Improving AES Core Performance via An Advanced ASBUS Protocol

Xiaokun Yang, University of Houston Clear Lake
Wujie Wen, Florida International University
Ming Fan, Broadcom Corporation

Security is becoming a de-facto requirement of System-on-Chips (SoC), leading up to a significant share of circuit design cost. In this paper, we propose an advanced SBUS protocol (ASBUS), in order to improve the data feeding efficiency of the Advanced Encryption Standard (AES) encrypted circuits. As a case study, the direct memory access (DMA) combined with AES engine and memory controller are implemented as our design-under-test (DUT) using field-programmable gate arrays (FPGA). The results show that our presented ASBUS structure outperforms the AXI-based design for cipher tests. As an example, the 32-bit ASBUS design costs less in terms of hardware resources and achieves higher throughput ($1.30\times$) than the 32-bit AXI implementation, and the dynamic energy consumed by the ASBUS cipher test is reduced to 71.27% compared with the AXI test.

CCS Concepts: •**Hardware** → **Buses and high-speed links; Application specific integrated circuits;** *Design modules and hierarchy;* Arithmetic and datapath circuits; VLSI system specification and constraints; •**Security and privacy** → Hardware-based security protocols;

Additional Key Words and Phrases: Advanced Encryption Standard (AES), Advanced eXensible Interface (AXI), bus protocol, filed-programmable gate array (FPGA), System-on-Chips(SoC)

ACM Reference Format:
Gang Zhou, Yafeng Wu, Ting Yan, Tian He, Chengdu Huang, John A. Stankovic, and Tarek F. Abdelzaher, 2010. A multifrequency MAC specially designed for wireless sensor network applications. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 23 pages.
DOI: 0000001.0000001

1. INTRODUCTION

The rapid rise in Internet-connected devices imposes increasingly higher requirements on high-performance and high-security System-on-Chips (SoC), in terms of low-cost, low-power, and data security. It creates new problems and challenges between the complexity of security algorithms and the limited computing resources of embedded chips. For decades, numerous hardware optimizations were proposed using application-specific integrated circuit (ASIC) [Good and Benaissa 2012] and field-programmable gate arrays (FPGA) [Wang and Ha 2013; N. Mentens and Verbauwhede 2005] on the dominant symmetric-key cryptosystem – the Advanced Encryption Standard (AES) [aes 2001]. However, all the previous research frequently fall back on refining the inside of the AES engines and suppose that data can be input to the engines immediately; indeed, refining the inside of the AES cores is useful, yet the focus, such as data feeding efficiency of the interface, is still on the entire bus architectures.

Basically, AES is a symmetric block cipher that processes on a 4×4 matrix of bytes, named as an AES state. Depending on the key length, each state should be processed

Author's addresses: X. Yang, Department of Engineering, University of Houston Clear Lake; W. Wen, Electrical and Computer Engineering Department, Florida International University; M. Fan, Broadcom Corporation

DOI: 0000001.0000001

10, 12, or 14 rounds. And each round, except for the final round, consists of four different byte-oriented transformations: 1) non-linear byte substitution using a S-box (SubBytes/InvSubBytes), 2) shifting rows of the state array (ShiftRows/InvShiftRows), 3) mixing the data within each column of the state array (MixColumns/InvMixColumns), and 4) adding a round key to the state (AddRoundKey), while the final round does not have the MixColumns/InvMixColumns transformation.

Focusing on the bus interconnection, we simplify SubBytes/InvSubBytes as a black box with 8-bit input and 8-bit output. Likewise, the MixColumns/InvMixColumns is a black box with 32-bit input and 32-bit output. The 32-bit input of MixColumns/InvMixColumns black box is a combination of four shifted bytes from SubBytes/InvSubBytes black boxes. As an example for the encryption engine shown in Figure 1, the first MixColumns(0) processes on the first substituted byte of the first word on bus (SubBytes(0)), the second substituted byte of the second word (SubBytes(5)), the third substituted byte of the third word (SubBytes(a)), and the fourth substituted byte of the fourth word (SubBytes(f)). Similarly, the second, third, and fourth MixColumns transformations process on four bytes that are from different words on bus. Therefore, using the traditional buses in the linear row-major order, such as the AMBA Advanced High-Performance Bus (AHB) [AHB 1999], Advanced eXensible Interface (AXI) [AXI 2003], Wishbone [Wis 2003], and OCP [OCP 2001], the MixColumns/InvMixColumns transformation cannot start until all the 128-bit data being buffered and shifted. Moreover, additional bus commands are needed when bus addresses are non-linear or non-contiguous. The implementations of buffers and rearrangement can increase the slice cost, the extra commands occupy bus cycles and reduce bus efficiency, and the frequent toggle activities of logics, signals & IOs consume much dynamic power.

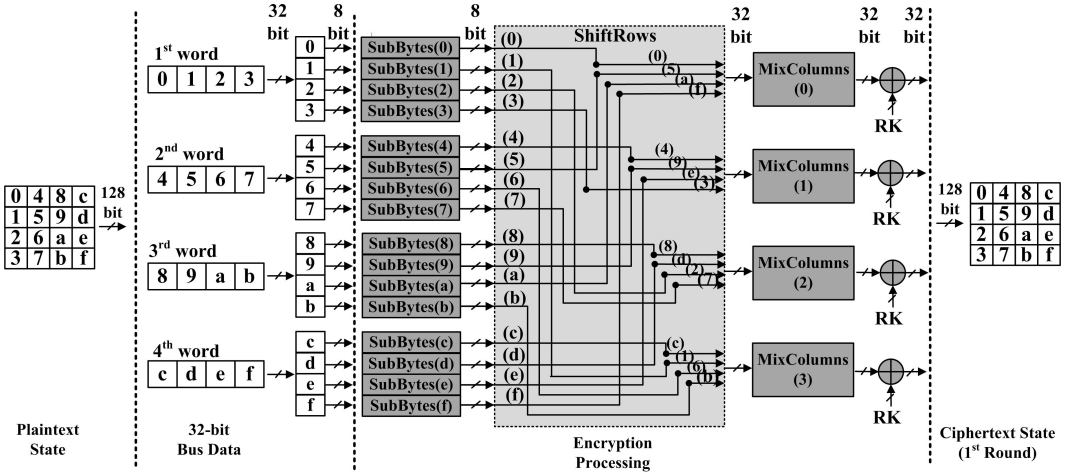


Fig. 1. AES State Processing.

In our previous work [Yang and Andrian 2014], a high-performance SBUS protocol is presented which is able to access data by crossing non-linear boundaries, guaranteeing the high-efficiency for both linear and block transfer modes. In this paper, we make our focus on the remaining AES transfer type on the advanced SBUS (ASBUS) and try to utilize the fact that the AES cryptosystem uses a 4×4 matrix of bytes in shifted/inverse-shifted column-major order. Since AES states are structured in such a way, we propose the use of a novel state transfer, whereby a single command

from a master in a master/slave transfer protocol can put multiple non-linear and pre-scheduled data of states on the bus at the same time. This novel transfer mode does not guarantee the high-performance inside the AES core but showing that the latency and power overhead of the bus interface can be avoided by directly forwarding the cipher processing. In the other words, the presented state transfer can significantly reduce the resource consumption for data buffering and rescheduling by pre-arranging data transferred on bus. The proposed work includes our previous framework that simulates a pre-scheduled interface on the AES engine [Yang and Wen 2017], and is expanded to an advanced bus protocol with 32-, 64-, and 128-bit bus widths. Furthermore, The AES performance is also estimated using static analysis models, register transfer level (RTL) designs, and FPGA based implementations. Our results show that the ASBUS increases valid throughput to $\times 1.30$ and reduces dynamic energy to 71.27% compared with AXI interface.

The organization of this paper is as follows: section 2 briefly reviews the related works of AES designs, and section 3 introduces our AES core implementation and our pervious work – the SBUS protocol. In section 4, our proposed ASBUS and the novel state transfer mode are presented. Then, the bus latencies are statically formulated and analyzed in section 5. In section 6, we implement the direct memory access (DMA) located on ASBUS and AXI buses, combined with AES engines and memory controllers, as our hardware designs. We also illustrate the RTL design, simulation, synthesis, and power analysis in this section. The experimental results are shown in section 7. Finally, section 8 concludes this paper.

2. RELATED WORK

Mathematically, AES operates on the state which is a 4×4-byte matrix. Each state is performed by 10, 12, or 14 rounds, and in each round, except for the final round, four transformations, including SubBytes, ShiftRows, MixColumns, and AddRoundKey are performed for encryption, while InvSubBytes, InvShiftRows, InvMixColumns, and AddRoundKey are performed for decryption.

Among the transformations in AES encryption/decryption, the SubBytes/InvSubBytes transformation is a non-linear operation requiring the highest area and consuming much power of the circuit. Some of the earlier SubBytes/InvSubBytes implementations are based on look-up table (LUT), such as those described in [Fischer and Drutarovsky 2001; McLoone and McCanny 2001; K. Stevens 2005]. The unbreakable LUT accessing limits the high-efficiency applications, such as parallel computation and pipeline operations. Thus, an alternative composite field method for the S-Box computation [Rijmen 2000] is suggested by V. Rijmen, who is one of the AES inventors. Based on this finite field arithmetic, many high-performance implementations are proposed to replace the LUT-based S-Box transformations using combinational logics [Zhang and Parhi 2004; Canright 2005; Mui 2007; J. Wolkerstorfer and Lamberger 2000; A. Satoh and Munetoh 2000].

Moreover, [Zhang and Parhi 2006] and [M. M. Wong and Hijazin 2012] analyze and compare the complexity of the SubBytes/InvSubBytes implementation using different irreducible polynomials. The AES performance is also considered on the core structural level in [C. Hsing Wang and Wu 2010], [S. Fu Hsiao and Tu 2006; AReyhani-Masoleh 2012; Sklavos and Koufopavlou 2012; Hodjat and Verbauwhede 2006; Suntiamorntut and Wittayapanpracha 2012]. For instance, the four primitive transformations are decomposed, rearranged, and regrouped as new linear and non-linear operations in [C. Hsing Wang and Wu 2010] to provide 1.28 Gigabits per second throughput for 128-bit keys. In [S. Fu Hsiao and Tu 2006], the transformations Affine/Inverse Affine, ShiftRows/InvShiftRows, and MixColumns/InvMixColumns are combined into a single

function unit Affine/ShiftRows/MixColumns or InvMixColumns/InvShiftRows/Inverse Affine, and the substructure sharing algorithm is applied to reduce the area cost.

However, the previous research focuses on optimizing the inside of the AES engines, assuming that data can be fed to the engines without any bus protocol overhead. From the system point of view, traditional bus protocols, such as the AMBA AHB [AHB 1999] and AXI [AXI 2003] from ARM Holdings, Wishbone from Silicore Corporation [Wis 2003], OCP from OCP-IP [OCP 2001], CoreConnect from IBM [Cor 1999], and STBus from STMicroelectronics [STB 2004], are very low-efficiency to supply data in the rectangular array of bytes. In fact, the bus interface plays a pivotal role in advancing the AES performance: the resource costs are influenced by the complexity of the data buffering and scheduling, the speed is determined by the data-feeding efficiency, and the energy consumption is dependent on the switching activities of logics, signals & IOs.

3. BACKGROUND

In this section, we briefly illustrate our AES core design from the structural point of view, and then introduce our previous work – the SBUS protocol.

3.1. AES Circuit Structure

The AES standard specifies the Rijndael algorithm, a symmetric block cipher that can process 128-bit states, using cipher keys with lengths of 128, 192, and 256 bits. The key length is represented by N_b=4, 6, or 8, which denotes the number of 32-bit data in the cipher key. For the AES algorithm, the number of rounds to be performed depends on the key size. It is represented by N_r, where N_r=10 when N_b=4, N_r=12 when N_b=6, and N_r=14 when N_b=8. As a case study, we implement the 10-round AES algorithm using the composite field arithmetic. In theory, the composite field of GF(2^8) can be built iteratively from GF(2) using the irreducible polynomials [Paar 1994]:

$$GF(2) \rightarrow GF(2^2) : x^2 + x + 1 \tag{1}$$

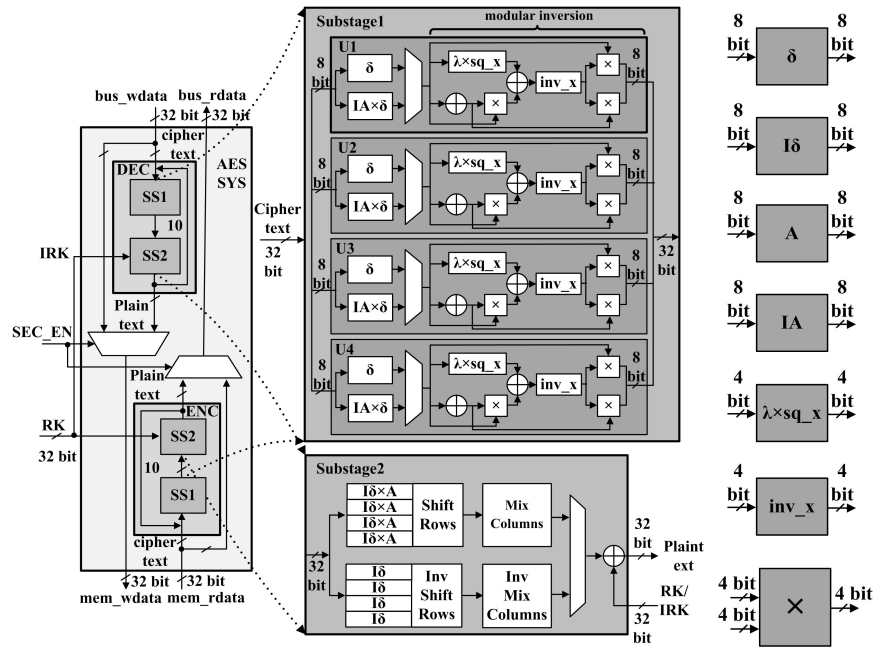
$$GF(2^2) \rightarrow GF((2^2)^2) : x^2 + x + \phi \tag{2}$$

$$GF((2^2)^2) \rightarrow GF(((2^2)^2)^2) : x^2 + x + \lambda \tag{3}$$

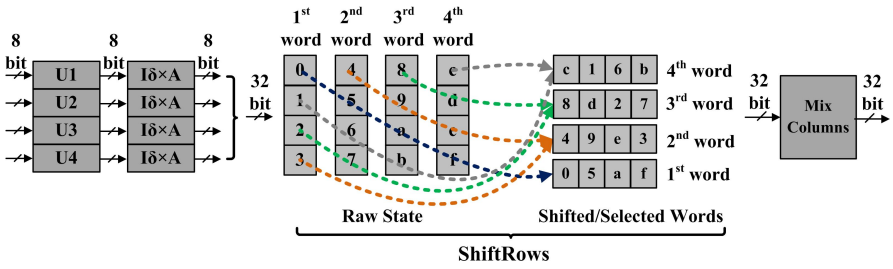
First, x^2+x+1 is the only irreducible polynomial of degree 2 over GF(2). Second, there are two values of ϕ that make x^2+x+ϕ irreducible over GF(2^2), and 8 possible values of λ that make x^2+x+λ irreducible over GF($(2^2)^2$) constructed by using each of ϕ. All together, there are sixteen ways to construct the composite field GF($(((2^2)^2)^2)$) using irreducible polynomials in the equations. As an example, we implement the AES engine using ϕ=$\{10\}_2$ and λ=$\{1100\}_2$ in our work.

Figure 2(a) shows our AES engine design based on a 32-bit interface or bus. For the non-cipher transfers, the encryption engine can be bypassed using the read data path, and the decryption engine can be bypassed using the write data path. For the cipher tests, the AES engines are enabled.

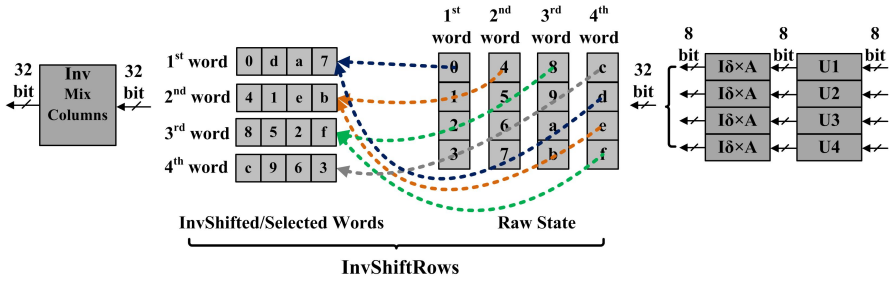
There are two substages, substage1 and substage2, for both encryption and decryption processes. The SubBytes/InvSubBytes transformation is decomposed as a modular inversion over GF(2^4) located in substage1 and four linear functions, A, IA, isomorphic mapping (δ), and inverse isomorphic mapping ($I\delta$). In order to shorten the S-box critical path, IA is combined with δ ($IA \times \delta$) in substage1, and $I\delta$ is merged with A ($I\delta \times A$) in substage2. In addition, the ShiftRows/InvShiftRows, MixColumns/InvMixColumns, and AddRoundKey transformations are integrated in substage2 to obtain approximately equal delay to substage1. Concentrating on the bus efficiency, the key expansion unit is configured by software through the control bus.



(a) AES Core (b) Black Boxes



(c) ShiftRows & MixColumns



(d) InvShiftRows & InvMixColumns

Fig. 2. AES Operator

To simplify the description, we consider all the operators as black boxes as shown in Figure 2(b). As an example, we combine the multiplication with constant λ and squaring in GF(2^4) to reduce the combinational logic cost and shorten the critical path. Let "a" denote the input and "b" denote the output in a one-in, one-out black box hereafter. The bit-width of "a" and "b" are 8-, 4-, and 2-bit, respectively, when the operator is in GF(2^8), GF(2^4), and GF(2^2) fields. Hence, the logic design can be modified as below:

$$\begin{aligned} b_3 =& a_2 \oplus a_1 \oplus a_0 \\ b_2 =& a_3 \oplus a_0 \\ b_1 =& a_3 \\ b_0 =& a_3 \oplus a_2 \end{aligned} \tag{4}$$

In equation 4, the multiplication with constant λ and squaring in GF(2^4) is implemented by "XOR" gates denoted as "$\oplus$" hereafter. Using the combining logic, the design of $\lambda \times sq\_x$ is optimized as 4 "XOR" gates with 2 "XOR" gates in the critical path. It reduces one "XOR" gate delay in the critical path compared to [Zhang and Parhi 2006].

Likewise, the inversion in GF(2^4) field of the modular inversion (inv_x) can be implemented as a 4-bit input and 4-bit output black box, and the other operators, including δ and $I\delta$, A and IA, can be implemented as 8-bit input and 8-bit output black boxes, which are shown in Figure 2(b).

For a two-in, one-out assignment, let "a" and "b" denote 2 inputs, and "c" denote the output hereafter. The bit-width of "a", "b", and "c" are 4-bit and 2-bit, respectively, when the operator is in GF(2^4) and GF(2^2). As an example, the multiplication in GF(2^4) field of the modular inversion can be further decomposed into multiplication in GF(2^2), and then to GF(2). Assume c=$a \times b$, where a=a_Hx+a_L and b=b_Hx+b_L. Here, a_H and b_H are the upper half term, and a_L and b_L are the lower half term. Then, the product of a and b is

$$c = (b_H a_H + b_H a_L + b_L a_H)x + b_H a_H \varphi + b_L a_L \tag{5}$$

This equation is in the form of GF(2^2). In order to decompose the GF(2^2) multiplication to GF(2), the logic for computing GF(2) multiplication is rewritten as

$$\begin{aligned} c_1 =& b_1 a_1 \oplus b_0 a_1 \oplus b_1 a_0 \\ c_0 =& b_1 a_1 \oplus b_0 a_0 \end{aligned} \tag{6}$$

and the logic for computing GF(2) multiplication with constant φ is

$$\begin{aligned} b_1 =& a_1 \oplus a_0 \\ b_0 =& a_1 \end{aligned} \tag{7}$$

Using equation 6 and equation 7, the multiplication in GF(2^4) can be implemented in hardware as a two 4-bit inputs and one 4-bit output black box involving only "XOR" and "AND" gates, as shown in Figure 2(b).

As a 32-bit input and 32-bit output black box, MixColumns/InvMixColumns transformation processes on shifted/inverse-shifted columns of a state. Let the prefix "s_" denote the MixColumns output signal and "is_" denote the InvMixColumns output signal. The logic implementations of MixColumns and InvMixColumns are rewritten as:

$$\begin{aligned} s\_s_0 =& \{02\}(s_0 \oplus s_5) \oplus s_a \oplus s_f \oplus s_5 \\ s\_s_1 =& \{02\}(s_9 \oplus s_e) \oplus s_3 \oplus s_4 \oplus s_e \\ s\_s_2 =& \{02\}(s_2 \oplus s_7) \oplus s_8 \oplus s_d \oplus s_7 \\ s\_s_3 =& \{02\}(s_b \oplus s_c) \oplus s_1 \oplus s_6 \oplus s_c \end{aligned} \tag{8}$$

$$
\begin{aligned}
is\_s_0 &= (\{02\}(s_0 \oplus s_d) \oplus s_a \oplus s_7 \oplus s_d) \oplus \\
&(\{02\}(\{04\}(s_0 \oplus s_a) + \{04\}(s_d \oplus s_7)) + \{04\}(s_0 \oplus s_a)) \\
is\_s_1 &= (\{02\}(s_1 \oplus s_e) \oplus s_b \oplus s_4 \oplus s_e) \oplus \\
&(\{02\}(\{04\}(s_4 \oplus s_e) + \{04\}(s_1 \oplus s_b)) + \{04\}(s_1 \oplus s_b)) \\
is\_s_2 &= (\{02\}(s_2 \oplus s_f) \oplus s_8 \oplus s_5 \oplus s_f) \oplus \\
&(\{02\}(\{04\}(s_8 \oplus s_2) + \{04\}(s_5 \oplus s_f)) + \{04\}(s_8 \oplus s_2)) \\
is\_s_3 &= (\{02\}(s_3 \oplus s_c) \oplus s_9 \oplus s_6 \oplus s_c) \oplus \\
&(\{02\}(\{04\}(s_c \oplus s_6) + \{04\}(s_9 \oplus s_3)) + \{04\}(s_9 \oplus s_3))
\end{aligned}
\tag{9}
$$

In equation 8 and 9, s_x represents the hexadecimal number of byte in the raw state. For example, the first MixColumns output processes on the s_0, s_5, s_a, and s_f bytes of the raw state, and the first InvMixColumns output processes on the s_0, s_d, s_a, and s_7 bytes of the raw state, which are shown in Figure 2(c) and Figure 2(d). In the other words, the four input bytes of the MixColumns/InvMixColumns transformation are selected from four different substituted words or state columns. Since the algorithm processes data in such a way, we optimize the data transfer by feeding selected bytes from data bus for AES engines.

3.2. SBUS Interface

We proposed a high performance on-chip data communication standard termed the Master-Slave bus (MSBUS) in [Yang and Andrian 2014]. It is composed of a control bus (MBUS) and a data bus (SBUS). The control bus (MBUS) is developed as a low-cost and low-power bus, and the data bus (SBUS) is created as a high-throughput full-duplex bus with the feature of block data transfer. In this section, we focus on introducing SBUS interface and the communication protocol.

As a multi-master and signal-slave bus, all the data transfer requests, denoted as REQs, from SBUS masters must be granted access to SBUS first. Only the master with the highest-priority can start a transfer when granted use of the bus, denoted by GNT, by an arbiter. Then, the signals providing information on the address (ADDR), direction (WR), and length (LEN[9:0]) of the transfer, as well as the current transfer mode indicated by the most two significant bits of the LEN[11:10] signal, can be sent by the granted master. Each bit of the write data valid signal (WD_VLD[3:0]) represents the corresponding valid byte of the word-size write data (WDATA[31:0]). Moreover, slaves must send a data response (RESP[1:0]), RESP[1] for write and RESP[0] for read, within a timeout window.

As an example shown in Figure 3, there are two bus operations, one write followed by one read. Each operation consists of two distinct sections: the command phase, involving the bus arbitration and commands, and the data phase including several cycles depending on the burst length. First, the slave cannot receive the write data in cycle 4 and 7 as represented by the de-asserted ready signal RESP[1]. The master thus must hold the write data, 32'h04 and 32'h0c, for another one clock cycle. In cycle 6, the read ready signal denoted by RESP[0] is de-assert, meaning that the read data on SBUS is not available or unstable in this cycle. Hence the master needs to wait one more cycle for a valid read data.

4. PROPOSED ASBUS PROTOCOL

This section presents an ASBUS for designing low-power and high-speed AES-encrypted microcontrollers. It provides a novel state transfer mode, and also backward supports the conventional linear and block transfer modes.

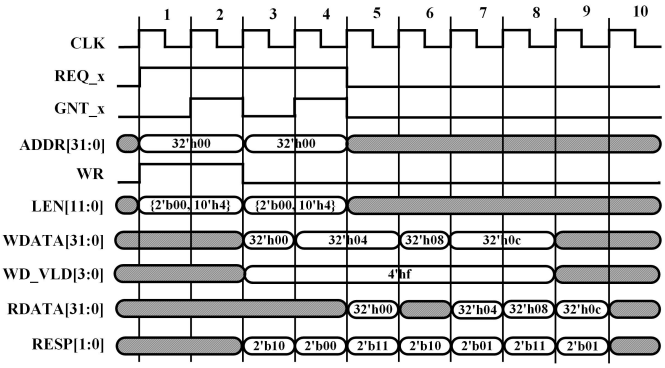
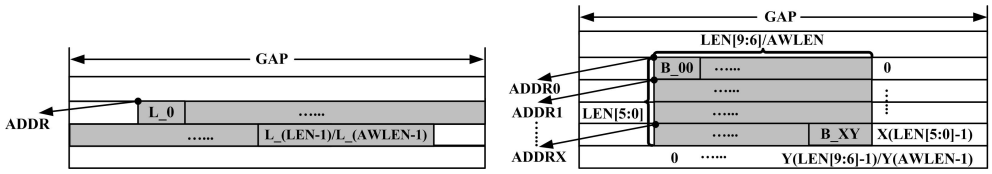


Fig. 3. SBUS/ASBUS Timing Diagram.



(a) Linear (b) Block

Fig. 4. Transfer Modes

4.1. Linear and Block Transfer Modes

Before discussing the state transfer mode, we briefly illustrate the traditional linear and block transfer types. In ASBUS, we define a bus transfer as a linear operation when the LEN[11:10] signal is binary 2'b00 and a block operation when 2'b01. As a linear transfer shown in Figure 4(a), the LEN[9:0] signal gives the exact data number in the row-major order for ASBUS, and the AWLEN signal indicates the number of data transfers in a burst for AXI.

Apart from traditional linear data transfer, the block transfer is supported by ASBUS to improve the performance of matrix-based applications in some specific fields, such as image processing [Gonzalez and Woods], computer vision, and wireless communication [wir 1999]. It defines the rectangle size and makes every memory boundary-crossing command computable by hardware, so that the time consumption of software configuration and bus commands is reduced. As shown in Figure 4(b), the LEN[5:0] signal denotes the block height and the LEN[9:6] signal denotes the block width. Since the non-linear addresses are computable by hardware using the ASBUS protocol, only the initial address (ADDR0) is needed for the entire bursts. In the contrast, each boundary-crossing addresses, from ADDR0 to ADDRX, should be initiated by the granted master using traditional buses. As an example, AWLEN represents the burst length of each linear transfer for the AXI bus.

Furthermore, Figure 5 compares the AXI and ASBUS timing diagrams as a case study. In Figure 5(a), one address stage and one response stage are needed to transfer four words in linear mode using AXI. The one response cycle can be reduced using ASBUS as shown in Figure 5(b), however, because the data received/valid signal (RESP) is driven in the same cycle of the data stage (DATA).

Figure 5(c) and Figure 5(d) show the examples of 4×4-byte matrix transfer. Using the AXI bus, four boundary-crossing or noncontiguous addresses (A0, A1, A2, and A3) associated with four responses (R0, R1, R2, and R3) are required to access the ma-

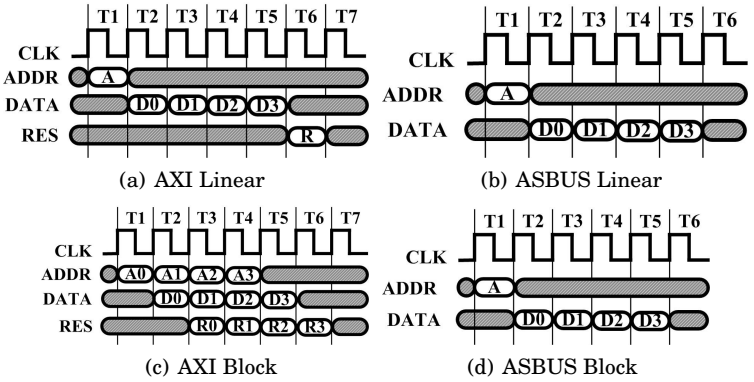


Fig. 5. Transfer Timing

trix. However, only the initial address (A) is needed using ASBUS. All the non-linear addresses can be calculated by hardware, which is defined by the ASBUS protocol.

4.2. Proposed State Transfer Mode

A novel transfer mode, the AES state transfer, is the main contribution to the ASBUS architecture in this paper. It advantageously optimizes data supply efficiency involving encryption/decryption processing. This transfer mode may reduce the processing load of data scheduling and buffering and power consumption in system environments making use of AES cryptographic processing.

Figure 6 shows the memory layout, where only one address (ADDR) is required to transfer several AES states (from S_00 to S_XY). First, in the AES state transfer mode, the "AES state" is adopted as the basic unit of data transfer on the ASBUS. Second, the AES state transfer is processed on the ASBUS in the column-major order, rather than the row-major order as the linear and block transfer types. Third, in a "read" operation, the plaintext state is cyclically-shifted into the encryption engine, and in a "write" operation the ciphertext state is cyclically-inverse-shifted into the decryption engine.

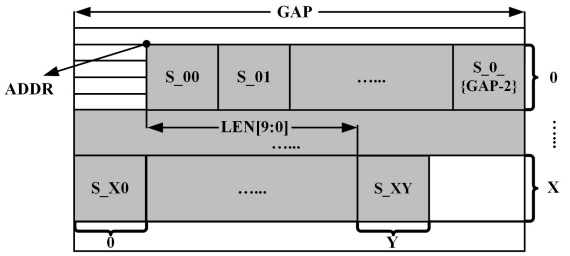


Fig. 6. State Transfer Mode.

More specifically, Figure 7(a) illustrates a two-state transfer example. Assume the byte sequence in the raw state is hexadecimal "0" to "3", "4" to "7", "8" to "b", "c" to "f" for the first, second, third, and fourth columns, respectively. However, the first write data driven on the 32-bit ASBUS is "0", "5", "a", and "f", the second write data sequence is "4", "9", "e", and "3", the third write data sequence is "8", "d", "2", and "7", and the

fourth write data sequence is “c”, “1”, “6”, and “b”. In such a way, the write data selected from each column of the state array can be mixed and encrypted immediately.

Likewise, the first read data sequence is “8”, “5”, “2”, and “f”, the second read data sequence is “c”, “9”, “6”, and “3”, the third read data sequence is 8, 5, 2, and f, and the fourth read data sequence is c, 9, 6, and 3, which are cyclically shifted before entering the encryption engine. Therefore, the selected read data from each column of the state array can be inversely mixed and decrypted immediately.

In addition, notice that only one command (C0) is needed to access two AES states (S0 and S1). In Figure 7(b) and Figure 7(c) , the “LEN[11:10]” signal is binary 2’b10, and the other bits represents the state number as two states. Each bit of WD_VLD indicates the valid byte of the write data, “1’b1” for valid and “1’b0” for invalid. The write data is rescheduled in the cyclically-inverse-shifted order, and the read data is rescheduled in the cyclically-shifted order.

Finally, we compare the bus transfers between AXI and ASBUS. As an example shown in Figure 8(a), six command cycles, involving four addresses (A0, A1, A2, and A3) and four responses (R0, R1, R2, and R3) with two of them overlapped, are needed. The sustatge1 of the first encryption round starts at the T4 cycle after the first data being stably sampled. However, the sustatge2 cannot be started immediately, due to the requirement of the shifted/inverse-shifted rows of the state for the MixColumns/InvMixColumns transformation. In this case, the substage2 cannot be initiated until the whole state being buffered and rescheduled at the T7 cycle. In sum, the first round costs eleven cycles for one state encryption.

Comparing with AXI bus, ASBUS uses only the initial address (A) to transfer multiple states. More important, the substage2 can be immediately started after the first word of substage1 being stably sampled at the T4 cycle, because each word of substage1 is pre-scheduled by ASBUS and ready-to-use for the MixColumns/InvMixColumns transformation. Likewise, the second, third, and fourth words of the state can be consecutively encrypted at the T5, T6, and T7 cycles.

5. STATIC ANALYSIS MODELS

In this section, we formulate and compare performance metrics of AXI and ASBUS to estimate ASBUS efficiency.

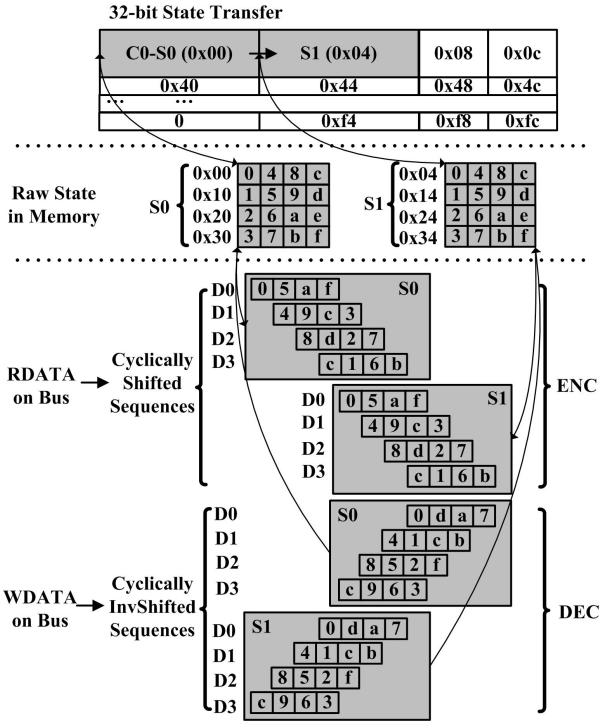
5.1. Transfer Latency Models

In order to focus on the bus efficiency, we assume that the bus grant to any request and the bus response to any transfer are always available immediately. So both of the arbitration and the command cost only two clock cycles. In addition, the request, grant, and address transactions can be overlapped between two back-to-back transfers.

Let P_{XL} and P_{AL}, respectively, denote the probability of the back-to-back transfers of AXI and the probability of the back-to-back transfers of ASBUS in the linear mode. Hence, the AXI linear (XL) transfer latency, denoted by CY_{XL}, can be formulated as

$$CY_{XL} = 4 \times ceil(\frac{N_L}{XS}) + N_L - 2 \times ceil(\frac{N_L}{XS}) \times P_{XL}. \tag{10}$$

where N_L represents the number of data bursts, and P_{XS} ranges from 0 to [ceil(N_L/XS)-1]/ceil(N_L/XS). In this equation, the ceil() function represents that rounds fraction up. XS indicates the maximum AXI burst size, specified by ARLEN for read and AWLEN for write, which is 16 for AXI3 or 256 for AXI4 compatibility [**?**]. Each AXI transfer requires four cycles for arbitration and command stages, two cycles for handshaking between a request and a grant, one cycle for address, and one cycle for response. When two transfers are back-to-back, two command cycles can be overlapped.



(a) Data Access

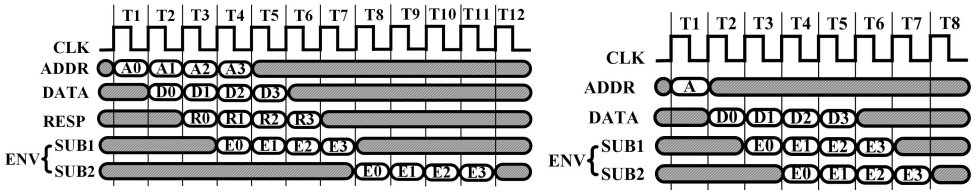
C0: {ADDR, LEN, WD_VLD, WR}={32'h00, 12'h802, 4'hf, 1'b1}
D0: WDATA->{0x00, 0x13,0x22,0x31, 0x01, 0x10, 0x23, 0x32}
D1: WDATA->{0x02, 0x11,0x20,0x33, 0x03, 0x12, 0x21, 0x30}
D2: WDATA->{0x04, 0x17,0x26,0x35, 0x05, 0x14, 0x27, 0x36}
D3: WDATA->{0x06, 0x15,0x24,0x37, 0x07, 0x16, 0x25, 0x34}

(b) Write Command

C0: {ADDR, LEN, WR}={32'h00, 12'h802, 1'b0}
D0: RDATA->{0x00, 0x11,0x22,0x33, 0x01, 0x12, 0x23, 0x30}
D1: RDATA->{0x02, 0x13,0x20,0x31, 0x03, 0x10, 0x21, 0x32}
D2: RDATA->{0x04, 0x15,0x26,0x37, 0x05, 0x16, 0x27, 0x34}
D3: RDATA->{0x06, 0x17,0x24,0x35, 0x07, 0x14, 0x25, 0x36}

(c) Read Command

Fig. 7. State Transfer Access



(a) AXI State Transfer

(b) ASBUS State Transfer

Fig. 8. State Transfer Timing

In contrast, ASBUS integrates the arbitration and address phases together, and also combines the data and slave-driven response phases. Therefore, it uses only two cycles with an immediate grant. The total latency of ASBUS transfers, denoted by CY_{AL}, thus is

$$CY_{AL} = 2 \times ceil(\frac{N_L}{AS}) + N_L - 2 \times ceil(\frac{N_L}{AS}) \times P_{AL}. \quad (11)$$

where AS represents the maximum ASBUS transfer size, which is 1024 beats for the 10-bit ASBUS transfer length signal. In this equation, $P_(AL)$ ranges from 0 to [ceil(N_L/AS)-1]/ceil(N_L/AS).

AXI protocol does not define how to access data by block, so designers must consider the specific operations for the matrix-based applications and algorithms. Using the AXI linear transfer type, the cycle cost for a block processing can be calculated as

$$CY_{XB} = 4 \times N_H \times ceil(\frac{N_W}{XS}) + N_H \times N_W - 2 \times N_H \times ceil(\frac{N_W}{XS}) \times P_{XB}. \quad (12)$$

Here, N_W and N_H, respectively, denote the block width and block height. P_{XB} represents the probability of the back-to-back AXI transfers within the same row, ranging from 0 to [N_H $\times$ ceil(N_W/XS)-1]/[N_H $\times$ ceil(N_W/XS)].

ASBUS requires only one command stage for each matrix access by means of the build-in boundary-crossing scheme using the block transfer. Thus, the total cycle cost of an ASBUS block transfer can be formulated as

$$CY_{AB} = 2 \times ceil(\frac{N_H}{AH}) \times ceil(\frac{N_W}{AW}) + N_H \times N_W - 2 \times ceil(\frac{N_H}{AH}) \times ceil(\frac{N_W}{AW}) \times P_{AB}. \quad (13)$$

where AH and AW are the maximum block height and the maximum block width that can be processed by the ASBUS block transfer. In this work, AH is 64 due to the 6-bit block height signal LEN[5:0], and AW is 16 due to the 4-bit block width signal LEN[9:6]. P_{AB} denotes the probability of the back-to-back ASBUS block transfers, ranging from 0 to [ceil(N_H/AH) $\times$ ceil(N_W/AW)-1]/[ceil(N_H/AH) $\times$ ceil(N_W/AW)].

Finally, the latency of the AES cipher tests using AXI and ASBUS is considered. In our work, not only the command and data cycles but also the AES encryption/decryption latency is calculated. Assume that the encryption/decryption processing is fully pipelined, each cipher round thus uses five clock cycles for the 32-bit bus, in which four cycles are consumed by substage1 and four cycles are consumed by substage2 with three of them overlapped. Likewise, three cycles are needed for the 64-bit bus and two cycles are needed for the 128-bit bus to complete each AES state round. Furthermore, assume that all the transfers are back-to-back, and the command stages, data stages, and AES cipher/inverse-cipher operations are completely overlapped. The total number of cycles spent by the 32-, 64-, and 128-bit AXI encryption/decryption (XE) procedures are

$$CY_{XE32} = 2 + 6 \times N_E + 50 \times N_E - (12 \times N_E + 38 \times N_E) \times P_{XE}. \quad (14)$$

$$CY_{XE64} = 2 + 4 \times N_E + 30 \times N_E - (6 \times N_E + 24 \times N_E) \times P_{XE}. \quad (15)$$

$$CY_{XE128} = 2 + 3 \times N_E + 20 \times N_E - (3 \times N_E + 17 \times N_E) \times P_{XE}. \quad (16)$$

where N_E denotes the number of AES states. In these three equations, the AXI back-to-back probability, denoted as $P_(XE)$, ranges from 0 to (N_E-1)/N_E .

Using the specific state transfer mode, ASBUS consumes only one command to transfer multiple AES states. The number of data cycles depends on the ASBUS width. For instance, the 32-, 64-, and 128-bit ASBUS, respectively, need $4N$, $2N$, and N data

Table I. Modeling Performance Comparison

Tests	CY
AXI linear	$(4-2P)ceil(\frac{N_L}{XS})+N_L$
ASBUS linear	$(2-2P)ceil(\frac{N_L}{AS})+N_L$
AXI block	$(4-2P)\times N_H \times ceil(\frac{N_W}{XS})+N_H \times N_W$
ASBUS block	$(2-2P)\times ceil(\frac{N_H}{AH})\times ceil(\frac{N_W}{AW})+N_H \times N_W$
AXI Cipher32	$2+2N_E(28-25P)$
AXI Cipher64	$2+2N_E(17-15P)$
AXI Cipher128	$2+N_E(23-20P)$
ASBUS Cipher32	$2+2N_E(27-25P)$
ASBUS Cipher64	$2+2N_E(16-15P)$
ASBUS Cipher128	$2+N_E(21-20P)$

cycles to transfer N_E states. Hence, the total cycles consumed by ASBUS encryption/decryption (AE) tests are

$$CY_{AE32} = 2+4\times N_E+50\times N_E-(4\times N_E+46\times N_E)\times P_{AE}. \quad (17)$$

$$CY_{AE64} = 2+2\times N_E+30\times N_E-(2\times N_E+28\times N_E)\times P_{AE}. \quad (18)$$

$$CY_{AE128} = 2+N_E+20\times N_E-(N_E+19\times N_E)\times P_{AE}. \quad (19)$$

for the 32-, 64-, and 128-bit ASBUS, respectively. In Table I, we simplify and summarize all the above analysis using the back-to-back probability ranging from 0 to 1.

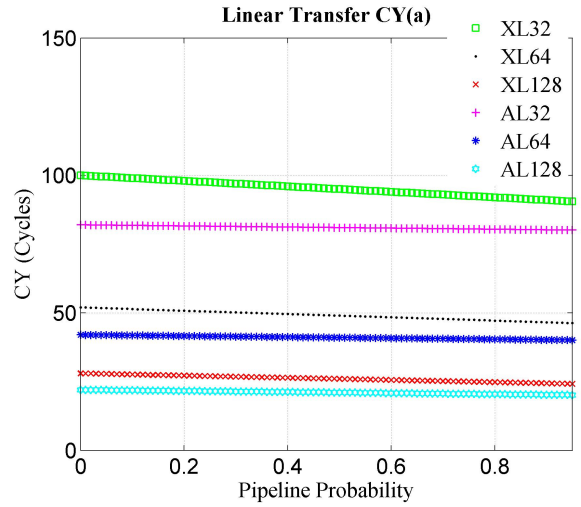
5.2. Static Performance Analysis

In what follows, we compare the bus latency using AXI and ASBUS in Figure 9. The burst sizes are 80 words, 10×8 words, and 20 AES states, respectively, for linear, block, and state transfer tests. In the other words, N_L, N_H N_W, and N_E, are 80, 8, 10, and 20, respectively, in Table I. The horizontal axis represents the back-to-back probability (P) ranging from 0 to 0.95.

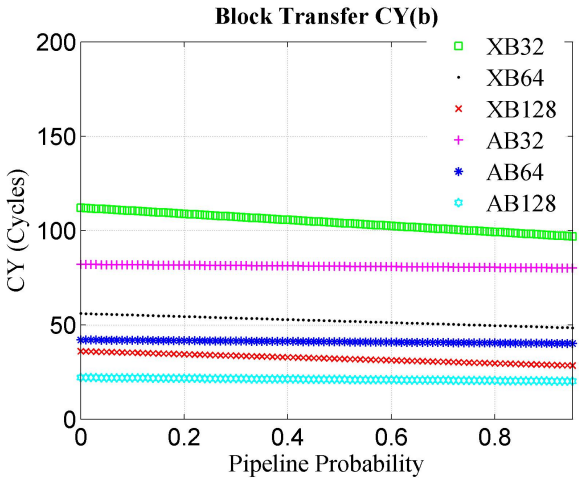
In Figure 9(a), it can be observed that the latency can be reduced when many data transfers are back-to-back, or the back-to-back probability is high. When the probability reaches the maximum 0.95, the clock cycles consumed by the ASBUS linear transfers are 88.51%, 86.61%, and 83.06% compared with the AXI linear tests for 32-, 64-, and 128-bit buses, respectively. Likewise, the clock cycles consumed by the ASBUS block transfers are 82.75%, 82.85%, and 70.77%, respectively, compared with the AXI block tests, for all the three bus sizes' tests, which is shown in Figure 9(b).

Furthermore, the comparison between AXI and ASBUS cipher tests are shown in Figure 10. For the same bus sizes, the ASBUS cipher test costs less cycles than the AXI transfer, particularly when the back-to-back probability is high. As an example, when the back-to-back probability is the maximum 0.95, the clock cycles consumed by ASBUS transfers are 76.74%, 64.29%, and 51.22% compared with AXI transfers for 32-, 64-, and 128- buses, respectively. Additionally, the latency of ASBUS tests are 96.43%, 94.13%, 91.32% of AXI tests when the probability is 0. Thus, the novel state transfer mode can achieve higher throughput than AXI, particularly the bus transfers are frequent and consecutive.

Second, the resource costs are considered statically. To improve the AES speed, we need to pay for large overhead logics and optimize the number of parallel resource costs. Figure 11 shows the pipeline structures and the resource costs depending on different bus-based implementations. Let S denote the logic utilization of substage1, and let M denote the logic cost of substage2. When the bus size is 32-bit as shown in Figure 11(a), four parallel S (4S) connected with one M (1M) instances are necessary to internally pipeline and parallel all the ten-round cipher/inverse-cipher processing per state.



(a) Linear



(b) Block

Fig. 9. Static Performance Analysis.

Furthermore, the resources are doubled to externally parallel the write and read channels of the full-duplex bus. As the 64-bit bus-based implementation shown in Figure 11(b), the cipher/inverse-cipher processing can be sped up, but the S and M instances are doubled. It requires 8 S (8S) and 2 M (2M) instances for the encryption/decryption process of each round to make all the data transfer internal pipeline and parallel. Similarly, sixteen S (16S) and four M (4M) of each round are necessary to the 128-bit bus-based implementation shown in Figure 11(c).

As an alternative technology to the ASIC design, FPGA implements the basic combinational logic by the 2^k-bit static random-access memory (SRAM), which represents a K-input and one-output LUT. Different from the logic gate computation, it is capable of realizing any Boolean function of up to K variables by loading the SRAM cell with the truth table of that function. Therefore, although the 128-bit bus design costs

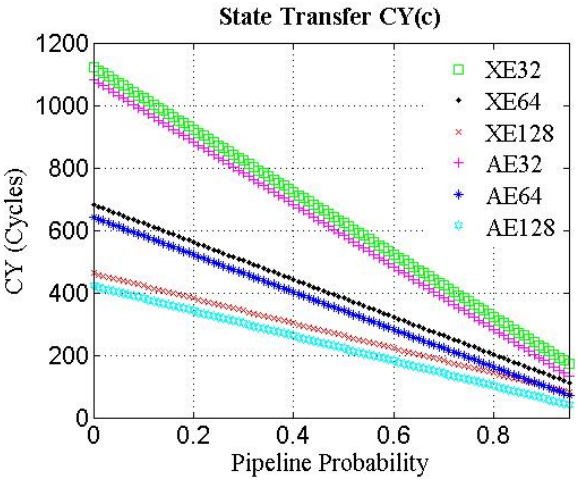
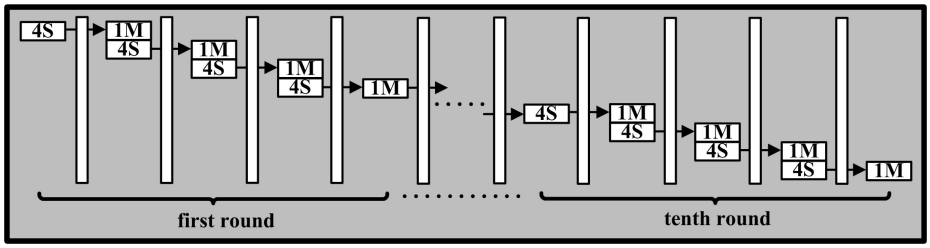
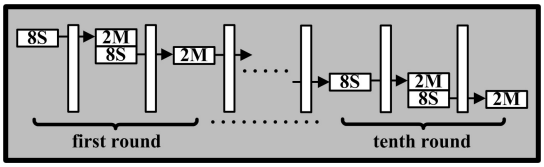


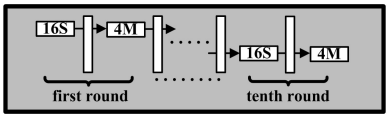
Fig. 10. Static Performance Analysis for State Transfer.



(a) 32-bit bus



(b) 64-bit bus



(c) 128-bit bus

Fig. 11. Pipeline Structures of AES Cores.

more S and M instances, it reduces the FPGA slice usage due to the short path of each cipher/inverse-cipher round. However, comparing with 32-bit bus, the 128-bit bus sacrifices the power consumption due to the high toggle activity of IOs and signals.

6. HARDWARE IMPLEMENTATION

This section presents all the 32-, 64-, and 128-bit implementations using AXI and ASBUS, targeted to accurately evaluate the architectural performance. We use Verilog HDL [Ver 2001] to complete the RTL design, and set up a universal verification methodology (UVM) [uvm 2011] and [uvm 2012] environment to verify all the design under tests (DUTs). Finally, the FPGA back-end flow is performed to estimate the area cost and power consumption.

6.1. RTL Design and Verification

In our study, we implement all the 32-, 64-, and 128-bit AXI and ASBUS DMAs, combined with AES cores and memory controllers (XDAM and ADAM) as the DUTs. As the design structure shown in Figure 12, the XDAM/ADAM can be accessed by all the masters located on AXI/ASBUS. All the master requests are granted sequentially, according to each master's priority. The arbiter performs this function by observing a number of different requests, and deciding which is currently the highest priority master.

In addition, eight commands can be preprocessed using command queues. The data path modules – write data path and read data path, are used to multiplex cipher and non-cipher data processing between AXI/ASBUS masters and memory. More specifically, the AES ENC/DEC engine is bypassed for the conventional linear and block transfers. For the cipher tests, the write data path decrypts the ciphertexts then writes the plaintexts into memory, or the read data path encrypts the plaintexts from memory then transfer the ciphertexts on AXI/ASBUS. Finally, the memory controller's address, mapping from hexadecimal 0x00 to 0xff, is used to provide the control signals for external memory.

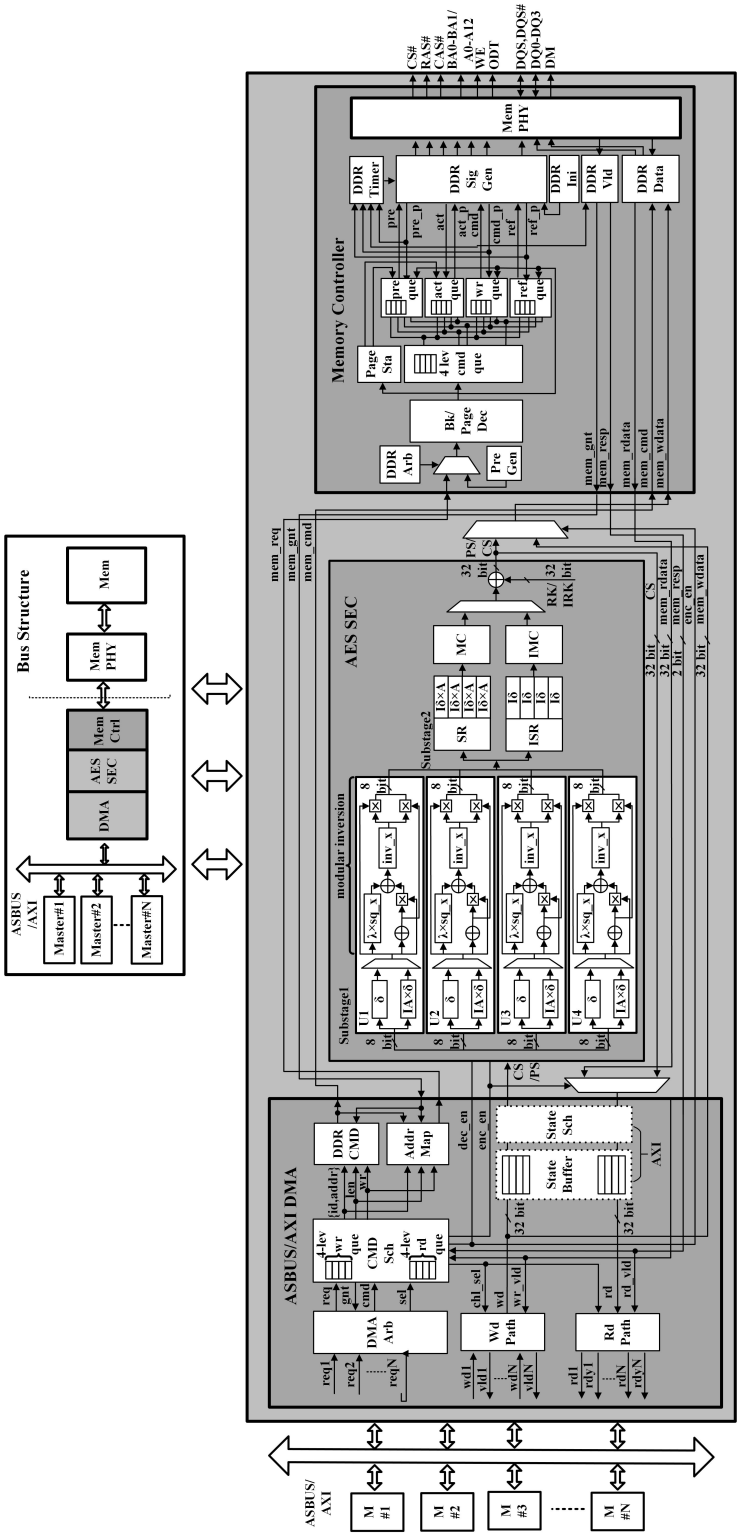


Fig. 12. ADAM Structure.

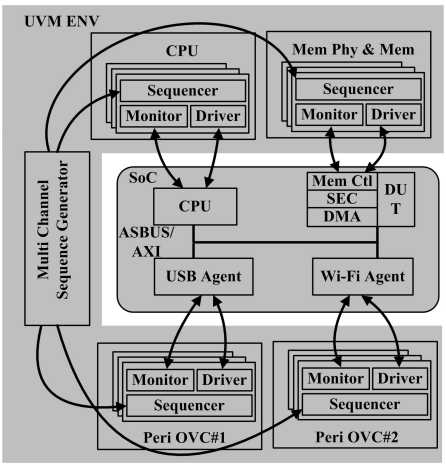


Fig. 13. UVM Environment.

To verify DUTs and evaluate transfer performance, we build up a UVM-based verification environment shown in Figure 13. It integrates four encapsulated ready-to-use and configurable verification agents: the micro-processor, the memory phy, and two AXI/ASBUS masters indicated as Peripheral OVC #1 (USB2.0 Host Controller) and Peripheral OVC #2 (Wi-Fi Mac). Each of them contains three components: the sequencer, driver, and monitor.

The typical test cases used in our study are that 40 words, 10×4 words, and 10 AES states are written into memory then read out. The non-cipher tests, including linear and block tests, bypass the AES engines. In contrast, the cipher tests employ the AES core as data encryption/decryption. For example, the USB2.0 agent initiates a 10-state write command to the data bus. The initial address is hexadecimal 0x00 and the data on AXI/ASBUS are ciphtertext. Then, our DUT, the XDAM or ADAM, responds to the request, decrypts the ciphertext, and then writes the plaintext into memory. To verify the bus function, the Wi-Fi Mac agent requests a 10-state read operation to the same memory address. Likewise, our DUT responds the request, reads data out and encrypts the plaintext to be ciphertext, and then sends them on AXI/ASBUS.

6.2. Area and Power Analysis

In what follows, all the 32-, 64-, and 128-bit XDAMs and ADAMs are synthesized and placed & routed using Xilinx ISE 14.7 with the target device Virtex6xc6vlx550t-2ff1760 [xil 2012]. Then, several fully placed & routed native circuit description (NCD) and physical constraint files (PCF) are generated.

Table II shows the synthesis results, including IO number, resource utilization, and the maximum operational frequency (MOF). In the second column, it can be observed that ADAM uses less IOs than XDAM for the same bus sizes. For different bus sizes, it is obvious that the 128-bit bus costs much more IOs than the 32- and 64-bit buses. As shown in the third column, the total number of occupied slices of ADAMs are less than the XDAMs for all the 32-, 64-, and 128-bit buses. Moreover, the compact ASBUS structure achieves higher operational clock frequency than AXI, for all the three bus sizes, which is shown in the fourth column.

Furthermore, inputting all the NCD and PCF files, as well as the simulation value change dump (VCD) files into the XPower Analyzer tool, the power statistics of AXI- and ASBUS-based designs can be obtained in Table III. Since static power consump-

Table II. Resource Comparison

Resource Costs	IOs	Slices	MOF (MHz)
32-bit XDAM	533	26106	133.010
32-bit ADAM	324	24822	183.636
64-bit XDAM	661	22603	131.528
64-bit ADAM	460	21319	176.154
128-bit XDAM	917	18344	130.152
128-bit ADAM	732	17060	184.176

Table III. Power Consumption

Test Cases	Static Power (mW)	Dynamic Power (mW)	Total Power (mW)
32-bit AXI linear	3799	612	4411
64-bit AXI linear	3796	577	4373
128-bit AXI linear	3797	623	4420
32-bit ASBUS linear	3796	574	4370
64-bit ASBUS linear	3794	540	4335
128-bit ASBUS linear	3796	584	4381
32-bit AXI block	3801	752	4553
64-bit AXI block	3812	971	4783
128-bit AXI block	3826	1263	5089
32-bit ASBUS block	3798	695	4493
64-bit ASBUS block	3802	927	4729
128-bit ASBUS block	3828	1226	5054
32-bit AXI Cipher	3805	771	4576
64-bit AXI Cipher	3818	1063	4881
128-bit AXI Cipher	3852	1747	5599
32-bit ASBUS Cipher	3802	716	4518
64-bit ASBUS Cipher	3816	995	4810
128-bit ASBUS Cipher	3847	1650	5497

tion is mostly determined at the circuit level, the static power of the same design is almost a constant for different test cases, as shown in the second column. Our work, thus, concentrates on analyzing dynamic power shown in the third column.

First of all, it can be observed that the ASBUS tests consume less dynamic power compared with AXI tests, because of the less toggle rate of logics, signals, and IOs. In addition, the wider bus consumes more dynamic power in all the block and cipher tests. In the linear mode, however, the 32-bit bus consumes more dynamic power than the 64-bit bus, because the signal and IO switching rate is very low in this case and the clock power becomes the dominant factor of the dynamic power consumption.

7. EXPERIMENTAL RESULTS

In this section, we summarize the experimental results involving clock cycle (CY), dynamic energy (DE), valid bandwidth (VDB), dynamic energy (DE), slice efficiency (SE), and dynamic energy efficiency (DEE) in Table IV.

It is clear to illustrate the performance comparison between ADAM and XDAM in Figure 14. Figure 14(a) and Figure 14(b) show the performance ratios of non-cipher tests, including linear and block cases. Since all the time consumption ratios are less than 1, ASBUS consumes less time than the AXI for all the three bus sizes. Particularly for the block tests, the latency of ASBUS are 83.67%, 84.00%, and 73.33%, respectively, compared with AXI for all the 32-, 64-, and 128-bit buses.

Additionally, the dynamic energy, which is the integral of dynamic power, or the product of average dynamic power and transfer time, is considered. Although the dynamic power consumed by ADAM and XDAM are close to each other, the dynamic energy consumption of ASBUS linear tests are 83.60%, 81.89%, and 79.32%, respectively, compared with the AXI linear tests, and the dynamic energy consumption of

Table IV. Experimental Results Comparison

Tests	Clock Cycles	VDB (GBps)	DE (uJ)	SE (KBps/Slice)	DEE (GBps/J)
32-bit AXI linear	92.00	0.70	0.56	26.65	1.14
64-bit AXI linear	48.00	1.33	0.28	58.99	2.31
128-bit AXI linear	26.00	2.46	0.16	134.19	3.95
32-bit ASBUS linear	82.00	0.78	0.47	31.44	1.36
64-bit ASBUS linear	42.00	1.52	0.23	71.48	2.82
128-bit ASBUS linear	22.00	2.91	0.13	170.52	4.98
32-bit AXI block	98.00	0.65	0.74	25.02	0.87
64-bit AXI block	50.00	1.28	0.49	56.63	1.32
128-bit AXI block	30.00	2.13	0.38	116.30	1.69
32-bit ASBUS block	82.00	0.78	0.57	31.44	1.12
64-bit ASBUS block	42.00	1.52	0.39	71.48	1.64
128-bit ASBUS block	22.00	2.91	0.27	170.52	2.37
32-bit AXI Cipher	172.00	0.37	1.33	14.25	0.48
64-bit AXI Cipher	112.00	0.57	1.19	25.28	0.54
128-bit AXI Cipher	82.00	0.78	1.43	42.55	0.45
32-bit ASBUS Cipher	132.00	0.48	0.95	19.53	0.68
64-bit ASBUS Cipher	72.00	0.89	0.72	41.69	0.89
128-bit ASBUS Cipher	42.00	1.52	0.69	89.32	0.92

ASBUS block tests are 77.33%, 80.19%, and 71.19%, respectively, compared with the AXI block tests, for all the 32-, 64-, and 128-bit bus implementations.

Moreover, based on the fair assumption of the same operational clock frequencies for ASBUS and AXI, the conventional bandwidth between full-duplex ASBUS and AXI are the same. However, when we consider the valid data bandwidth defined as the valid data without protocol overhead that can be transferred in one cycle, the ASBUS surpasses AXI due to the high-efficient protocol. For example, the valid data bandwidth of ASBUS linear test is 1.18 times of AXI linear test, and the valid data bandwidth of ASBUS block test can reach 1.36 times of AXI block test, when the bus size is 128 bit.

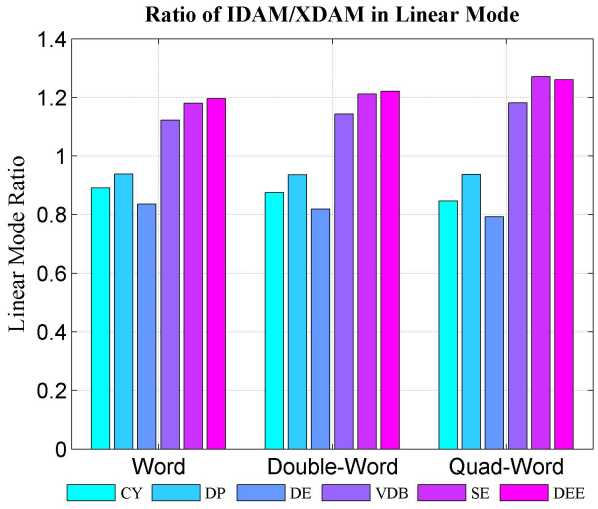
Furthermore, the slice efficiency is also computed in terms of valid data number that can be transferred per second per slice. It can be observed that the slice efficiency of ASBUS linear tests are around 1.27 times of AXI linear tests, and the slice efficiency of ASBUS block test is 1.47 times compared with AXI block test when the bus size is 128 bits.

Finally, dynamic energy efficiency is defined in terms of valid data number that can be transferred per second per watt, or valid data number that can be transferred per joule. The dynamic energy efficiency of ASBUS linear tests are around 1.26 times compared with AXI linear tests for all the three bus-size designs, and the dynamic energy efficiency of ASBUS block test can reach 1.40 times of AXI block test when the bus size is 128 bits. In other words, ASBUS can transfer 1.40 times as much data as AXI with the same time and power consumption in this case.

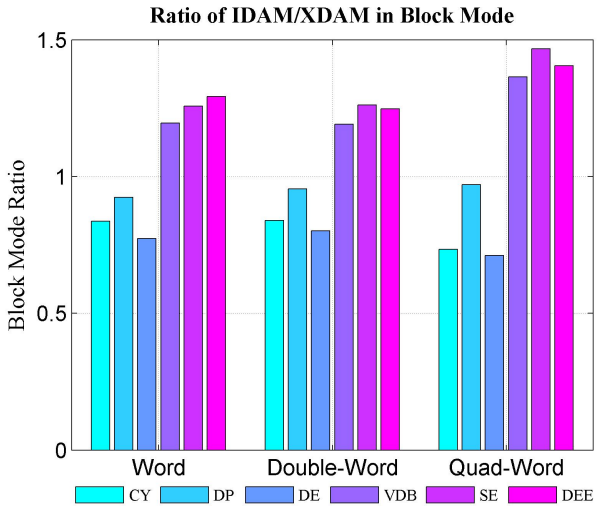
In this paper, we focus on comparing the performance of cipher tests shown in Figure 15. It can be observed that the ASBUS cipher tests achieve higher performance than the AXI tests. First of all, the clock cycles spent by ASBUS cipher tests are 76.74%, 64.29%, and 51.22%, respectively, compared with AXI cipher tests for 32-, 64-, and 128-bit bus sizes.

Second, the dynamic energy consumed by the ASBUS cipher tests are 71.27%, 60.17%, and 48.38% compared with the AXI cipher tests for the 32-, 64-, and 128-bit buses, respectively, although the dynamic power of ASBUS cipher tests and AXI cipher tests are very close to each other.

Third, the conventional bandwidth and valid data bandwidth of the ASBUS cipher transfers can reach 2.95 GBps and 1.52 GBps, respectively, on the 128-bit ASBUS. The



(a) Linear



(b) Block

Fig. 14. Performance Comparison.

ADAM/XDAM valid data bandwidth ratios are 1.30, 1.56, and 1.95, respectively, when the bus size is 32, 64, and 128 bits.

Finally, we consider the slice efficiency and dynamic energy efficiency of all the AXI and ASBUS tests. The 128-bit ASBUS cipher test can transfer 89.32 Kbytes per second per slice cost. As the highest slice efficiency of all the cipher tests, it is 2.10 times compared with the 128-bit AXI Cipher test.

Additionally, the dynamic energy efficiency of the ASBUS cipher tests are 1.40, 1.66, and 2.07 times compared with the AXI cipher tests for the 32-, 64-, and 128-bit buses, respectively. In the other words, ASBUS can transfer 2.07 times as much data as AXI with the same time and power consumption when bus sizes are 128 bits.

In conclusion, the AES system performance of ASBUS surpasses that of AXI due to the high-efficiency state transfer mode. Moreover, the 128-bit implementations cost

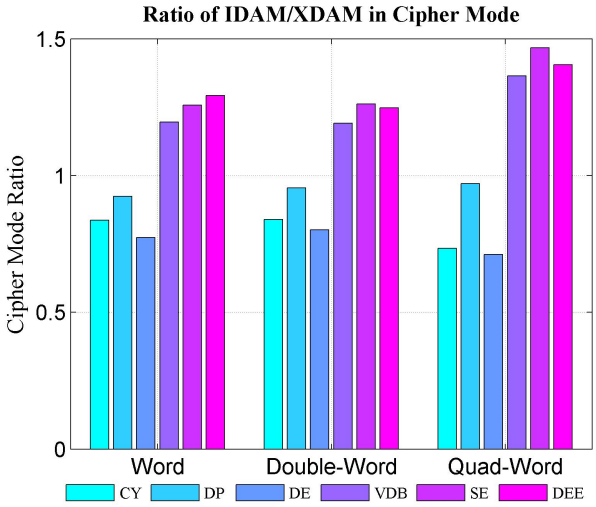


Fig. 15. Cipher Test Performance Comparison.

more IOs and dynamic power, but achieve higher slice efficiency and dynamic energy efficiency than 32- and 64-bit buses. Considering the design requirements and resource limitation, designers can choose different bus sizes based implementations.

8. SUMMARY

In this paper, we propose an advanced SBUS architecture for the AES-encrypted SoCs. To the best of our knowledge, it is the first performance analysis for AES circuits from the bus protocol perspective. As the results, the ASBUS based designs cost less in terms of hardware resource than the AXI based implementations, and the ASBUS cipher tests achieve higher valid bandwidth and consume less dynamic power than AXI. Based on the ASBUS architecture, we also evaluate the performance on different bus sizes. To sum up, the 128-bit design achieves higher valid bandwidth, but consumes more dynamic power than the 32- and 64-bit designs. In contrast, the 32-bit design consumes the least power but sacrifices bandwidth. Based on the resource and performance requirements, users can choose the proposed ASBUS implementations to fulfill the tradeoffs of different applications.

With the emerging area of Internet-connected tiny-size chips, leveraging limited computing resource and overhead cost for complex security mechanisms at the SoC level is a challenging issue. By optimizing the data-feeding efficiency using a novel bus, we believe that our work provides a solution for the AES-encrypted circuits to meet high-security and high-performance chip requirements.

REFERENCES

1999. AMBA Specification. Axis, Sunnyvale, CA, USA.

1999. CoreConnect Bus Architecture. IBM, Yorktown Heights, New York, NY, USA.

1999. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. IEEE Standard 802.11-1999.

2001. Open Core Protocol Specification. OCP Int. Partnership, Beaverton, OR, USA.

2003. AMBA AXI Protocol Specification. Axis, Sunnyvale, CA, USA.

2003. Wishbone BUS. Silicore Corp., Corcoran, MN, USA.

2004. STBus Interconnect. STMicroelectronics, Geneva, Switzerland.

January 2012. Xilinx, Virtex-6 Family Overview.

June 2011. UVM 1.1 Reference Manual. Accellera, Tualatin, OR, USA.

May 2012. UVM 1.1 User Guide. Accellera, Tualatin, OR, USA.

November 2001. FIPS PUB 197, Advanced Encryption Standard (AES). National Institute of Standards and Technology, U.S. Department of Commerce.

Sep, 2001. IEEE Standard Verilog Hardware Description Language. The Institute of Electrical and Electronics Engineers, Inc., 3 Park Ave., NY, USA.

K. Takano A. Satoh, S. Morioka and S. Munetoh. December 2000. A Compact Rijndael Hardware Architecture with S-Box Op-timization. *in Proc. ASIACRYPT* (December 2000), 239–245.

M. Mozaffari-Kermaniand AReyhani-Masoleh. August 2012. Efficient and High-Performance Parallel Hardware Architec-tures for the AES-GCM. *IEEE Trans. Comput.* 61, 8 (August 2012), 1165–1178.

C. Lin Chuang C. Hsing Wang and C. Wen Wu. April 2010. An Efficient Multimode Multiplier Supporting AES and Fun-damental Operations of Public-Key Cryptosystems. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 18, 4 (April 2010), 553–563.

D. Canright. 2005. A Very Compact Rijnael S-Box. (2005).

V. Fischer and M. Drutarovsky. May 2001. Two methods of Rijndael implementation in reconfigurable hardware. *in Proc. CHES 2001* (May 2001), 77C92.

R. C. Gonzalez and R. E. Woods. Digital Image Processing. Englewood Cliffs, NJ, USA: Prentice-Hall, 68C99.

T. Good and M. Benaissa. December 2012. 692-nW Advanced Encryption Standard (AES) on a 0.13-um CMOS. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 18, 12 (December 2012), 1753–1757.

A. Hodjat and I. Verbauwhede. April 2006. Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES processors. *IEEE Trans. Comput.* 55, 4 (April 2006), 366–372.

E. Oswald J. Wolkerstorfer and M. Lamberger. December 2000. An ASIC implementation of the AES S-boxes. *in proc. ASICRYPT* (December 2000), 239–245.

O. A. Mohamed K. Stevens. 2005. Single-Chip FPGA Implementation of a Pipelined, Memory-Based AES. *Canadian Conference on Electrical and Computer Engineering* (2005), 1296–1299.

A. K. Nandi M. M. Wong, M. L. D. Wong and I. Hijazin. Jun. 2012. Construction of Optimum Composite Field Architecture for Compact High-Throughput AES S-Boxes. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 20, 6 (Jun. 2012), 1151–1155.

M. McLoone and J. V. McCanny. September 2001. Rijndael FPGA implementation utilizing look-up tables. *IEEE Workshop on Signal Processing Systems* (September 2001), 349C360.

E. NC Mui. 2007. Practical Implementation of Rijndael S-Box Using Combina-tional Logic. (2007).

B. Preneeland N. Mentens, L. Batinan and I. Verbauwhede. 2005. A Systematic Evaluation of Compact Hardware Implementa-tion for the Rijndael S-Box. *in Proc. Topics Cryptology (CT-RSA)* 3376 (2005), 323–333. DOI:http://dx.doi.org/10.1007/978-3-540-30574-3_22

C. Paar. 1994. Efficient VLSI architecture for bit-parallel computations in Galois field. *Ph.D. dissertation, Institute for Experimental Mathematics* (1994).

V. Rijmen. 2000. Efficient Implementation of the Rijndael S-box. (2000). http://ftp.comms.scitech.susx.ac.uk/fft/crypto/rijndael-sbox.pdf

M. Chih Chen S. Fu Hsiao and C. Shin Tu. March 2006. Memory-Free Low-Cost Designs of Advanced Encryption Standard Using Common Subexpression Elimination for Sub-functions in Transformations. *IEEE Trans. Circuits Syst. I, Reg. Papers* 53, 3 (March 2006), 615 – 626.

N. Sklavos and O. Koufopavlou. December 2012. Architectures and VLSI Implementations of the AES-Proposal Rijndael. *IEEE Trans. Comput.* 51, 12 (December 2012), 1454–1459.

W. Suntiamorntut and W. Wittayapanpracha. March 2012. The Study of AES Encryption for Wireless FPGA Node. *International Journal of Communications in Information Sci-ence and Management Engineering* 12, 3 (March 2012), 40–46.

Y. Wang and Y Ha. January 2013. FPGA-Based 40.9-Gbits/s Masked AES with Area Optimiza-tion for Storage Area Network. *IEEE Trans. Circuits Syst. II. Exp. Briefs* 60, 1 (January 2013), 36–40.

X. Yang and J. Andrian. July 2014. A High Performance On-Chip Bus (MSBUS) Design and Verification. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (TVLSI)* 23, 7 (July 2014), 1350–1354.

X. Yang and W. Wen. in press 2017. Design of A Pre-Scheduled Data Bus (DBUS) for Advanced Encryption Standard (AES) Encrypted System-on-Chips (SoCs). The 22nd Asia and South Pacific Design Automation Conference. (ASP-DAC 2017).

X. Zhang and K.K. Parhi. October 2006. On the optimum constructions of composite field for the AES algorithm. *IEEE Trans. Circuits Syst. II. Exp. Briefs* 53, 10 (October 2006), 1153–1157.

X. Zhang and K.K. Parhi. September 2004. High-Speed VLSI Architecture for the AES Algorithm. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 12, 9 (September 2004), 957–967.