An Improved Clocking Methodology for Energy Efficient Low Area AES Architectures using Register Renaming

Siva Nishok Dhanuskodi and Daniel Holcomb

Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA
sdhanusk@umass.edu, holcomb@engin.umass.edu

***Abstract*—Sub-round implementations of AES have been explored as an area and energy efficient solution to encrypt data in resource constrained applications such as the Internet of Things. Symmetry in AES operations across bytes and words allows the datapath to be scaled down to 8 bits resulting in very compact designs. However, such designs incur an area penalty to store intermediate results or energy penalty to shift data through registers without performing useful computation. We propose a smart clocking scheme and rename registers to minimize data movement and clock loading, and also avoid storing a duplicate copy of the system state. In comparison to the most efficient 8-bit implementation from literature, we save 45% energy per encryption and reduce clock energy by 70% at a reasonable area cost.**

I. Introduction

Efficiency of block cipher implementations is a significant concern in low power secure systems. Most such systems use a block cipher such as AES as a building block for encryption and decryption modes or even hashing. In the era of IoT and ubiquitous computing, the collection and communication of sensitive data is increasingly being handled by lightweight devices. Furthermore, even in larger SoC systems, the required encryption throughput is often low, and therefore lightweight solutions may be preferred to high-performance ones.

AES (Advanced Encryption Standard) is the dominant block cipher in use today, and implementations of AES have received significant attention in literature, from high performance to low power. Because the algorithm is fully specified and not especially flexible, most existing designs tend to utilize similar microarchitectures, with clever circuit-level or component-level power optimizations.

In this work, we give a general technique that modifies the microarchitecture of sub-round AES implementations using register renaming. Our work notably avoids two inefficiencies of existing 8-bit architectures. Firstly, it avoids using additional state registers for temporary storage of round outputs during the round computation. Secondly, it conserves clock and data power by clocking and moving data around only when needed by the algorithm.

The significant contributions of this paper are as follows:

- Presenting a microarchitectural technique for register renaming that allows complexity to be shifted from the datapath to the control logic where its cost is lower.
- Demonstrating in technology-independent terms that, relative to the best 8-bit implementation from literature, the proposed technique improves energy-per-operation, power, and clock switching, at a modest area cost.

II. Background and Related Work

A. The AES Algorithm

The Advanced Encryption Standard (AES) is a ubiquitous block cipher. We review here a few relevant details of AES and refer interested readers to its documentation for more depth [11]. The AES algorithm uses a 128-bit block size and, for the 128-bit key size variant, performs 10 iterations of its round function. Each round operates on 128-bit state with a 128-bit round key to produce 128-bit state that serves as the input for the next round. The round function (Fig. 1) comprises a SubBytes operation where the same byte-wise substitution function is applied to each of the 16 bytes, then ShiftRows and MixColumns operations combine data from different 4-byte words, and finally the round key is added to the output of MixColumns to create the next state that will be used as the input to the next round. The 128-bit round keys are expanded from a single key input.

In a straightforward round-based AES implementation, the entire round function is performed combinationally in one clock cycle. Hardware modules are instantiated for each of the functional blocks shown in Fig. 1. In such an implementation, area is dominated by the 16 S-box instances. A comparison of S-box styles by Tillich et al. [12] shows that the most compact S-box designs have an area of around 300 NAND gate equivalents, but the more energy-efficient and shorter critical path S-boxes are several times larger. Previous works have looked at unrolled implementations of block ciphers but they are not necessarily energy efficient [2], [5]. Sub-round implementations perform a fraction of a round in each clock cycle, and this allows a smaller number of S-boxes to be reused across clock cycles thereby saving area.

B. 8-bit architectures in literature

The most compact AES implementations use 8-bit data paths. In such a design, a single S-box is reused 16 times per round, and therefore each round requires at least 16 cycles to complete. 8-bit implementations of AES are less energy-efficient than full-round implementations, and the inefficiency is mainly in the control and data movement. Among the computations performed in a round, SubBytes operates on 8 bits, and AddRoundKey is a bitwise XOR; only MixColumns is natively performed on 32-bit inputs, but is known to have an efficient serialization [7] that takes 8-bit inputs in four consecutive cycles. Since computation itself can scale down to an 8-bit datapath, the inefficiency of 8-bit architectures arises from the costs of moving data around and avoiding hazards. Two dominant techniques for moving data through the computation are RAM and shift register-based schemes.

Early 8-bit AES designs [6], [9] used small RAM blocks to hold state, and control logic to generate addresses to read and write the RAM. Because data can be written to, and read from, arbitrary addresses, these techniques make it easy to avoid data hazards without increasing the amount of storage available. The latency is high in these techniques (534 and 1016 cycles per block respectively) as very little useful work is performed in each cycle. RAM-based techniques can be low in power,

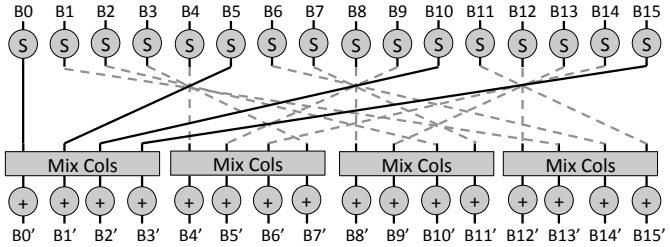

Fig. 1: Structure of AES round. From top to bottom, the 16 data bytes go through SubBytes, ShiftRows, MixColumns, and AddRoundKey. The round outputs are the inputs for the next round.

but relatively higher in energy because of the energy cost of reading and writing data to and from RAM in each cycle.

Shift register-based datapaths improve on RAM-based datapaths and are the most compact way to orchestrate data movement in 8-bit AES. Most of the control complexity is handled implicitly by the wiring, and data bytes proceed in lockstep through the S-box and MixColumns at appropriate times. This shift register-based approach is employed by recent low power implementations [7][13][10] and shown to perform well. Note that the shift-register implementation style causes every byte of the state to move at least 16 times per round (e.g. 20 shifts per round in [7]), and this can have significant energy cost. The total latency of a shift register-based 8-bit AES can be as low as 160 cycles[13], which is a significant improvement over the RAM-based scheme.

C. Shift Rows operation

A complicating factor in sub-round AES implementations is that the round computation produces output bytes in an order that differs from their input order. For example, as shown in Fig. 1, the first quarter of the round computation uses bytes B_0, B_5, B_{10}, B_{15} and produces output values that will become bytes B'_0, B'_1, B'_2, B'_3 for the next round. The round output bytes are produced in sequential order if the input bytes are read in the order $(B_0, B_5, B_{10}, B_{15}, B_4, B_9, B_{14}, B_3, B_8, B_{13}, B_2, B_7, B_{12}, B_1, B_6, B_{11})$; we denote this ordering as Shift Rows Order (SRO). The reordering of bytes by the computation causes a Write After Read (WAR) hazard. As the first Mix Column outputs B'_0, B'_1, B'_2, B'_3 are produced, they must be written to a location that will not overwrite the current values of B_1, B_2, B_3 which have not yet been used in the current round.

III. Methodology

In this section, we describe a clocking methodology that improves energy efficiency of sub-round AES implementations. 8-bit architectures proposed in literature ([10], [13]) spend a lot of energy in data movement. These architectures move data through at least 16 registers per AES round. Our scheme uses register renaming to avoid data hazards without having to store a duplicate copy of the state register. Further, movement of each data byte is limited to 5 registers per round, thereby saving clock and data energy.

A. Improved clocking

In sub-round implementations of AES, care should be taken that the state register is not corrupted by WAR hazard discussed in Sec. II-C. Adding a shadow register file [10] to store intermediate results solves the problem but doubles the area

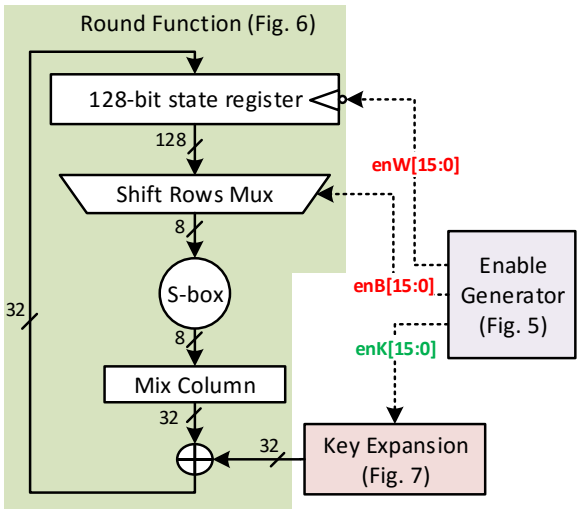

Fig. 2: Proposed 8-bit architecture

of state registers. Shift register based schemes [7], [13] avoid this area penalty by storing the duplicate copy in the shifting behavior of the datapath. However, such an approach has energy inefficiencies due to data movement and clock load. Consider the architecture shown in Fig. 2 which has a state register whose bytes are individually clocked into and out of the registers using enable signals with a timing as shown in Fig. 3. The register outputs are tri-stated and the byte in physical register P_i is passed through the Shift Rows Mux (tri-state) to the S-Box when $enB[i]$ is active. The register enable signals $enB[i]$ are generated such that bytes are read out in Shift Rows Order, and the round function operates on one byte per cycle. The computed results are written back to the state register on the negative edge of the word enable signal $enW[i]$. In this scheme, each byte in the state register is clocked once per round as opposed to 16 times/round in other schemes [13]. Further, data moves through 1 state register and 4 registers in the Mix Column block which is again fewer than the 16 or more moves needed in shift register based schemes. One may notice that the proposed architecture (Fig. 2) does not contain any shadow registers. That is because a duplicate copy of the system state is not stored. WAR hazards are addressed in the following manner. Let byte B_i be read from register P_j for computation. Once the resulting output byte has been computed, it can be written back to register P_j as B_i is no longer required. However, the resulting byte is no longer byte B_i, so now the register P_j is logically renamed to ensure correct functionality.

B. Register Renaming

Let $P_0, P_1, \ldots, P_{15}$ be 16 8-bit **physical** registers that store the 128-bit data. Similarly, let $B_0, B_1, \ldots, B_{15}$ be 16 8-bit **logical** registers, which also correspond to data bytes. The physical registers store the AES state, and the logical registers describe what byte is stored in each register. The correspondence between physical registers and logical registers changes over time, and a logical register may be found in different physical locations in different rounds of AES. At first appearance, this might seem to greatly complicate control flow, because a logical register required for the AES algorithm may

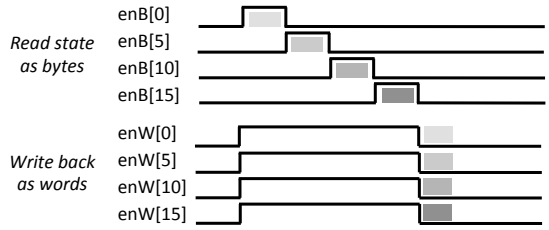

Fig. 3: Timing of control signals for a quarter of one round. enB signals enable bytes to be read from physical registers into the datapath S-Box, and enW signals allow round outputs to be written back to physical registers on falling edge.

need to be accessed from different physical registers across rounds. However, periodicity in register renaming results in a much simpler control logic as discussed below.

We first present the schedule of reading and writing registers used in our scheme, but we do not yet address the design of the control logic that generates the enable signals to realize this schedule. Once we've established here which physical addresses should be enabled in each cycle, we come back to the question of control logic in Sec. IV-A.

Fig. 4 illustrates our scheme of logically renaming registers to avoid the WAR hazards (Sec. II-C); each column in the figure corresponds to a physical address, and the markings in the squares denote the logical addresses contained therein during each cycle. Initially, the logical registers are mapped to the corresponding physical registers, that is $B_i = P_i$ for all i. The black squares in the figure indicate when data is read from each physical register, and the labels on those squares indicate which byte is stored in that register at the time of the read. The blue squares indicate cycles in which bytes of round output are written to physical registers, and the labels on the squares denote which bytes are being written to each register. Grey squares show the time between writing a byte to a physical register and subsequently reading out that same byte. White squares indicate that the byte stored in the physical register has been read, but nothing has yet been written back. For example, in cycle 2 byte B_5 is read from register P_5, computed on for two cycles and the resulting byte (B_1) is written back to P_5 in cycle 5, causing the register to be renamed accordingly. Round boundaries are indicated by thick lines (e.g. after cycle 16). Note that four bytes are written concurrently on every fourth cycle (i.e. in cycles 5,9,13,17 and so on). By the end of four entire rounds (64 cycles), all bytes are returned to the same physical registers in which they started, and the pattern repeats.

Note several very important details of Fig. 4. First, in each round, the bytes are read in Shift Rows Order (B_0,B_5,B_{10}...), although the pattern of reading from physical addresses that realizes this order changes across rounds due to the renaming. Second, in each round, the bytes are written in order with B_0,B_1,B_2,B_3 written first, then the next 4 bytes 4 cycles later, and so on. This means that, aside from the control logic that governs when each register is read and written, the remainder of the AES computation is entirely decoupled from the renaming and clocking scheme. The job of the control logic is then to read each of the physical registers at the times indicated by the black squares, and to write each of the physical registers at the times indicated by the blue squares.

At the beginning of the second round (cycles 17-21) bytes B_0,B_5,B_{10},B_{15}, processed in Shift Rows Order, are read from physical registers P_0,P_9,P_2,P_{11}. The enable signals that

Cycle	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	
-	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	
1	B0																Rounds 1, 5, 9
2						B5											
3											B10						
4																B15	
5	B0				B4	B1					B2					B3	
6										B9							
7															B14		
8				B3													
9				B7	B4				B8	B5					B6		
10														B13			
11			B2														
12								B7									
13			B10					B11	B8				B12	B9			
14		B1															
15							B6										
16												B11					
17	B0	B13					B14					B15	B12				Rounds 2, 6, 10
18										B5							
19			B10														
20												B15					
21	B0		B2		B4					B1		B3					
22														B9			
23							B14										
24																B3	
25					B4		B6		B8					B5		B7	
26		B13															
27											B2						
28				B7													
29		B9		B11					B8		B10		B12				
30						B1											
31															B6		
32								B11									
33	B0					B13		B15					B12		B14		Rounds 3, 7
34														B5			
35											B10						
36								B15									
37	B0				B4			B3			B2			B1			
38		B9															
39															B14		
40												B3					
41		B5			B4				B8			B7			B6		
42						B13											
43			B2														
44																B7	
45			B10			B9			B8				B12			B11	
46										B1							
47							B6										
48				B11													
49	B0			B15			B14			B13			B12				Rounds 4, 8
50		B5															
51			B10														
52				B15													
53	B0	B1	B2	B3	B4												
54						B9											
55							B14										
56								B3									
57					B4	B5	B6	B7	B8								
58										B13							
59											B2						
60												B7					
61									B8	B9	B10	B11	B12				
62														B1			
63															B6		
64																B11	
-													B12	B13	B14	B15	
Cycle	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	

Fig. 4: Illustration of register renaming

control reading (writing) from (to) these physical registers are orchestrated by a control unit (Enable Generator in Fig. 2) that is aware of renaming and tracks bytes across physical registers. In the general case, in our scheme byte B_j, in round k, is mapped to physical register P_i, where i is as shown in Eqn. 1.

$$i = (j + 12k(j \mod 4)) \mod 16 \quad (1)$$

IV. Implementation

In this section, we describe the implementation details of the architecture shown in Fig. 2. An AES round operation consists of Shift Rows (permute bytes from different words), substitution operation (S-box), followed by Mix Column (mix bytes from different words) and addition of round key. All these operations operate on bytes except the Mix Column which

TABLE I: Physical registers to be enabled in each clock cycle

Round	Cycle	+0	+1	+2	+3
1,5,9	1	P_0	P_5	P_{10}	P_{15}
	5	P_4	P_9	P_{14}	P_3
	9	P_8	P_{13}	P_2	P_7
	13	P_{12}	P_1	P_6	P_{11}
2,6,10	17	P_0	P_9	P_2	P_{11}
	21	P_4	P_{13}	P_6	P_{15}
	25	P_8	P_1	P_{10}	P_3
	29	P_{12}	P_5	P_{14}	P_7
3,7	33	P_0	P_{13}	P_{10}	P_7
	37	P_4	P_1	P_{14}	P_{11}
	41	P_8	P_5	P_2	P_{15}
	45	P_{12}	P_9	P_6	P_3
0,4,8	49	P_0	P_1	P_2	P_3
	53	P_4	P_5	P_6	P_7
	57	P_8	P_9	P_{10}	P_{11}
	61	P_{12}	P_{13}	P_{14}	P_{15}

operates on words. The Enable Generator produces byte enable signals (enB in Fig. 2) for registers that cause the AES state to be passed through the Shift Rows MUX in Shift Rows Order during every round of encryption. The data then goes through the S-box, and gets mixed with three other bytes in the Mix Column block. Finally, 32 bits of the round key are added to the data and written back to the state register. As described in the previous section, data is written back to the register it was read from, and renaming ensures no data hazards occur.

A. Enable Generation

Our enable generation logic allows bytes to be processed in appropriate order without the typical 8-bit architectural approach of shifting data through several flip-flops and multiplexers [8]. In our design, the outputs of the state register are multiplexed (by the Shift Rows Multiplexer - SRM) as shown in Fig. 2. We implement the 16:1 SRM using tri-state buffers enabled by the one-hot enB signals produced by the enable generator. To preserve Shift Rows Ordering, the control circuitry generating enB needs to enable the physical registers in each cycle as listed in Tab. I. The physical registers listed in the table correspond to the location of the black squares in Fig. 4 across four rounds. Note that each column in Tab. I can be described as a repeating pattern with a circular shift at round boundaries. The four columns have circular shifts of 0, 1, 2, and 3 positions at the round boundaries, respectively. This observation enables us to generate the 64 cycle pattern of control signals required for datapath orchestration and register renaming at the cost of just 23 single-bit registers, as discussed below.

Fig. 5 shows the Enable Generator. It consists of a single byte-select shift register and four word-select shift registers. As the name implies, the Word Select registers collectively enable a word (32 bits) that the Mix Column block operates on over 4 cycles. The Byte Select unit enables one byte of this word per clock cycle to pass through SRM and to use the datapath S-box (Fig. 2) before entering MixColumns. Each Word Select register shifts around a single 1 value and the current position of the 1 value determines which register is enabled in the current word. The position of the 1 within each word select register only changes on every fourth clock cycle (when $shift$ is asserted). For example, the state of the Word Select registers shown in Fig. 5 causes enable signals $enW[0]$, $enW[5]$, $enW[10]$ and $enW[15]$ to be asserted for the next four cycles. This corresponds to the start of round 1 of AES, in which the first 4 bytes are read from registers P_0, P_5, P_{10}, P_{15}. The Byte Select unit sequences the enB signals for these registers to allow one of the 4 bytes byte per cycle to proceed through SRM through the S-Box and into Mix Columns. Note that this sequence of enable signals across four cycles is the case shown in the waveforms of Fig. 3. At the end of the 4th cycle $rotate$ is asserted, the Word Select shift registers all advance by one position, and the next word (P_4, P_9, P_{14}, P_3) is processed. Once a round of AES is completed at the end of 16 cycles, the control input to the multiplexers in the Word Select shift registers causes them to rotate, and P_0, P_9, P_2, P_{11} becomes the first word processed in the second round. This shift at the round boundary accounts for the register renaming, as these registers are the ones that contain bytes B_0,B_5,B_{10},B_{15} (see cycles 17-20 of Fig. 4).

In this way, despite the apparent complexity of the control signals, the enable generation circuitry comprises only 23 flops. Effectively, the scheme works because the control logic is mimicing the AES shift rows structure, but doing so in the control logic to avoid moving entire bytes around the datapath. Note that the flops in the Word Select shift registers are clocked by the system clock an not a divided clock, even though they only shift every fourth cycle. This is done to avoid having an additional clk-to-q delay on the critical path, as would occur if the shift register used a derived clock.

The enable signals for the Key Expansion unit (enK) are computed by treating Word Select register WS_A and Byte Select register in Fig. 5 as word address and byte address respectively. enK selects each of the 16 key registers (Fig. 7) one per clock cycle in sequential order.

B. Round Function

The schematic of the round function is shown in Fig. 6. All registers and data wires in the figure are 8-bits wide. The state registers are shown in red, and are organized in four groups $\{P_0, P_4, P_8, P_{12}\}$, $\{P_1, P_5, P_9, P_{13}\}$, $\{P_2, P_6, P_{10}, P_{14}\}$ and $\{P_3, P_7, P_{11}, P_{15}\}$. The inputs of all registers in a group are tied together, but since register P_i is clocked by negative edge of $enW[i]$ signal, a byte is always written to one register in a group and is ignored by the other three in the group because their clocks do not switch. The outputs of all 16 P_i registers are connected to the S-Box input via tri-state buffers with $enB[i]$ as the respective byte enables.

During a regular round computation, a four-byte word is selected by asserting four enW signals (Sec. IV-A) for a quarter of a round. enB signal then enables one byte of the word per cycle to pass through to the S-Box. We choose Decode Switch Encode S-box which performs one hot encoding to eliminate glitches and reduce energy consumption [3]. However, our architecture is agnostic to the choice of S-box and one can choose area efficient alternatives [10], [4] if desired. The S-box operation is followed by Mix Column operation which is performed over 4 clock cycles operating on the enabled word. We adopt the Mix Column design from [7] except that we do not pipeline the output. Instead, 32-bits are read out of MixColumns, XORed with 32 bits of round key and written into state at once; this decision prevents stalls, saves three register moves, and reduces clock loading. Given that enW is serving as the clock to the 128 bits of AES state, the registers in our design switch only once per round as opposed to once per cycle in conventional 8-bit architectures [7].

In AES with 128-bit key size, rounds 0 and 10 operate differently than the other rounds. In our scheme, rounds 0 and 10 work as follows. In round 0, plaintext bytes ($Data\_in$) are read sequentially and XORed with corresponding input key bytes (Key_in). To match the "word write" of the regular round, we use three registers to pipeline the data. These registers

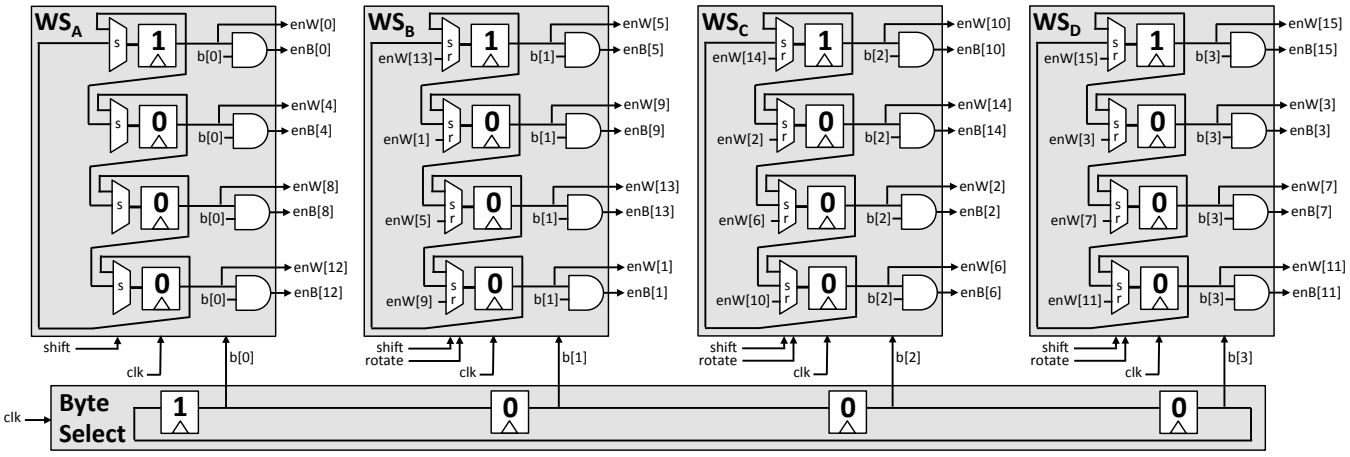

Fig. 5: Enable Generator

can be clock gated after round 0 to save energy. For round 10, Mix Column operation is not required and we XOR the S-Box output with 8-bits of round key to output an encrypted byte $Data\_out$ (Fig. 6). Note that since AES operation is decoupled from renaming, encrypted bytes are output in correct sequential order.

Each round of AES is completed in 16 cycles, leading to a latency of 160 cycles to encrypt a block. Among all the registers in our design, only the Enable Generation and the 32 bits of MixColumn state switch every clock cycle, which is a small percentage of the overall registers in the design. The state registers switch once per round thereby reducing clock load significantly. When considering that each data byte will be clocked through the state once and clocked 4 times through MixColumns, this adds up to only 5 register moves per byte per round, as opposed to approximately 20 moves in conventional 8-bit architectures [7], [10].

C. Key Expansion

The schematic of Key Expansion is shown in Fig. 7. The functionality of key expansion is straightforward (register read/write is sequential) and interested readers can refer to [11] for more detail. The enable signals for the key registers (enK) are generated by the Enable Generator. These signals enable registers K_0 through K_{15} in the same sequence one register per cycle. The key registers are similar to data registers in that their outputs are tri-stated and they sample data on the negative edge of enK, once per round. The enable for the tristates are slightly different as shown in Fig. 7. Each byte i from words K_a, K_b and K_c is enabled twice, once while computing round key, and once more while being XORed with corresponding byte in the next successive word. Bytes from word K_d are also enabled twice - for computing g-function and round key.

V. Results

In this section, we present results using our clocking methodology. The RTL for the 8-bit datapath is written by us and validated against an online tool. We use Synopsys Design Compiler for synthesis, and Synopsys HSIM for circuit simulation with NCSU 45nm PDK [1]. We use the nominal voltage (1.1V) for our experiments. Tab. II compares the energy per encryption (pJ/bit) of our scheme with two references designs which we implement ourselves for fair comparison. The 128-bit datapath, as expected, is the most energy efficient design

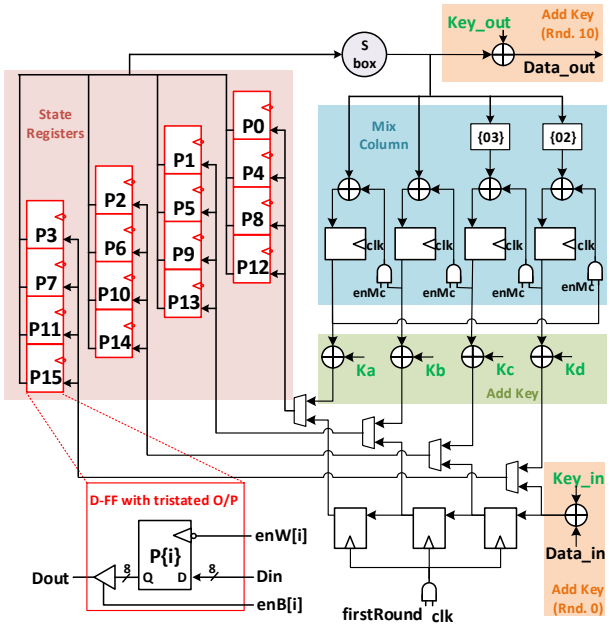

Fig. 6: Schematic of Round Function

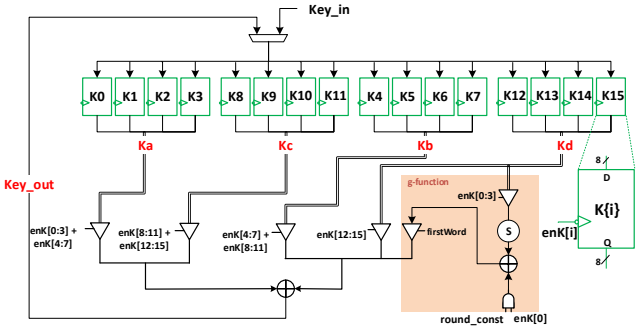

Fig. 7: Schematic of Key Expansion

TABLE II: Comparison of energy per encryption (pJ/bit) breakdown of our scheme with 128-bit and 8-bit reference designs implemented by us

	128-bit Ref.	**Our 8-bit**	**8-bit Ref. [13]**
Shift Rows	–	–	0.29
S-box (data&key)	1.33	0.58	1.10
Mix Column	0.44	0.61	0.66
Key Expansion	0.15	0.12	0.18
Flip-flops	0.39	1.48	6.18
Tristates	–	0.81	–
CLK	0.07	0.36	1.19
Control	–	0.81	N/A
Other	0.54	1.70	0.73
Energy	**2.92**	**5.66**	**10.33**

TABLE III: Comparison of area breakdown of our scheme with 128-bit and 8-bit reference designs implemented by us

	128-bit Ref.	**Our 8-bit**	**8-bit Ref. [13]**
Shift Rows	–	–	78.2
S-box (data&key)	10395.4	1045.9	1039.5
Mix Column	223.4	109.1	108.0
Key Expansion	1038.8	85.9	110.6
Flip-flops	1157.6	1410.9	1338.5
Tristates	–	902.3	–
Control	–	277.4	N/A
Other	984.3	178.0	87.3
Area	**13799.6**	**4009.4**	**2762.1**

at 2.92 pJ/bit energy consumption because it incurs minimal data movement overhead. Our implementation of the reference 8-bit design [13] consumes 10.33pJ/bit for an encryption. Our design achieves a 45% improvement at 5.66pJ/bit. The energy numbers reported in [13] are different than our implementation of their scheme, due to several factors such as S-box design, 65nm technology and voltage scaling.

The energy numbers shown in Tab. II demonstrate the specific benefits of using our clocking methodology. From table in comparison to 8-bit reference [13], our design consumes 4x less flip-flop energy because data moves through 5 flops per round instead of 20. Further, our design spends 70% less clock energy. This is because all 296 flops in the reference design switch every clock cycle. In our design 61 flops involved in Mix Column operation and enable generation switch every cycle, while the rest (280) of the flops holding data and key switch once per round. By optimizing the individual components in the design (like the S-box) one could potentially improve the overall energy efficiency of all designs listed in Tab. II.

An area comparison is presented in Tab. III. The 128-bit design is obviously the most expensive in terms of area. Our design incurs a 45% area penalty compared to our implementation of the reference 8-bit design [13], consuming an extra $1250\mu m^2$. This area penalty comes mainly from tristates which are used to MUX data, and the control logic for generating enable signals. The tristates in NCSU 45nm PDK can be redesigned to reduce area penalty. Finally, our scheme is competitive in performance with [13] as seen in Tab. IV with a 20% degradation of f_{max} due to addition of tristates and control signals in the critical path. Running at f_{max}, our scheme consumes 2.3 mW of power.

TABLE IV: Comparison of performance

	128-bit Ref.	**Our 8-bit**	**8-bit Ref. [13]**
f_{max} (MHz)	636	510	636
Latency (cycles)	10	160	160
Throughput (Mbps)	8141	408	504

VI. Conclusion

In this work, we presented a microarchitectural technique to improve energy efficiency of 8-bit implementation of AES. Our improved clocking methodology greatly reduces activity of data and key registers to a single update per round, which is at least 16x smaller than conventional 8-bit implementations. Register renaming eliminates the need for additional state registers to store intermediate results and minimizes data movement through registers, thereby saving energy. In comparison to the most efficient 8-bit implementations, we consume 45% lower energy for an encryption operation with a 70% reduction in clock energy, while paying a 45% area cost. Our methodology can be extended to other sub-round implementations of AES like 32-bit which we will explore as part of future work. For a 32-bit implementation, energy inefficiencies in data movement and clocking are smaller and so would be the energy savings using our scheme. In conclusion, the proposed 8-bit AES architecture presents a new energy-efficient implementation style that is attractive for low-power designs such as mobile SoCs.

Acknowledgments

This research was supported through the STARSS program by National Science Foundation Grant CNS-1619558 and Semiconductor Research Corporation Task 2685.001.

References

[1] NCSU Free PDK 45. http://www.eda.ncsu.edu/wiki/FreePDK45:Contents.

[2] Banik, S., Bogdanov, A., Regazzoni, F., Isobe, T., Hiwatari, H., and Akishita, T. Round gating for low energy block ciphers. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (May 2016), pp. 55–60.

[3] Bertoni, G., Macchetti, M., Negri, L., and Fragneto, P. Power-efficient ASIC Synthesis of Cryptographic Sboxes. In *Proceedings of the 14th ACM Great Lakes Symposium on VLSI* (New York, NY, USA, 2004), GLSVLSI '04, ACM, pp. 277–281.

[4] Canright, D. A very compact S-box for AES. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2005), Springer, pp. 441–455.

[5] Dhanuskodi, S. N., and Holcomb, D. Energy optimization of unrolled block ciphers using combinational checkpointing. In *RFIDSec 2016: 12th Workshop on RFID and IoT Security, 2016* (Dec 2016).

[6] Feldhofer, M., Dominikus, S., and Wolkerstorfer, J. Strong authentication for RFID systems using the AES algorithm. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2004), Springer, pp. 357–370.

[7] Hamalainen, P., Alho, T., Hannikainen, M., and Hamalainen, T. D. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *9th EUROMICRO Conference on Digital System Design (DSD'06)* (2006), pp. 577–583.

[8] Jarvinen, T., Salmela, P., Hamalainen, P., and Takala, J. Efficient byte permutation realizations for compact AES implementations. In *2005 13th European Signal Processing Conference* (Sept 2005), pp. 1–4.

[9] Kaps, J.-P., and Sunar, B. Energy comparison of AES and SHA-1 for ubiquitous computing. In *International Conference on Embedded and Ubiquitous Computing* (2006), Springer, pp. 372–381.

[10] Mathew, S., Satpathy, S., Suresh, V., Anders, M., Kaul, H., Agarwal, A., Hsu, S., Chen, G., and Krishnamurthy, R. 340 mV;1.1 V, 289 Gbps/W, 2090-Gate NanoAES Hardware Accelerator With Area-Optimized Encrypt/Decrypt GF(2 4) 2 Polynomials in 22 nm Tri-Gate CMOS. *IEEE Journal of Solid-State Circuits 50*, 4 (April 2015), 1048–1058.

[11] Pub, N. F. 197: Advanced encryption standard AES. *Federal Information Processing Standards Publication 197* (2001), 441–0311.

[12] Tillich, S., Feldhofer, M., and Grosssschädl, J. Area, delay, and power characteristics of standard-cell implementations of the AES S-box. In *International Workshop on Embedded Computer Systems* (2006), Springer, pp. 457–466.

[13] Zhao, W., Ha, Y., and Alioto, M. AES architectures for minimum-energy operation and silicon demonstration in 65nm with lowest energy per encryption. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (May 2015), pp. 2349–2352.