Rouvroy, F. X. Standaert, el. "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," Procedings of the international conference on Information Technology: Coding and Computing 2004 (ITCC 2004), pp. 583 – 587, Vol. 2, April 2004