Atomic-AES v2.0

Subhadeep Banik[1], Andrey Bogdanov[2] and Francesco Regazzoni[3]

[1] Temasek Labs, Nanyang Technological University, Singapore
bsubhadeep@ntu.edu.sg
[2] DTU Compute, Technical University of Denmark, Lyngby
anbog@dtu.dk
[3] ALARI, University of Lugano
regazzoni@alari.ch

Abstract. Very recently, the Atomic AES architecture that provides dual functionality of the AES encryption and decryption module was proposed. It was surprisingly compact and occupied only around 2605 GE of silicon area and took 226 cycles for both the encryption and decryption operations. In this work we further optimize the above architecture to provide the dual encryption/decryption functionality in only 2060 GE and latency of 246/326 cycles for the encryption and decryption operations respectively. We take advantage of clock gating techniques to achieve Shiftrow and Inverse Shiftrow operations in 3 cycles instead of 1. This helps us replace many of the scan flip-flops in the design with ordinary flip-flops. Furthermore we take advantage of the fact that the Inverse Mixcolumn matrix in AES is the cube of the forward Mixcolumn matrix. Thus by executing the forward Mixcolumn operation three times over the state, one can achieve the functionality of Inverse Mixcolumn. This saves some more gate area as one is no longer required to have a combined implementation of the Forward and Inverse Mixcolumn circuit.

Keywords: AES 128, Serialized Implementation.

1 Introduction

The Atomic-AES architecture for combined implementation of the AES Encryption/Decryption circuit was proposed very recently [2]. The circuit builds on the encryption only circuit proposed by Moradi et al. [26] at Eurocrypt 2011, and adds subtle tweaks to the circuit to enable combined functionality of encryption and decryption. The Atomic-AES architecture occupies around 2605 GE of area when synthesized with the standard cell library of STM 90nm CMOS logic process and takes 226 cycles for both encryption and decryption operations.

In this work, we propose Atomic-AES v2.0 architecture that at 2060 GE, occupies around 400 GE less using the same standard cell library. The architecture has encryption/decryption latency of 246/326 cycles. Each encryption round is computed in 23 cycles, each decryption round takes 31 cycles. The area savings comes from principally two avenues:

1. The Shiftrow/Inverse Shiftrow operations are performed over three cycles rather than 1. This helps the designer replace a lot of the scan flip-flops in the design with ordinary flip-flops, which on average saves 1 GE of area per bit of storage.
2. Additionally the Inverse Mixcolumn matrix used in AES is the cube of the the forward Mixcolumn matrix. This implies that executing the Mixcolumn operation 3 times over the state achieves the functionality of Inverse Mixcolumn. Thus the designer no longer needs a combined implementation of the Forward and Inverse Mixcolumn Circuit.

1.1 Organization

The paper is organized in the following manner. Section 2 gives a brief background and description of the architecture presented in [2]. Section 3, describes some of the modifications in Atomic-AES v2.0. Section 4 tabulates all implementation results and compares it with previous architectures present in literature. Section 5 concludes the paper.

2 Background and Preliminaries

In Figure 1, a pictorial description of the architecture in [2] is given. As can be seen the basic elements of storage are the 16 byte sized registers made of scan flip-flops in the state and key path respectively, used to store the intermediate states and roundkeys. Each round function is calculated in 21 cycles.

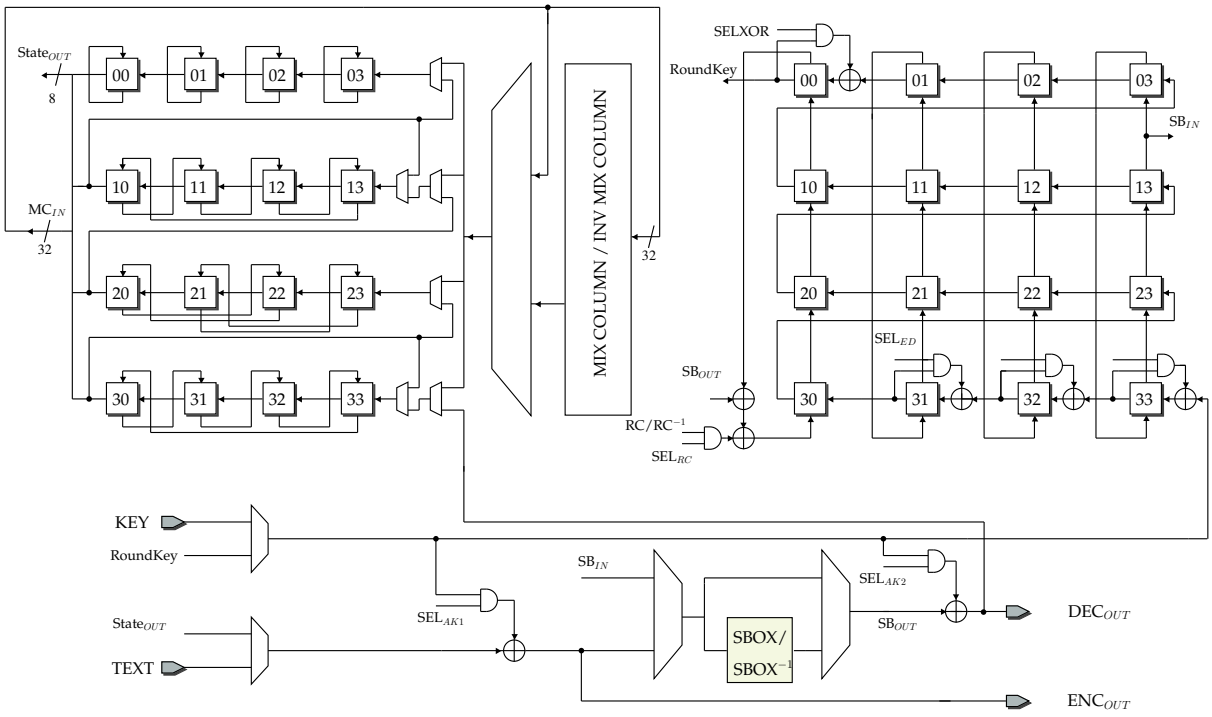

Fig. 1: The 8-bit serial architecture in [2]

2.1 Encryption Flow

We will label the 21 cycles per round by the integers 0 to 20. The encryption process starts with the addition of the whitening key and the S-box computation of the first round function. In order to do so the finite sate machine (FSM) generating the round signals is initialized to cycle number 5. So in cycles numbered 5 to 20 (i.e. the very first 16 cycles) the following transformations take place:

Cycles 5 to 20: The 8 bit chunks of plaintext and key are respectively filtered out of the main state and key multiplexers respectively. They are xored, and the resultant signal fed to the S-box. The output of the S-box is fed to the bottom most multiplexer in the state path (marked by SB_{IN}), from where it is shifted serially forward in the next round. Effectively, after the cycle 20 is completed, the state registers would store the value $S(PT \oplus K)$, where $S(\cdot)$ denotes the bytewise application of the AES S-box function. In the same period the 8 bit chunk of the Key is input to key register marked "33", from where it is serially forwarded in the next round, much like in the state register. Therefore, at the end of cycle 20, the Key registers hold the value of the initial whitening key.

After this the cycle counter is automatically reset to 0, and each 21 cycle round function is executed 10 times, thus accounting for a total latency of $16 + 21 * 10 = 226$. During this period the order of operations is as follows:

$$\text{Shiftrow} \quad \rightarrow \quad \text{Mixcolumn} \quad \rightarrow \quad \text{Add roundkey + S-box of next round}$$

To clarify, let us see the cyclewise description of the data movement:

Cycle 0: This cycle is reserved for the Shiftrow operation. Since each 8-bit register in the state and key paths are constructed using scan flip-flops, they have two input data ports which they filter depending on a select signal. As can be seen in Figure 1, the state registers are connected to facilitate the Shiftrow operation during cycle 0. The key register is "frozen" in this cycle and so no data movement takes place. [4]

Cycles 1 to 4: The Mixcolumn operation is performed during these 4 cycles. The Mixcolumn circuit used in this architecture is a $\{0,1\}^{32} \rightarrow \{0,1\}^{32}$ logic block, and so data from leftmost column (registers marked 00,10,20,30) of the state is fed as input to the Mixcolumn circuit. In the subsequent cycle the Mixcolumn

[4] One way to achieve this is to use a gated clock which does not present a leading edge during the shiftrow period.

output is driven into the rightmost column (registers marked 03,13,23,33). This operation carried out over 4 cycles computes the Mixcolumn over the entire state. Note that this operation is bypassed in the 10th encryption round as the Mixcolumn function is omitted in the final round.

During this period, the non-linear function of the Keyschedule operation is computed in the Key registers. Recall that the non linear operation in the AES Keyschedule is given as

$$F(K_3) = S(K_3 \lll 8) \oplus RCON_i,$$

where K_3 denotes the third column of the current roundkey, $\lll$ denotes the left rotate operation and $RCON_i$ is the i^{th} round constant (note that the round constant is added to the most significant byte of $S(K_3 \lll 8)$). $(K_3 \lll 8)$ is a 32 bit value and so $S(K_3 \lll 8)$ implies the S-box function applied to each of the 4 bytes of the input. In order to implement the rotation operation, the data is taken from the output of the key register marked "13" and fed to the S-box. Although the architecture uses only one S-box, in cycles 1 to 4, the state path operations do not use the S-box circuit and so the key path S-box operations can be done in this period. The S-box output is xored to the output of the register "00" and the round constant and, in the next cycle is driven into the register marked "30". Note that since there is "vertical" movement of data in the key registers in this period, at the end of cycle 4, the four columns of the key register store the values $K_0 \oplus F(K_3), K_1, K_2, K_3$ respectively, where K_i denotes the i^{th} column of the current roundkey.

Cycles 5 to 20: The bytes of state and roundkey are respectively taken out of the registers marked "00" of both the state and key paths and xored together and fed to the S-box. The output of the S-box is again driven into the bottom most state register "33" and serially shifted forward in the subsequent rounds. This sequence of operations is exactly similar as the ones performed in the very first 16 cycles, with the only exception that an intermediate state and roundkey chunks are xored instead of the raw plaintext and key.

The operations in the Key register are a little more interesting during this period. Note that in order to perform roundkey addition during these cycles, the data emanating from key register "00" be equal to the current roundkey. However we have seen that at the end of cycle 4 the columns of the key registers hold the value $K_0 \oplus F(K_3), K_1, K_2, K_3$. Note that if K_0, K_1, K_2, K_3 and L_0, L_1, L_2, L_3 denote the 4 columns of the current and next roundkey then we have

$$L_0 = K_0 \oplus F(K_3), \quad L_1 = K_1 \oplus L_0, \quad L_2 = K_2 \oplus L_1, \quad L_3 = K_3 \oplus L_2.$$

Thus at the end of cycle 4, only the 0^{th} column holds the correct next roundkey L_0. The problem is solved by having an extra xor gate taking inputs from the registers "00" and "01" and output feeding into "00". Since the movement of data is switched to "horizontal", this helps to perform on the fly addition as the key chunks are driven out of the "00" register. The addition is however not executed at cycles 8,12,16,20 by zeroing the SELXOR signal because as previously noted, the 0^{th} column already has the required roundkey. Also after the roundkey addition, each 8-bit key is circularly shifted back into the key registers through register "33" in order to facilitate the operations in the next round function.

The i^{th} round in this architecture computes the Substitution layer for the $(i+1)^{th}$ AES encryption round. This being so, in the tenth and final encryption round the only operations that need be performed are Shiftrows and the final roundkey addition. Thus in the tenth round, the Mixcolumn operation is bypassed in cycles 1-4 and the output ciphertext is available just after the roundkey addition from cycles 5 through 20.

2.2 Decryption Flow

On the state side the connections between the byte sized scan registers are configured so as to support Shiftrow, Inverse Shiftrow and Serial loading and unloading of bytes. The following sequence of operations is used for Decryption:

$$\text{Inverse Mixcolumn} \quad \rightarrow \quad \text{Inverse Shiftrow} \quad \rightarrow \quad \text{Inverse S-box + Add roundkey}$$

The decryption starts with the addition of the whitening key. The finite sate machine (FSM) generating the round signals is again initialized to cycle number 5. So in cycles numbered 5 to 20 (i.e. the very first 16 cycles) the following transformations take place:

Cycles 5 to 20: The 8 bit chunks of ciphertext and key are respectively filtered out of the main state and key multiplexers respectively They are xored, and the resultant signal fed to the state registers. Note that in the corresponding encryption stage, we additionally calculated the S-box of the first round. Hence in order to accommodate both encryption and decryption we need a multiplexer after the S-box circuit as shown in Figure 1. The Key bytes are input to key register "33", from where it is serially forwarded in the next round. However the SEL_{ED} signal is set to 1 at rounds 8, 12, 16, 20 due to which at beginning of the next phase, the Key four register columns hold the value L_0, K_1, K_2, K_3 respectively.

After this the cycle counter is automatically reset to 0, and each 21 cycle round function is executed 10 times. Since the data flow in the key registers have already explained in the previous subsection, we concentrate on the state register.

Cycles 0 to 4: Cycles 0-3 perform the Inverse Mixcolumn operation on the state columns, in exactly the same way forward Mixcolumn is executed in the encryption stage in cycles 1 to 4. However only in the very first round the Inverse Mixcolumn operation is bypassed, as required in AES decryption. Cycle 4 is reserved for the Inverse Shiftrow operation.

In the Keyschedule, the key register is frozen in cycle 0. Thereafter in cycles 1 to 4, $F(K_3)$ is computed in the same manner as described in the encryption cycles and added to L_0 in the first column. And as a result at the beginning of cycle 5, the key columns contain $K_0 = L_0 \oplus F(K_3), K_1, K_2, K_3$ which is the complete next roundkey. Since the complete roundkey is already available, the SELXOR signal controlling the xor gate in the topmost row is zeroed as the roundkeys are serially driven out for the add roundkey operation. Thus all the functionalities of Inverse Keyschedule are completely accommodated using this architecture. Furthermore the complete decryption roundkey is available from cycles 5 through 20, which is incidentally the period during which we perform the add roundkey operation.

Cycles 5 to 20: The bytes of state are taken out from register "00" and input into the combined forward and reverse S-box circuit to compute the Inverse S-box operation. The output of the S-box is then xored with the current roundkey byte from the key register "00" and circulated serially back into the state registers via the register marked "33". Note that the order of S-box and Add roundkey in the decryption phase is exactly the opposite as the encryption phase. As a result we employ two 8-bit xor gates, one before and one after the S-box circuit, for key addition in the encryption and decryption stages respectively. The xor gate inputs are controlled by and gates as shown in Figure 1, in order to bypass the addition operation as required.

In the tenth and final round, the decrypted plaintext is made available from cycles 5 through 20 after the add roundkey operation.

3 Atomic-AES v2.0: Architecture and Dataflow

We will now present a full description of the proposed architecture for Atomic-AES v2.0 which provides dual functionalities for encryption and decryption. A diagram for the proposed architecture is presented in Figure 2.

3.1 Main changes

There are 2 main changes due to which which it was possible to reduce the area. they are listed as follows:

1. **Replacing scan flip-flops with ordinary flip-flops:** One of the reasons why scan flip-flops were used for implementing both the state and key registers was that these storage units needed to support multiple modes of operation, in which each byte sized register needed to accept data from multiple sources. The state registers need to support serially loading and unloading data as well as the Shiftrow and Inverse Shiftrow operations. The key registers support 2 types of data movement: horizontal and vertical. the horizontal is meant for serial loading/unloading data, while the vertical is used to efficiently compute the nonlinear function F used in the Keyschedule.

   To begin the optimization let us start with the Keyschedule. The vertical movement of data used to compute the F function is required only in the outermost columns of the key registers, i.e. columns 0 and 3. It is

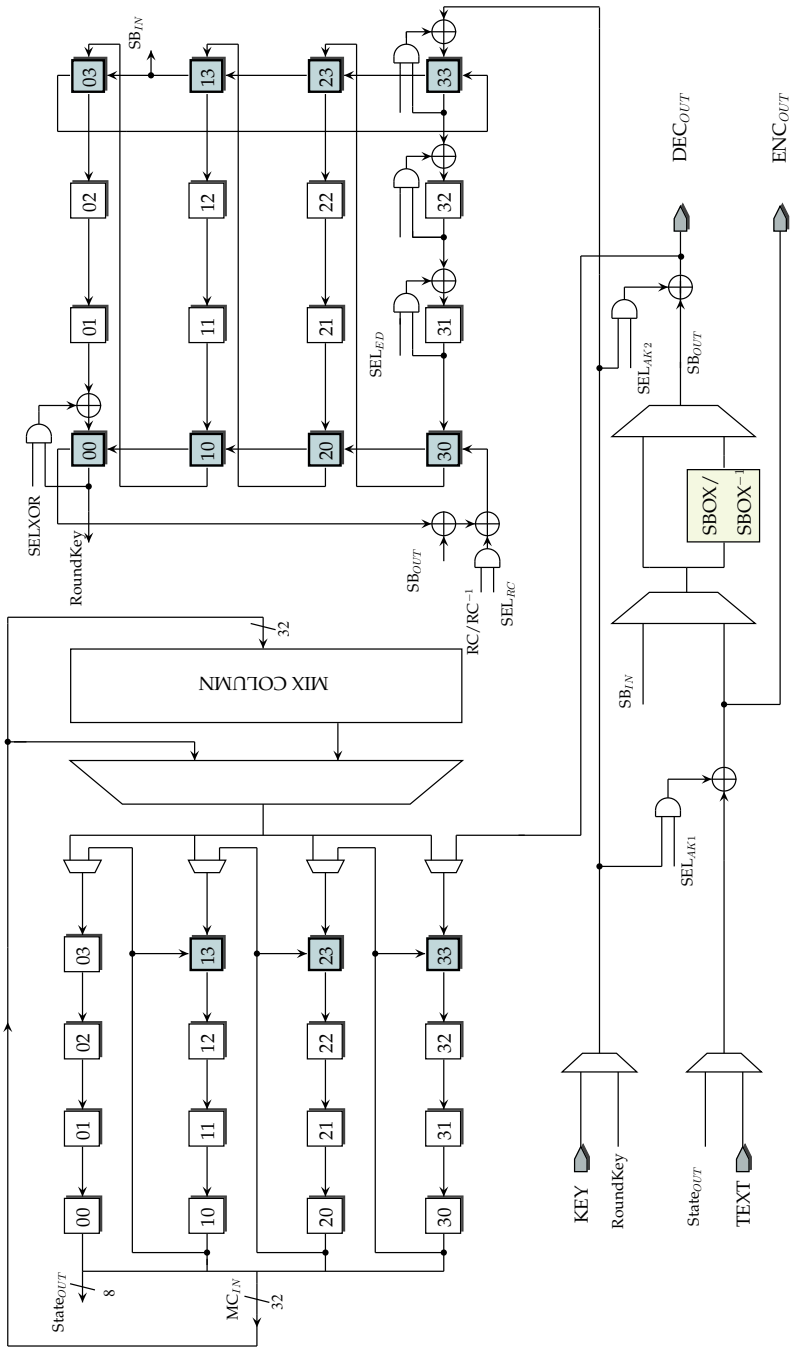

Fig. 2: The AES 8 bit Encryption/Decryption architecture for Atomic-AES v2.0

actually not required in the two middle columns 1 and 2. It is therefore possible to implement the middle-most columns with ordinary rather than scan flip-flops. Of course this requires that the data movement in the middle columns be frozen when the function F is being calculated. This can be easily achieved using clock gating techniques.

In the state registers, we argue that that scan flip-flops are required to implement only the byte registers "13", "23" and "33". These are the byte registers in the final column of rows 1, 2, 3 respectively. Scan flip-flops are not required for "03" because the zeroth row does not require data movement during the Shiftrow or Inverse Shiftrow operations. For either the Shiftrow or Inverse Shiftrow operations, there is a maximum movement of three columns to the left for any row. Indeed, except the zeroth row which does not require data movement, any row which has a movement of x columns towards the left for Shiftrow would undergo a movement of $4 - x$ columns towards the left for Inverse Shiftrows. Thus if the designer is prepared to allow 3 clock cycles for the Shiftrow/Inverse Shiftrow operation then both operations can be achieved by single directional data movement towards the left. This is precisely why, the remaining byte registers can be implemented with ordinary flip-flops. However, the designer has to take help of clock gating to freeze data movement in certain rows during the row-wise shifting operation. This has been tabulated in Table 1. As we will see shortly, during the Encryption flow, the Shiftrow is executed in cycles labelled 0, 1, 2 and during Decryption the Inverse Shiftrow operation is executed in cycles 12, 13, 14. Figure 2, gives a complete picture of the architecture. Registers implemented using scan flip-flops are labeled in grey. Except for 3 registers in the state and 8 in the key, all can be implemented using ordinary flip-flops. This saves us around 400 GE for implementing the storage signals alone.

		Shiftrow Cycles			**Inverse Shiftrow Cycles**		
#	Row	0	1	2	12	13	14
1	**0**	F	F	F	F	F	F
2	**1**	F	F	O	O	O	O
3	**2**	F	O	O	F	O	O
4	**3**	O	O	O	F	F	O

Table 1: Data flow in the rows of state registers during Shiftrow/Inverse Shiftrow. (F: Frozen, O: Operational)

2. **Replacing combined Mixcolumn with forward Mixcolumn:** In [2], the designers had implemented a combined Mixcolumn/Inverse Mixcolumn circuit which took 166 xor gates and a 32 bit multiplexer. Since the circuit operated on a column every clock cycle, a total of 4 cycles were required to compute the Mixcolumn over the entire state. In this work, we take advantage of the fact that the Inverse Mixcolumn matrix used in AES is the cube of the Forward Mixcolumn matrix, i.e.

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}^3$$

This directly implies that if the designer runs the forward Mixcolumn operation 3 times over the state i.e. for $3 \times 4 = 12$ cycles, he would functionally achieve the Inverse Mixcolumn operation. This in turn means that a forward Mixcolumn circuit which occupies 108 xor gates is sufficient for both purposes. This saves us area equal to 58 xor gates and one 32 bit multiplexer, which amounts to around 130-140 GE.

3.2 Encryption Flow

The encryption flow is almost the same as the one used in [2] and which has been described in brief in the previous section and it maintains exactly the same order of operations. There are subtle differences however. Since Shiftrow is executed over 3 cycles rather than 1, one encryption round is carried out over 23 cycles rather than 21. The circuit uses a maximum length 5 bit LFSR to generate control signals, which has a period of 31 cycles which we label as 0 to 30. In the beginning the control system is initialized to cycle 15.

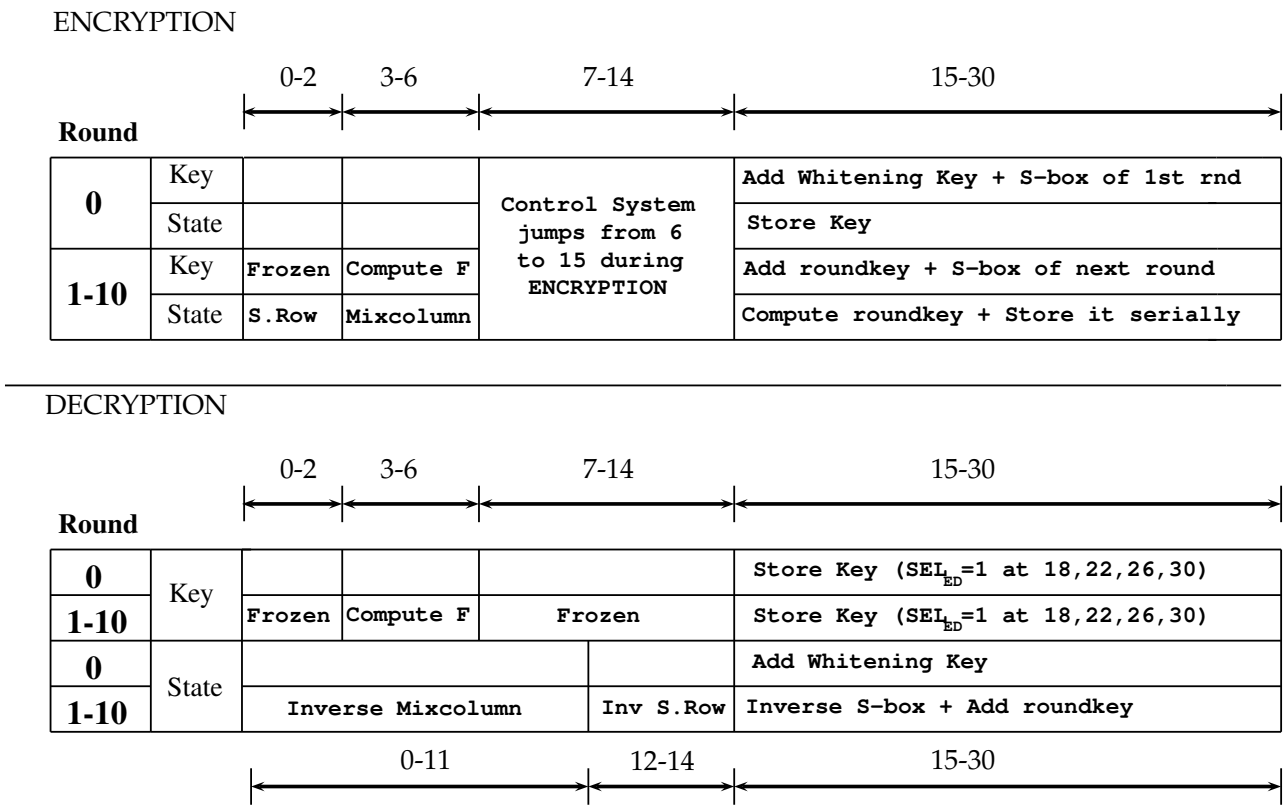

Fig. 3: Operation sequences in the Encryption/Decryption stages

Cycles 15 to 30: As before the initial $S(PT \oplus K)$ operation is performed and the result is stored serially in the state registers and the key bytes are stored serially in the the key registers.

Thereafter the counter is reset to 0, and the 10 encryption rounds are executed one after the other. Each round consists of the following ordered sequence of operations:

Cycles 0 to 2: The state registers execute Shiftrow, and the Key registers are frozen.

Cycles 3 to 6: The state registers execute Mixcolumn, and the outermost columns of the Key register compute the function F as explained in the previous section. During encryption, the control system transitions from cycle 6 to cycle 15, so that cycles 7 to 14 do not occur.

Cycles 15 to 30: Exactly as in the previous section, the bytes are driven serially out of "00" from both the state and key side, the add roundkey and substitution layer of the next round are performed and the resultant signal, and the key bytes are driven serially back into the state/key registers respectively.

The encryption function thus takes $23 \times 10 + 16 = 246$ cycles to complete.

3.3 Decryption Flow

The decryption flow is also almost the same as the one used in [2] and exactly the same order of operations is maintained. The main differences are that Inverse Mixcolumn is executed over 12 cycles and and Inverse Shiftrow over 3 cycles. Thus one decryption round takes 31 cycles to complete. As before, in the beginning the control system is initialized to cycle 15.

Cycles 15 to 30: As before the initial whitening key addition i.e. $CT \oplus K$ operation is performed and the result is stored serially in the state registers and the key bytes are stored serially in the the key registers. As explained in the previous section, the SEL_{ED} signal is set to 1 in cycles 18, 22, 26, 30 to enable efficient backward generation of the roundkeys.

Thereafter the counter is reset to 0, and the 10 decryption rounds are executed one after the other. Each round consists of the following ordered sequence of operations:

Cycles 0 to 14: The state registers execute Inverse Mixcolumn during 0 to 11, and Inverse Shiftrow during 12 to 14. The key registers are frozen during 0 to 2 and again from 7 to 14. In the 4 cycles in between, (i.e. during 3 to 6) the non-linear function F is computed exactly as explained in the previous section.

Cycles 15 to 30: Again, the bytes are driven serially out of "00" from both the state and key side, the inverse S-box is applied on the state bytes after which the add roundkey is performed and the resultant signal, and the key bytes are driven serially back into the state/key registers respectively. The SEL_{ED} signal is again set to 1 in cycles 18, 22, 26, 30 to enable efficient backward generation of the next roundkey.

The decryption function thus takes $31 \times 10 + 16 = 326$ cycles to complete. The flow has also been explained diagrammatically in Fig 3.

3.4 Control System

All control signals are generated using a maximal length 31 cycle LFSR. Some additional logic is used to sense the clock cycle 6 in the encryption cycle and transition to cycle 15.

4 Performance Evaluation

In order to perform a fair performance evaluation, we implemented the circuit using VHDL[5]. Thereafter the following design flow was adhered to for all the circuits: a functional verification at the RTL level was first done using Mentor Graphics Modelsim software. The designs were synthesized using the standard cell library of the 90nm and 65nm logic process of STM (CORE90GPHVT v 2.1.a and CORE65LPLVT v 5.1) with the Synopsys Design Compiler, with the compiler being specifically instructed to optimize the circuit for area. A timing simulation was done on the synthesized netlist to confirm the correctness of the design, by comparing the output of the timing simulation with known test vectors. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity. The results are tabulated in Table 2.

We outline some of the essential lightweight metrics of the known implementations of encryption/decryption architectures of AES and compare it with our own. Energy consumption was listed rather than power as it is a measure of the total electrical work done during one encryption/decryption. Since the circuits in Table 2 are implemented using different CMOS logic processes, there are most likely to be wide variations in energy consumption and maximum throughput. For example the throughput of [24] is quite high as it is implemented using the standard cell library of the 22nm CMOS logic process which is faster than the other logic processes listed in the table. The throughput of [27] is also high as it is a 32-bit serial circuit and thus has considerably lower latency.

#	Architecture	Type	Library	Area (GE)	Latency (cycles)	Energy (nJ)	TP_{max} (Mbps)
1	8-bit Serial [26]	E	UMC 180nm	2400	226	8.4	-
2	Grain of Sand [17]	ED	Philips 350nm	3400	1032/1165	46.4/52.4	9.9/8.8
3	8-bit Serial [24]	ED	22nm	4037	336/216	3.9/2.5	432/671
4	32-bit Serial [27]	ED	110nm	5400	54/54	-	311
5	Atomic-AES [2]	ED	STM 90nm	2605	226/226	3.3	94.4
			STM 65nm	2931		2.2	57.8
6	Atomic-AES v2.0	ED	STM 90nm	**2060**	246/326	3.2/4.2	88.4/66.7
			STM 65nm	2430		1.9/2.5	54.4/41.1

Table 2: Performance Comparison of Atomic-AES with previous architectures in literature (Figures separated by '/' indicate corresponding figures for encryption/decryption, E: Encryption only, ED: ENC/DEC)

[5]Source codes can be found at 1. Atomic AES: `https://dl.dropboxusercontent.com/u/79538921/atomicaes.zip`
2. Atomic AES v2.0: `https://dl.dropboxusercontent.com/u/79538921/aav2.zip`

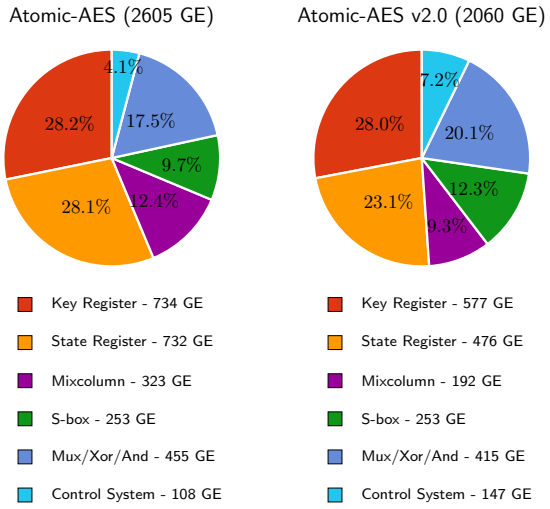

Fig. 4: Area requirements of the individual components for Atomic AES and Atomic AES v2.0

In Figure 4, we present a componentwise breakdown of the circuit size. We use clock gating to generate the clock for the Key registers, since the data movement has to be frozen for one cycle. Apart from the multiplexers included in the implementation of the combined Forward and Inverse S-box, Mixcolumn and Round Constants, a quick glance at Figure 2, tells us that we need

1. Four 8-bit multiplexers around the state register, one 32-bit multiplexer to bypass the Mixcolumn circuit, one 8-bit multiplexer after the S-box, and two 8-bit multiplexers to filter the raw key/plaintext (ciphertext) and the roundkey/state byte respectively.
2. Apart from this six 8-bit xors around the key registers and two 8-bit xors during state-key addition.
3. One input of seven out of the eight xor gates is controlled by an and gate.

This adds up to around 415 GE for the multiplexers, xor, and gates in the circuit. The LSFR based control system, the round constants and the logic for clock-gating take around 147 GE. Adding up, this leads to 2060 GE for the entire circuit.

5 Conclusion

In this work, we present a compact architecture for AES that performs the dual function of encryption and decryption. the circuit can be synthesized using 2060 GE area using the standard cell library of the STM 90nm CMOS logic process. This is an improvement over the work in [2] which occupied 2605 GE using the same standard cell library.

Acknowledgement: The authors would like to sincerely thank Thomas Peyrin and Meicheng Liu for valuable discussions leading up to this work.

References

1. E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser, K. Yasuda. AES-COPA v.1. Submission to the Caesar Competition. Available at `http://competitions.cr.yp.to/round1/aescopav1.pdf`.
2. S. Banik, A. Bogdanov, F. Regazzoni. Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core. In IACR Cryptology ePrint Archive, 2016: 927 (2016).
3. S. Banik, A. Bogdanov, F. Regazzoni. Exploring Energy Efficiency of Lightweight Block Ciphers. In SAC 2015, LNCS, vol. 9566, pp. 178-194, 2015.
4. S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, F. Regazzoni. Midori: A Block Cipher for Low Energy. In ASIACRYPT 2015, LNCS, vol. 9453, pp. 411-436, 2015.

5. S. Banik, A. Bogdanov, F. Regazzoni, T. Isobe, H. Hiwatari, T. Akishita. Round gating for low energy block ciphers. In IEEE Hardware Oriented Security and Trust (HOST), pp. 55-60, 2016.
6. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers. The Simon and Speck Families of Lightweight Block Ciphers. In IACR eprint archive. Available at `https://eprint.iacr.org/2013/404.pdf`.
7. A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In CHES 2007, LNCS, vol. 4727, pp. 450-466, 2007.
8. J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knežević, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, T. Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Asiacrypt 2012, LNCS, vol. 7658, pages 208-225, 2012.
9. J. Boyar, P. Matthews, R. Peralta. Logic Minimization Techniques with Applications to Cryptology. In J. Cryptology, vol. 26, pp. 28–312, 2013.
10. P. Chodowiec, K. Gaj. Very Compact FPGA Implementation of the AES Algorithm. In CHES 2003, LNCS, vol. 2779, pp. 319-333, 2003.
11. C. De Cannière, O. Dunkelman, M. Knežević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In CHES 2009, LNCS, vol. 5747, pp. 272-288, 2009.
12. D. Canright. A very compact S-Box for AES. In CHES 2005, LNCS, vol. 3659, pp. 441-455, 2005.
13. J. Daemen, M. Peeters, G. V. Assche, V. Rijmen. Nessie Proposal: NOEKEON. Available at `http://gro.noekeon.org/Noekeon-spec.pdf`.
14. J. Daemen, V. Rijmen. The design of Rijndael: AES - the Advanced Encryption Standard. Springer-Verlag, 2002.
15. N. Datta and M. Nandi. ELmD v1.0. Submission to the Caesar compedition. Available at `https://competitions.cr.yp.to/round1/elmdv10.pdf`.
16. M. Dworkin. Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A. Available at `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`.
17. M. Feldhofer, J. Wolkerstorfer, V. Rijmen. AES Implementation on a Grain of Sand. In IEEE Proceedings of Information Security, vol. 152(1), pages 13-20, 2005.
18. Z. Gong, S. Nikova, Y.W. Law. KLEIN: a new family of lightweight block ciphers. In RFIDSec 2011, LNCS, vol. 7055, pp. 1-18, 2011.
19. J. Guo, T. Peyrin, A. Poschmann, M. J. B. Robshaw. The LED Block Cipher. In CHES 2011, LNCS, vol. 6917, pp. 326-341, 2011.
20. P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In DSD, pages 577-583, 2006.
21. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Ko, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, S. Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In CHES 2006, LNCS, vol. 4249, pp. 46-59, 2006.
22. S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, F. X. Standaert. Towards Green Cryptography: a Comparison of Lightweight Ciphers from the Energy Viewpoint. In CHES 2012, LNCS, vol. 7428, pp. 390-407, 2012.
23. A. Lutz, J. Treichler, F. Gürkaynak, H. Kaeslin, G. Basler, A. Erni, S. Reichmuth, P. Rommens, S. Oetiker, W. Fichtner. 2Gbit/s hardware realizations of RIJNDAEL and SERPENT: A comparative analysis. In CHES 2002, LNCS, vol. 2523, pp. 144158, 2002.
24. S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, R.K. Krishnamurthy. 340 mV–1.1V, 289 Gbps/W, 2090-gate nanoAES hardware accelerator with area-optimized encrypt/decrypt $GF(2^4)^2$ polynomials in 22 nm tri-gate CMOS. In IEEE Journal of Solid-State Circuits, vol. 50, pp. 1048–1058, 2015.
25. N. Mentens, L. Batina, B. Preneel and I. Verbauwhede. A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box. In CT-RSA 2005, LNCS, vol. 3376, pp. 323–333, 2005.
26. A. Moradi, A. Poschmann, S. Ling, C. Paar, H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Eurocrypt 2011, LNCS, vol. 6632, pp. 69-88, 2011.
27. A. Satoh, S. Morioka, K. Takano, S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In Asiacrypt 2001, LNCS, vol. 2248, pp. 239-254, 2001.
28. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In CHES 2011, LNCS, vol. 6917, pp. 342-357, 2011.
29. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, T. Iwata. The 128-bit Block-cipher CLEFIA(Extended Abstract). In FSE 2007, LNCS, vol. 4593, pp. 181-195, 2007.
30. T. Suzaki, K. Minematsu, S. Morioka, E. Kobayashi. TWINE: A Lightweight Block Cipher for Multiple Platforms. In SAC 2012, LNCS, vol. 7707, pp. 339-354, 2012.
31. R. Ueno, S. Morioka, N. Homma, T. Aoki. A High Throughput/Gate AES Hardware Architecture by Compressing Encryption and Decryption Datapaths - Toward Efficient CBC-Mode Implementation. In CHES 2016, LNCS, vol. 9813, pp. 538-558, 2016.
32. R. Ueno, N. Homma, Y. Sugawara, Y. Nogami, and T. Aoki. Highly Efficient $GF(2^8)$ Inversion Circuit Based on Redundant GF Arithmetic and Its Application to AES Design In CHES 2015, LNCS, vol. 9293, pp. 63–80, 2015.