Journal of Cryptographic Engineering (2022) 12:53–73
https://doi.org/10.1007/s13389-020-00251-6

REGULAR PAPER

Melting SNOW-V: improved lightweight architectures

Andrea Caforio[1] · **Fatih Balli**[1] · **Subhadeep Banik**[1]

Received: 7 July 2020 / Accepted: 17 November 2020 / Published online: 4 December 2020

Abstract
SNOW-V is a stream cipher proposed by Ekdahl et al. at IACR ToSC 2019(3) with an objective to be deployed as the encryption primitive in 5G systems. The stream cipher offers 256-bit security and is ready for deployment in the post-quantum era, in which as a rule of thumb (due to Grover's algorithm), quantum security will vary as the square root of the classical security parameters. The authors further report good software performance figures in systems supporting the AES-NI instruction set. However, they only provide a theoretical analysis of the cipher's hardware efficiency. In this paper, we aim to fill this gap. We look at the three most important metrics of hardware efficiency: area, speed and power/energy, and propose circuits that optimize each of these metrics and validate our results using three different standard cell libraries. The smallest SNOW-V circuit we propose occupies only around 4776 gate equivalents of silicon area. Furthermore, we also report implementations which consume as little as 12.7 pJ per 128 bits of keystream and operate at a throughput rate of more than 1 Tbps.

Keywords SNOW-V · Stream cipher · 5G · Hardware · Implementation

1 Introduction

Digital cellular communication networks have come a long way since the inception of mobile telephony. The first iteration dates back to the late seventies, commonly termed 1G. Ever since, the advances in communication technology have produced fresh alterations every decade. Text messaging and encryption was introduced in the second generation, while the third generation unlocked the gates of cyberspace on mobile devices. 4G, for the first time, lifted the down-link throughput rate above 1 gigabit per second and has since become the most prevalent network across the globe. Finally, the fifth generation further obliterates this speed bound as it aims at throughput rates of 20 gigabits per second and beyond.

SNOW-V is a stream cipher proposed by Ekdahl et al. [7] with an objective to be deployed as the encryption primitive in 5G systems. It is closely based on the stream cipher SNOW 3G [8] that was designed as the encryption function used in 3G systems. The stream cipher has 256-bit security and can be efficiently implemented on architectures supporting the AES-NI instruction set. However, there does not exist any independent analysis of the efficiency of the stream cipher on hardware ASIC except theoretical estimates of hardware efficiency by the designers. In this paper, we look at the three most important metrics of hardware efficiency: area, speed and power/energy. We propose circuits that optimize each of these metrics and validate our results using three different standard cell libraries.

✉ Andrea Caforio
andrea.caforio@epfl.ch

Fatih Balli
fatih.balli@epfl.ch

Subhadeep Banik
subhadeep.banik@epfl.ch

[1] LASEC, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

1.1 Contribution and organization

For low-area designs, we reduce the data path to one byte. Ideally, we would want byte serial AES circuits that are able to accept inputs byte by byte in every clock cycle. To reduce latency, it would be also desirable if the AES operations could be processed in 16 cycles (i.e. once byte per cycle), but most byte serial AES circuits [2,3] take more than 21 cycles to process an AES round function. So our first contribution was to take the Atomic-AES circuit and make suitable modifications to it to ensure a 16-clock encryption cycle. This enabled us to create the SNOW-V byte serial circuit around the modified Atomic-AES circuit that is able to seamlessly perform all

operations in a byte serial manner while keeping the throughput above 40 Gbits per second.

For the high-throughput and low-energy constructions, we revert back to data paths of 128 bits, since it is already known that byte serial architectures, on account of increased latency can lead to neither throughput nor energy efficiency. To get the optimal circuits for these performance metrics, we proceed in two directions. The first direction is concerned with the optimization of individual components of the circuits: we experiment with various architectures of the AES S-box, Mixcolumn and 32-bit adder which serve as core components of the SNOW-V circuit. The second direction is using generic techniques like round unrolling to improve performance.

The paper is organized in the following manner. Section 2 introduces an algorithmic description of the cipher. In Sect. 3, we first describe a modified implementation of the Atomic-AES architecture that executes AES operations in 16 cycles. Thereafter, we describe our SNOW-V architecture with low silicon area. Section 4 looks at both high-speed/low-energy architectures for the stream cipher, primarily because the routines and concepts involved in the optimization of these metrics are heavily inter-related. In Sect. 5, we present extensive synthesis and simulation results for all the architectures using three different standard cell libraries. This section also includes extensive discussion about the results and tries to makes sense of the sizable array of data tabulated in it. Section 6 concludes the paper.

2 Preliminaries

The SNOW-V stream cipher algorithm consists of two 256-bit linear-feedback shift registers that are continuously fed into a finite-state machine, consisting of two chained AES-128 round functions, that produces the key stream.

2.1 LFSR

Each 256-bit linear-feedback shift register consists of sixteen 16-bit cells. We denote by $a_0, \ldots, a_{15}$ the cells of LFSR-A and analogously $b_0, \ldots, b_{15}$ for the cells of LFSR-B. Each cell is a finite-field element over the extension $\mathbb{F}^A_{2^{16}} = \mathbb{F}_2[x]/g^A(x)$ for LFSR-A and $\mathbb{F}^B_{2^{16}} = \mathbb{F}_2[x]/g^B(x)$ for LFSR-B such that $g^A(x)$ is a primitive polynomial of the form

$$g^A(x) = x^{16} + x^{15} + x^{12} + x^{11} + x^8 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x].$$

Similarly, $g^B(x)$ is given by the primitive polynomial

$$g^B(x) = x^{16} + x^{15} + x^{14} + x^{11} + x^8 + x^6 + x^5 + x + 1 \in \mathbb{F}_2[x].$$

The feedback function of LFSR-A is given by the equation

$$a_{16} = b_0 \oplus \alpha a_0 \oplus a_1 \oplus \alpha^{-1} a_8,$$

where α is a root of $g^A(x)$ and α^{-1} its inverse over $\mathbb{F}^A_{2^{16}}$. Analogously, we can denote the feedback function of LFSR-B by

$$b_{16} = a_0 \oplus \beta b_0 \oplus b_3 \oplus \beta^{-1} b_8,$$

where β is a root of g^B and β^{-1} its inverse over $\mathbb{F}^B_{2^{16}}$.

An important property of both $\mathbb{F}^A_{2^{16}}$ and $\mathbb{F}^B_{2^{16}}$ is the efficient execution of the field multiplication by α, α^{-1} and β, β^{-1}, respectively. As a matter of fact, the operation can be encoded by two simple routines both consisting of a bit-wise shift and a XOR. Since α is a primitive element in $\mathbb{F}^A_{2^{16}}$, it is possible to write each element $w \in \mathbb{F}^A_{2^{16}}$ in the basis $(1, \alpha, \ldots, \alpha^{15})$ such that $w = w_{15}\alpha^{15} + \cdots + w_1\alpha + w_0$, where w_i are the individual bits of w. As a consequence, we can rewrite the multiplication $w\alpha$ as

$$\begin{aligned} w\alpha &= (w_{15}\alpha^{15} + \cdots + w_1\alpha + w_0)\alpha \\ &= w_{15}(\alpha^{15} + \alpha^{12} + \alpha^{11} + \alpha^8 + \alpha^3 + \alpha^2 + \alpha + 1) \\ &\quad + w_{14}\alpha^{15} + \cdots + w_0\alpha. \end{aligned}$$

The upper byte of the above product can be written as $[w_{14}, w_{13}, \ldots, w_7] + w_{15} \cdot C_{\mathrm{H}}$ and the lower byte as $[w_6, w_5, \ldots, w_0, 0] + w_{15} \cdot C_{\mathrm{L}}$ where $C_{\mathrm{H}} = 10011001$ and $C_{\mathrm{L}} = 00001111$. A similar reasoning can be followed for the multiplication by the inverse element α^{-1} and for the field $\mathbb{F}^B_{2^{16}}$. Algorithm 1 depicts the LFSR feedback routine for eight consecutive updates.

Algorithm 1 LFSR Update

1: **for** $i \in \{0, \ldots, 7\}$ **do**
2: $\quad \mathsf{tmp}_a \leftarrow b_0 + \alpha a_0 + a_1 + \alpha^{-1} a_8 \bmod g^A(\alpha)$
3: $\quad \mathsf{tmp}_b \leftarrow a_0 + \beta b_0 + b_3 + \beta^{-1} b_8 \bmod g^B(\beta)$
4: $\quad (a_{15}, a_{14}, \ldots, a_0) \leftarrow (\mathsf{tmp}_a, a_{15}, \ldots, a_1)$
5: $\quad (b_{15}, b_{14}, \ldots, b_0) \leftarrow (\mathsf{tmp}_b, b_{15}, \ldots, b_1)$

2.2 FSM

The finite-state machine is composed of two chained AES-128 round functions that are continuously fed with the updated LFSR states. Key addition is not performed in the AES cores,

i.e. the computation only consists of the substitution layer, the Shiftrows and Mixcolumn procedures. We denote by $\boxplus_{32}$ an adder that operates on 32-bit blocks. Furthermore, σ represents a byte-wise permutation of the form

$$\sigma = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15].$$

Algorithm 2 FSM Update

$T2 \leftarrow (a_7, a_6, \ldots, a_0)$
tmp $\leftarrow R2 \boxplus_{32} (R3 \oplus T2)$
$R3 \leftarrow \mathsf{AES}^R(R2)$
$R2 \leftarrow \mathsf{AES}^R(R1)$
$R1 \leftarrow \sigma(\text{tmp})$

2.3 Initialization routine

The initialization phase lasts for 16 rounds during which the FSM and LFSR are iteratively updated. Furthermore, the usual keystream output z is mixed into LFSR-A in each iteration. Finally, the state $R1$ is additionally updated with the key during the last two rounds.

Algorithm 3 Initialization

$(a_{15}, a_{14}, \ldots, a_8) \leftarrow (k_7, \ldots, k_0)$
$(a_7, a_6, \ldots, a_0) \leftarrow (iv_7, \ldots, iv_0)$
$(b_{15}, b_{14}, \ldots, b_8) \leftarrow (k_{15}, \ldots, k_8)$
$(b_7, b_6, \ldots, b_0) \leftarrow (0, \ldots, 0)$, $R1, R2, R3 \leftarrow 0, 0, 0$
for $i \in \{1, \ldots, 16\}$ **do**
  $T1 \leftarrow (b_{15}, \ldots, b_8)$
  $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$
  FSM Update
  LFSR Update
  $(a_{15}, \ldots, a_8) \leftarrow (a_{15}, \ldots, a_8) \oplus z$
  if $i = 15$ **then** $R1 \leftarrow R1 \oplus (k_7, \ldots, k_0)$
  if $i = 16$ **then** $R1 \leftarrow R1 \oplus (k_{15}, \ldots, k_8)$

2.4 Keystream routine

The keystream routine only differs from the initialization procedure in Algorithm 3 by omitting the supplementary updates of LFSR-A and $R1$.

Algorithm 4 Keystream

Initialization
while 1 **do**
  $T1 \leftarrow (b_{15}, \ldots, b_8)$
  $z \leftarrow (R1 \boxplus_{32} T1) \oplus R2$
  FSM Update
  LFSR Update
  output z

3 Low-area design

The SNOW-V white paper sketches a 64-bit and a 128-bit data path design both being configurable with either an internal or external AES-128 round function engine. Data paths of these widths guarantee a high throughput but come at the cost of a large area overhead, since most of the combinatorial components such as multiplexers or the AES-128 units span over the entire width. In its current forms, the SNOW-V architecture is not suitable for lightweight devices such as smart-cards. Since the momentum of Internet-of-things applications is ever increasing, it is important that such devices are adequately protected in a 5G environment. This means the area footprint has to be reduced significantly without compromising the throughput guarantees too much. In [7], the designers provide theoretical estimates for the area/throughput for their architecture. The area figures are given for the Samsung 350 nm library, and throughput figures based on the NanGate 15 nm library. We recalculated the area figures using the gate-count proposed by the designers and the actual silicon areas of the corresponding gates of three different standard cell libraries. We present the estimates in Table 1.

Note that theoretical estimates of area/throughput are not always accurate. The reason for that is theoretical estimates are usually based on the area/delay characteristics of the basic gate level elements of the circuit. The estimates presented in [7], for example, are calculated based on the count of the number of 2-input xor gates, nand gates, etc., whereas a typical standard cell library consists of many gates (some which accept more than 3 or 4 input wires). Any circuit compiler would, in the process of mapping the circuit into silicon, make use of all of them to give the best possible configuration with respect to area, power or delay.

In this section, we propose a new low-area design of SNOW-V that reduces the area requirement by up to 70 percent with respect to the area results proposed in the white paper. More specifically, we reinterpret the architecture in a way that supports a general serialization of the procedures. Our design features an 8-bit data path and employs two AES-128 round functions at the same time. As a concrete construction of the encryption cores, we utilize the byte-serial Atomic-AES v2.0 design by Banik et al. [2,3] that currently stands as the lowest-area implementation on a data path of this width. Although our design choice already significantly shrinks the overall circuit area, we investigate different design choices that aim at further gains such as single-bit data paths and the usage of only one encryption core.

3.1 Atomic-AES

Atomic-AES [2] constitutes a byte-serial AES-128 architecture that combines both an encryption and a decryption module. Its circuit exhibits a particularly small area of 2605

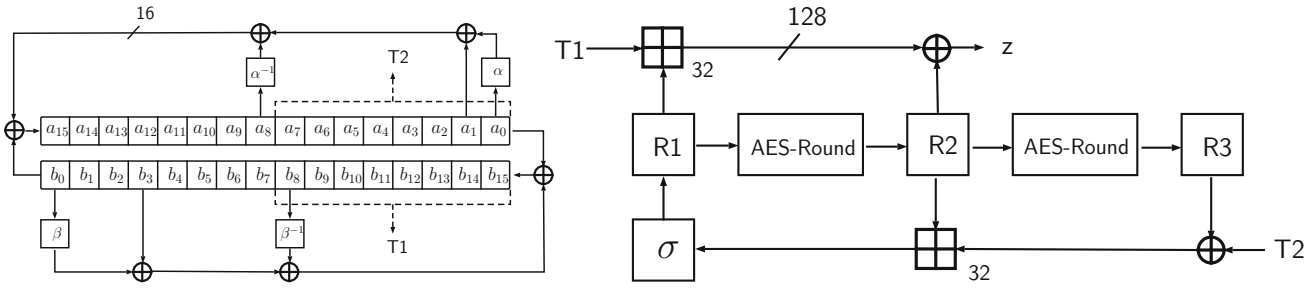


Fig. 1 High-level schematic of the SNOW-V linear-feedback shift registers (left) and the finite-state machine (right). Note that the initialization logic and the auxiliary multiplexers have been omitted

Table 1 Theoretical circuit area figures for the SNOW-V architecture in the white paper

	1 ext. core	1 int. core	2 ext. cores	2 int. cores
STM 90 nm				
Area (GE)	6729	10909	8024	16384
TSMC 90 nm				
Area (GE)	8751	13683	10238	20102
NanGate 15 nm				
Area (GE)	11362	16634	12062	22606
TP_{max} (Gbps)	358	358	712	712

Estimates for throughput were provided for only the NanGate 15 nm process

gate equivalents, at the cost of an increased number of computation cycles, i.e. 226 for both the encryption and decryption routines. In a subsequent work, termed Atomic-AES v2.0 [3], the authors further lowered the circuit area to 2060 GE operating at a latency of 246 cycles for encryption and 326 for decryption.

As only a single AES-128 round function invocation without key addition is required for the SNOW-V encryption steps, it is possible to reduce the Atomic-AES v2.0 circuit area and latency significantly. For a start, the entire key schedule unit can be removed; this comprises all the key registers and its accompanying control logic as well as the round constants. The original Atomic-AES v2.0 design can handle both encryption and decryption queries; hence, further gains are obtained by cutting all the decryption logic such as the inverse substitution layer. We also make some other modifications to accommodate one AES round in only 16 cycles. This represents a marked improvement over Atomic-AES (21 cycles) and Atomic-AES v2.0 (23 cycles). The principal difference of our circuit over the basic Atomic-AES v2.0 architecture is as follows:

A: Limiting Shiftrows logic to the bottom-most row of the AES state. In Atomic-AES v2.0, each row had separate circuitry for this functionality.
B: Moving the Mixcolumn circuit to the second column of the state. In Atomic-AES v2.0, this circuit was placed after the very first column.

To see how these modifications help reduce the latency, we need to inspect the data flow. Note that we assume a row-orientation of the bytes as shown in Fig. 2, i.e. the bytes enter the pipe row after row. Each of the cells XY in the figure represents an 8-bit flip-flop. Let us index the bytes by the symbols $b_0,\ b_1, \ldots,\ b_{15}$. Also denote by $s_i = S(b_i)$ where S is the AES S-box function.

Cycles 0 to 6: The first operation performed on the incoming bytes is always the S-box function, due to which the corresponding S-box circuit is placed at the entry to the register pipeline. In AES, the first row remains unaffected by the Shiftrows operation. Hence, the first seven bytes of the state, after the S-box operation (i.e. s_0 to s_6), enter the pipeline and are loaded into cells `21` to `33`.

Cycle 7: The values in cells `31`, `32`, `33` (i.e. $s_4,\ s_5,\ s_6$) and the 8th state byte s_7 (which is at this time present on the wire at the pipeline input) enter the Shiftrows component. The bytes are rotated to the left by 1, and result is made available on the input wires to the cells `30`, `31`, `32` and `33`. At the next rising clock edge (i.e. of cycle 8), these values are written into the respective cells.

Cycles 8 to 10: The ninth to eleventh state bytes ($s_8,\ s_9,\ s_{10}$) are piped into cells `31`, `32` and `33`.

Cycle 11: The second Shiftrows operation is applied to the third state row, i.e. the values in the cell `31`, `32`, `33` (s_8, s_9, s_{10}) and the pipeline input (s_{11}). Note that the second Shiftrows operation rotates bytes by two positions to the left. To accommodate this functionality, the Shiftrows

component has multiplexers that allow rotation by different positions during different periods of the execution cycle. As before, the Shiftrows outputs are written onto the registers at the next clock edge.

Cycles 12 to 14: The state bytes $s_{12},\ s_{13},\ s_{14}$ are piped into cells `31`, `32` and `33`.

Cycles 15: At the beginning of this cycle, the bytes in the cells `01`, `11`, `21` are s_0, s_5, s_{10}, and the byte waiting in the wire at the pipeline input is s_{15}. By the specifications of the AES round function, these are exactly the bytes that are placed in the first column after the first Shiftrows operation, and thus also the input bytes to the first Mixcolumn operation. Hence, wires from the cells `01`, `11`, `21` and the pipeline input are fed to the Mixcolumn circuit, and the output is written to the cells `00`, `10`, `20` and `30` in the next clock edge (i.e. cycle 16).

This cycle is also used to perform the last shiftrow operation i.e. on the fourth row. According to the specification, the shiftrow operation on the fourth row makes the following transformation:

$$(s_{12}, s_{13}, s_{14}, s_{15}) \rightarrow (s_{15}, s_{12}, s_{13}, s_{14}).$$

By forwarding s_{15} directly to the Mixcolumn circuit, we, in a sense, perform both these operations at once on this particular byte. The remaining bytes (s_{12}, s_{13}, s_{14}) that are in the cells `31`, `32` and `33` are entered into the Shiftrows circuit, and the result, as before, is written onto the cells in the rising edge of the next clock cycle.

Cycles 16 to 31: At cycle 16, the byte at `00`, after having executed all the AES component functions, is available at the output wire of the pipeline. Thus, each byte spends exactly 16 cycles in the pipeline. The remaining Mixcolumn operations are performed in cycles 16, 17 and 18. This time there is no need to perform a concurrent shiftrow operation as in the case of s_{15}, and so the Mixcolumn inputs are taken from the cells cells `01`, `11`, `21` and `31`. It is not difficult to see that the bytes are correctly aligned before each Mixcolumn execution takes place. As a result, at each clock cycle $16 + i$, $\forall i \in [0, 15]$ the ith output byte of the AES round function is available on the output wire of the pipeline.

The pruned Atomic-AES v2.0 round function is depicted in Fig. 2. The concrete area and latency gains between the pruned and unchanged variation are listed in Table 2.

3.2 Serialization in SNOW-V

Serialization of a primitive often complicates the involved procedures as the algorithms are not laid out for narrow data paths. More concretely, parallelization is often forgone in the wake of area reductive measures. However, in the case of SNOW-V, a serial architecture comes naturally as supported by the following observation.

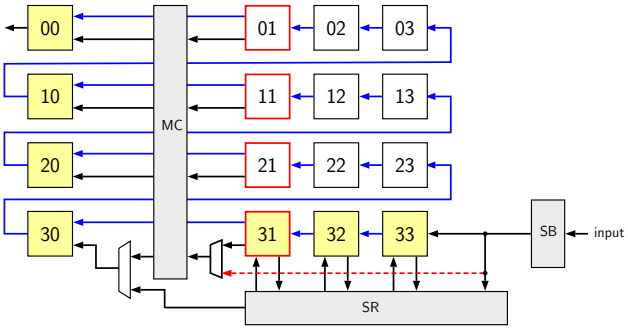


Fig. 2 Pruned Atomic-AES v2.0 round function. Scan flip-flops are marked yellow, ports to the Mixcolumn component are coloured in red and the wires that serially connect the cells are in blue colour

Observation 1 *During the 16 cycles of each round (here one round refers to one iteration of the "while loop" in Algorithm 4), (a) the calculation of z and tmp, (b) the update of the linear-feedback shift registers and (c) the loading of both encryption cores can be performed concurrently. Furthermore, this also holds for the inclusion of z into one LFSR state during the initialization phase.*

Note that a byte serial implementation of SNOW-V would have the LFSR cells chained byte-wise one after the other, which means that for a byte would have to move for two cycles to traverse one LFSR element and hence it would take 16 cycles to update each LFSR 8 times for one round execution. So LFSRs are now composed of 32 8-bit registers $(a_{15H}, a_{15L} ..., a_{0L})$ and $(b_{15H}, b_{15L} ..., b_{0L})$, respectively. We can model $R1$, $R2$ and $R3$ as shift registers with a granularity of eight bits. The AES encryption algorithm has already been shown to operate in 16 cycles. In a byte serial architecture, the data transfer is byte-wise and hence any new byte introduced in the most significant bytes $\{a, b\}_{15H}$ takes 32 cycles to traverse the entire LFSR. A newly appended LFSR byte, through location i.e. $\{a, b\}_{15H}$, is only included in the update function when it reaches position $\{a, b\}_{8L}$ which again takes 16 cycles. For the calculation of z, the circuit uses the byte driven out from the register housing b_{8L} initially (i.e. one of two the middle most byte registers of the LFSR B), and for the evaluation of tmp, it uses the byte from the register housing a_{0L} (the first in LFSR A). Note that the newly introduced elements do not interfere with the calculations of the current round for 16 cycles. In Fig. 3, we illustrate this fact for LFSR-B.

During one cipher round, 16 update calls are made to the LFSR unit. We denote by u_i a freshly inserted element and highlight the cells involved in the feedback function in green/light blue. Clearly, no element u_i enters the feedback function during one cipher round.

The same argument holds for the calculation of z and tmp as no element u_i will be used in the current round. The implication of this is that 16 cycles are enough to update both the

Table 2 Atomic-AES v2.0 area and latency comparison between a regular encryption core and the pruned variant. The area figures were obtained with the STM 90 nm process

Core	Area (GE)	Pruned area (GE)	Latency (cycles)	Pruned latency (cycles)
Atomic-AES v2.0	2060	1018	23	16

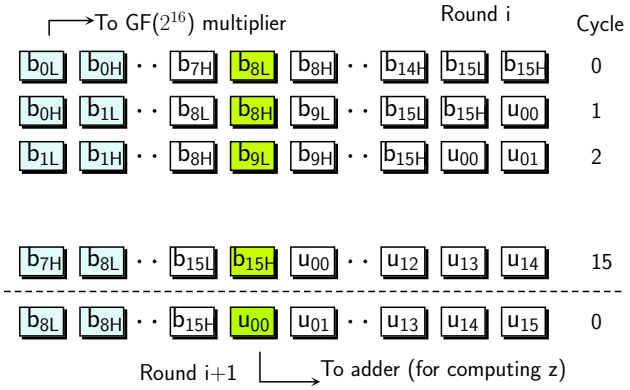


Fig. 3 Cycle-by-cycle data flow for LFSR-B. Each box denotes one-byte-sized register

LFSRs and the FSM housed in $R1$, $R2$, $R3$ and all operands in the computation of z and tmp can thus safely be shifted into the encryption cores and loaded into the FSM without any timing violation.

3.3 Dual-core

In a dual-core setting, two AES-128 cores are utilized; hence, the two encryptions of each round can be performed concurrently at the cost of a sharp spike in circuit area. The following observation elegantly mitigates this area issue.

Observation 2 *The intermediate values $R2$ and $R3$ are direct outputs from the* AES *encryption cores. Consequently, we can build the* AES *logic circuitry around these registers. Note that $R2 = AES^R(R1)$. In a byte serial data path, we drive bytes out of register $R1$ in every clock cycle, into the $R2$ register which has the pruned* Atomic-AES v2.0 *functionality as explained in 3.1. This being so, after exactly 16 cycles the output port of $R2$ produces, in the same input order, the bytes of the intermediate state $AES^R(R1)$. Since we also have $R3 = AES^R(R2)$, the $R3$ register can also be constructed similarly.*

3.4 Data flow

The data flow of our byte-serial dual-core implementation should closely match the encryption cycles. In accordance with Observation 1, it follows that each cipher round should exactly take 16 cycles during which a 128-bit key stream output is produced. More specifically, we require 64 cycles to instantiate the linear-feedback shift registers with its initial values, assuming there is only a single 8-bit input port to the circuit, and then proceed in 16-cycle intervals during which all the operations are executed in parallel. This includes the mixing of the output value z into LFSR-A during the initialization rounds as well as the additional mixing of the key material into $R1$. We asses the total number of cycles for the entire initialization procedure at $64 + 16 \cdot 16 = 320$ for the byte-serial construction based on the pruned Atomic-AES v2.0 core.

Table 3 SNOW-V initialization routine (rounds-4 to 15) and key stream (rounds 16 and above) sequence of operations

Round	Operations
-4	Load IV
-3	Load $k_0, \ldots, k_7$
-2	Load 0
-1	Load $k_8, \ldots, k_{15}$
0–13	FSM Update, LFSR Update, LFSR-A $\oplus$ z
14–15	FSM Update, LFSR Update, LFSR-A $\oplus$ z, $R1 \oplus k$
16-	FSM Update, LFSR Update, Keystream generation

Table 3 details the exact sequence of operations for both the initialization routine as well as the computation of key stream material.

Let us look at the table more carefully. Each round in the table consists of 16 clock cycles, and the data flow occurs in the following manner:

Rounds -4 to -1: These are the key and IV loading rounds. Since we have only one 8-bit input port in the circuit, we can introduce $16 * 8 = 128$ bits into the circuit in a single round. Round -4 is used to enter the IV, -3 to enter the first 16 bytes of the key, -2 to enter the initialization constant 0^{128} and -1 to enter the remaining bytes of the key.

Rounds 0 to 13: These are the first 14 initialization rounds used in SNOW-V. According to Algorithm 3, computing z, FSM Update, LFSR Update and adding z to $a_{15}||a_{14}||\cdots||a_8$ could be executed concurrently. z is given as $(R1 \boxplus_{32} (b_{15}||b_{14}||\cdots||b_0)) \oplus R2$. The bytes that are $R1$ and b_{8L} are passed through an 8-bit ripple-carry adder (with the carry in reset every four cycles to implement four independent 32-bit additions). The output of the adder is xored with the bytes driven out of $R2$ to produce a new byte of z every clock cycle.

The FSM Update requires to produce

A: $\mathsf{tmp} \leftarrow R2 \boxplus_{32} (R3 \oplus (a_7||a_6||\cdots||a_0))$
B: $R3 \leftarrow \mathsf{AES}^R(R2)$, $R2 \leftarrow \mathsf{AES}^R(R1)$
C: $R1 \leftarrow \sigma(\mathsf{tmp})$

tmp is produced byte-wise from the bytes coming out of $R2$, $R3$ and a_{0L} in a similar manner as z was produced. Note that the updated value of $R1$ is $\sigma(tmp)$ where σ is a byte-wise shuffle. To implement this functionality, the newly produced tmp bytes are driven in through the input port of $R1$ for the first 15 cycles of the round. The cells in $R1$ are additionally wired to implement the shuffle function (this is done by using scan flip-flops that additionally wire flip-flop t with flip-flop $\sigma(t)$). In the 16th cycle, the bytes in $R1$ are shuffled by enabling the scan functionality. As explained in Observation 2, the intermediate AES operations are also performed byte-wise in 16 cycles on bytes driven out of $R1$ and $R2$, the results of which are written byte-wise on to $R2$ and $R3$, respectively.

The LFSR Update requires to update each LFSR 8 times which also takes 16 cycles. Note that the finite field multiplications are defined over $GF(2^{16})$, whereas the data path is limited to 8 bits. To overcome this, we construct a special multiplier architecture (see Sect. 3.5) that produces the 8 lsbs of the product in one cycle and the 8 msbs of the product in the second cycle, so that the other byte-wise operations can be done seamlessly. Hence, computing the intermediate terms tmp_a, tmp_b can be done byte-wise in each cycle. The tmp_b bytes are pushed into the b_{15H} cell. The tmp_b byte is further xored with z before pushing into a_{15H}. Functionally this achieves the xoring of z to the 128 msbs of LFSR-A during initialization.

Rounds 14 to 15: In these rounds, the key bytes are additionally xored with $R1$. Since the xor is done after the σ operation, we need to load the bytes of the key in a shuffled according to the function σ^{-1}. We add these key bytes to tmp before they are introduced to $R1$. After the shuffle operation in the 16th cycle, we get $R1 = \sigma(tmp \oplus \sigma^{-1}(k)) = \sigma(tmp) \oplus k$ as required.

Rounds 16 onwards: This round onwards, the bytes of the keystream z are available as output. We discontinue the xoring of z to tmp_a as required in the specification. Also the addition of k to tmp is no longer required.

3.5 Other implementation details

As the LFSR granularity is 8 bits and multiplication is defined over $GF(2^{16})$, we need two cycles to execute this operation and exercise some fine grained control over multiplier circuit. In order to implement the bit shift for the multiplication with the regular root (i.e. α or β), we need an auxiliary d-flip-flop that holds the most significant bit of the lower byte $\{a, b\}_{0L}$. For example, assuming two adjacent byte registers $y(7\cdots0)$ and $x(7\cdots0)$ hold a_{0H} and a_{0L}, respectively, the first cycle

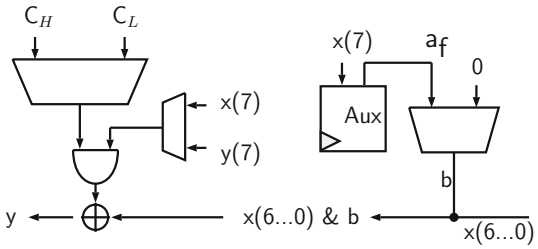


Fig. 4 Forward multiplication circuit. C_{L} and C_{H} denote the lower and upper half of the multiplication constant, and "&" denotes the concatenation operator. $x(7..0)$, $y(7..0)$ denote the current byte and next higher byte stored in the adjacent byte register (i.e. a_{0L}, a_{0H})

computes $[x(6\cdots0), 0] \oplus y(7) \cdot C_{\mathrm{L}}$ as the product output for the lower byte as per the multiplication rule explained earlier. Also it simultaneously stores the msb of a_{0L} located in $x(7)$ on the auxiliary flip-flop a_f. In the next cycle, due to data movement across the LFSR, a_{0H} displaces a_{0L} and occupies the registers locations $x(7\cdots0)$. It is now easy to see that the multiplication output for the higher byte is now $[x(6\cdots0),\ a_f] \oplus x(7) \cdot C_{\mathrm{H}}$. Similarly, the execution of the multiplication with the root inverse necessitates another 1-bit d-flip-flop storing the least significant bit of $\{a, b\}_{8H}$. The decision which byte of the constant (C_{L} or C_{H}) is utilized, can be performed through an 8-bit multiplexer, C_{L} for the lower byte and C_{H} for the lower byte in the next cycle. Figure 4 depicts a forward multiplication circuit.

The entire feedback function for one LFSR requires 58-bit XOR-gates, two within the multiplication circuits and three of them on the outside as part of the feedback function.

The construction of the byte-serial state machine is relatively straightforward. Three 8-bit multiplexers clear $R1$, $R2$ and $R3$ during the first round of the initialization. Another, together with an 8-bit XOR-gate, is needed to mix the key into $R1$ during the last two rounds. As the overall data path is 8 bits, it becomes necessary to emulate the four 32-bit adders per addition module by a single 8-bit ripple-carry adder. This necessitates a single-bit scan flip-flop that holds the carry bit between cycles. Note that in order to implement the σ-permutation, $R1$ consists of scan flip-flops instead of regular d-flip-flops as for $R2$ and $R3$. The finite-state machine circuit is depicted in Fig. 5.

3.6 Synthesis results

The byte-serial, dual-core, low-area SNOW-V architecture was synthesized using three common cell libraries, i.e. the STM and TSMC 90 nm processes and the relatively recent NanGate 15 nm process. The results are given in Table 4. We note the significant decrease in circuit area in comparison to all four proposed designs in the white paper. Also note that a low-latency library like NanGate 15 nm lifts the overall

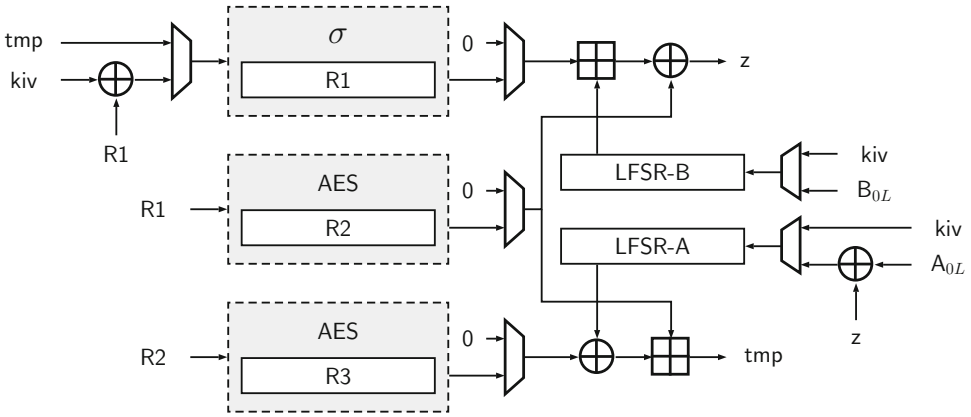


Fig. 5 Byte-serial low-area architecture. For the sake of simplicity, the control unit signals and the multiplication blocks within the LFSRs have been omitted

Table 4 Synthesis metrics for the low-area SNOW-V design tabulated for three widespread cell libraries

Library	Area (GE)	Delay (ns)	Power (mW)	Energy (μJ/4000 Blocks)	TP_{max} (Gbps)
STM 90 nm	4776	7.47	3.107	19.985	1.07
TSMC 90 nm	6210	6.67	0.179	1.153	1.19
NanGate 15 nm[1]	8227	0.18	0.068	0.436	44.7

The NanGate 15 nm process only supports d-flip-flops with resets; hence, no circuit area is gained through the usage of reset-less registers

throughput of the circuit above the 5G cut-off value of 20 Gbps.

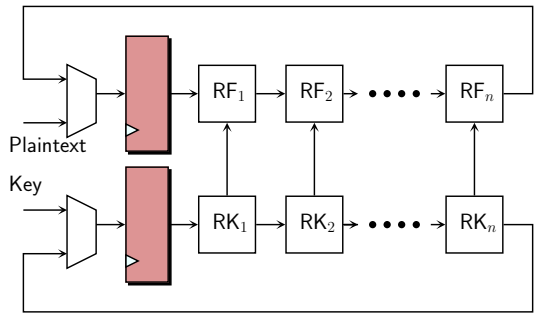
3.7 Alternative low-area designs

The next logical step, to further decrease the circuit area, would be to convert the design to a single-core module, i.e. containing only one Atomic-AES v2.0 engine. This would require two dedicated register banks for $R2$ and $R3$ as the registers within the encryption core instance cannot be reused anymore. It is an open question whether such a design improves the overall circuit area since the two additional registers occupy roughly the area currently held by the combination circuitry of one Atomic-AES v2.0 unit. However, in any case, the latency of the circuit will be doubled. Further marginal gains can be expected in a 1-bit serial configuration. For example, the bit-slide design by Jean et al. [9] has been specifically constructed for single-bit serial usage. On a 1-bit data path, we can shrink all the combinatorial components such as multiplexers and adders which results in further gains wrt circuit area. Nevertheless, on a bit-serial data path it is unlikely that the throughput can be kept above the 20 Gbps boundary.

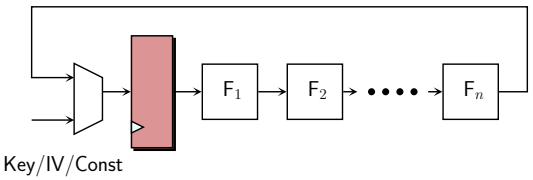
4 Low-energy/high-speed designs

We now focus on energy and speed optimization related aspects of the circuit design of SNOW-V, since they are equally important metrics for evaluating the merits of a particular cryptosystem. Particularly in the case of 5G, speed is a critical metric, as any crypto component employed in such systems is expected to produce output at a rate not lower than 20 Gbps.

The tradeoffs involved in optimizing cryptosystems for energy and speed have been studied extensively in [1,4]. In [1], the problem was first tackled for block ciphers. A block cipher operates by the repeated execution of a basic computation unit called a "round function" on the combination of the raw plaintext and secret key bits. In [1], the authors used round unrolling to increase the throughput of block ciphers, which is to say the circuits for multiple-round functions were stacked one after the other in order to get the circuit to compute the output of multiple round functions in a single clock cycle (Fig. 6a). If the block cipher specification calls for R round function executions, then an r-round unrolled circuit would compute the encryption in $1 + \lceil \frac{R}{r} \rceil$ cycles, assuming one cycle is required to load plaintext/key on to the respective registers. [1] showed that such increase in throughput by unrolling came with energy penalties. By studying the power consumption model of CMOS gates, it was proven that the power consumed in an r-round unrolled block cipher (encryption only) was a quadratic in r. Since there are r copies of the round function, the glitches produced due to signal delays in the i^{th} round function, are compounded in the $(i+1)^{st}$ round function and is compounded further in the $(i+2)^{nd}$ round function. Thus, in a sense the logic blocks at a lower depth produce fewer glitches and hence consume lower power. In fact, it was shown that the power consumed in each successive round function formed a simple arithmetic



(a) Block Cipher



(b) Stream Cipher

Fig. 6 Unrolled architectures of block and stream ciphers. RF_i/RK_i denote the ith round function/key update function used by the block cipher. F_i is simply the state update logic employed by the stream cipher

sequence. Since the total power consumed is a sum of these r terms of the sequence, it results in a quadratic in r. Thus, the total energy consumption in such circuits was a quasi-quadratic in r of the form:

$$(Ar^2 + Br + c) \cdot \left(1 + \lceil * \rceil \frac{R}{r}\right).$$

It was shown that for most lightweight ciphers with low critical path, $r = 2$ was the most energy efficient configuration, while $r = 1$ was optimal for the more heavyweight ciphers like AES. In both cases, boosting throughput by increasing the value of r beyond 2, proved counterproductive for energy. This is due to the simple reason that the gain in decrease in number of clock cycles (which is inverse in the first power of r) can no longer compensate for the compounding of the glitches produced thereof. As a result, the dynamic power increases rapidly (proportional to the second power of r) and increases energy consumption.

In [4], it was observed that a stream cipher functions much in the same way as a block cipher. Unlike a block cipher which after encryption, maps a raw plaintext to another element of the same domain, a stream cipher is essentially a random bit generator whose output is xored to the plaintext bits to produce ciphertext bits. A stream cipher takes a binary key and initialization vector (IV) as input and first sets up a finite state machine, which updates itself by an update function at every clock cycle and produces random keystream bits simultaneously (Fig. 6b). Similarly as in a block cipher, it is possible to boost throughput, i.e. produce more keystream bits per cycle by unrolling update functions. From the figure, we can already see parallels between the hardware architectures of unrolled block and stream ciphers. The role of round functions is simply replaced by state update functions in a stream cipher. The differences, however, lie in the minute details: for example, block ciphers must execute the recommended R rounds to encrypt data; thus, each encryption cycle must necessarily consist of $\left(1 + \lceil * \rceil \frac{R}{r}\right)$ clock cycles and would encrypt at most b bits (where b is the size in bits of the plaintext domain, e.g. $R = 10$, $b = 128$ for AES-128). On the other hand, an r round unrolled stream cipher, after an initialization phase, produces r keystream bits ready for xoring to plaintext in every clock cycle. For example, a single-round unrolled implementation of AES would take 2200 cycles to encrypt 200 blocks (12800 bits) of data. However, a 64-round unrolled Trivium (which occupies around half the area of AES) encrypts 12800 bits in $\frac{1152+12800}{64} = 219$ clock cycles.

Thus, in an asymptotic sense, for the same silicon budget, the time required to encrypt the same amount of data is significantly smaller in stream ciphers. It was due to this, that it was noted in [4] that most stream ciphers are more energy-efficient than any block cipher mode for encryption of longer data streams. Also some stream ciphers have extremely simple update functions consisting of a bit rotate and a Boolean function, and so unrolling multiple times does not necessarily compound glitch formation in the same amount as in block ciphers. As a result, [4] reported the optimal energy configuration of the stream cipher Trivium [6] at $r = 160$ for the STM 90 nm CMOS process.

4.1 Circuit details

When we evaluate SNOW-V from a circuit designer's point of view, we know two things.

1. First that unrolling would be a good generic technique to boost throughput, but it is likely to come with heavy energy penalty at least after a point. The update function of the SNOW-V finite state machine includes the quite heavyweight AES round function, and so it is unlikely that after unrolling the circuit $r = 160$ times, an energy consumption optima would be reached, as in Trivium.
2. To further increase speed and decrease energy, we have to look at micro-level architectures of the individual components of the circuit. There are four principal components of this circuit: **(a)** the shift registers, **(b)** the constant multipliers over $GF(2^{16})$, **(c)** the AES round functions and **(d)** the adders over $\mathbb{Z}_{2^{32}}$.

Of the above there is not much architectural optimization one could do for **(a)**, **(b)**: since the registers are essentially composed of the flip-flops provided by the corresponding

standard cell library and the constant multipliers are essentially linear circuits composed of solely xor gates. Thus, any optimization would naturally need to target **(c)**, **(d)**.

Before we proceed, let us give a clear idea of the circuit of SNOW-V that we experiment with. We essentially follow the basic top-level circuit structure shown in Fig. 6b. A single-round $r = 1$ architecture would consist of the following:

A: The state register consists of the thirty-two 16-bit linear-feedback shift registers A, B, and the three 128-bit registers $R1$, $R2$, $R3$.
B: In the 1st clock cycle, the state is loaded with the binary string $k_7, \ldots, k_0, iv_7, \ldots, iv_0 \| k_{15}, k_{14}, \ldots, k_8, 0^{64} \| 0^{128} \| 0^{128} \| 0^{128}$, where k_i, iv_i are the ith 16-bit blocks of the key, IV. This is done by assigning the select signal in the multiplexer before the state register to accept the above binary string. Thereafter, the select is deactivated, and the registers update themselves only with the output of the state update functions.
C: Each update function $F_i : \{0,1\}^{896} \rightarrow \{0,1\}^{896}$ computes the following in a single clock. It first computes $z = (R1 \boxplus_{32} T1) \oplus R2$, where $T1$ is the most significant 128 bits of the register B. It updates the registers $R1$, $R2$, $R3$ and updates the LFSRs A, B 8 times as given in Algorithm 3. Other additional functionalities like xoring z are xored to A during initialization, and xoring most (resp. least) significant bits of the key to $R1$ in the last (resp. second last) round of initialization is also done.
D: An r-round unrolled circuit would contain r copies of the update function circuit $F_1, F_2, \ldots, F_r$. The initialization is completed in $\lceil * \rceil \frac{16}{r}$ cycles, and each clock cycle produces $128r$ bits of keystream.

4.2 Component selection: high speed

To do a micro-level optimization, we started with the simplest possible form of the circuit at $r = 1$. Our intention was to find specific circuit level optimizations for the various components that would optimize metrics like energy and speed for the simplest possible architecture. We then take these optimized components and construct the state update functions for the purpose of unrolling multiple rounds. This is a reasonable tactic, since a well-constructed update function architecture is most certainly to produce better results when used as underlying building blocks for multiple-round unrolled architectures.

To increase the throughput, one natural optimization is to decrease the total critical path τ of the circuit, which would enable the circuit to operate at the maximum frequency of $f_{\max} = \frac{1}{\tau}$. In the original SNOW-V paper [7], the authors had conjectured that the primary contribution to the critical path of the circuit is likely due to be the AES round function. There are a couple of ways to reduce the critical path of the AES round function:

– A convenient way to synthesize any S-box is to implement it as a look-up table (LUT) and present it to the circuit compiler. However, it is also possible to take advantage of the algebraic structure in the AES S-box to get a more compact implementation (namely it is an affine function computed over the inverse in the AES finite field i.e $GF(2^8)/ < x^8 + x^4 + x^3 + x + 1 >$). Very recently in [14], this approach has been rigorously studied. As a result, three different implementations of the S-box were reported. The first (call it S_1) was the most compact, the second (S_2) had the least circuit depth (hence critical path) and a third (S_3) is an implementation in between S_1 and S_2. Since this represents the state of the art, we decided to experiment with all these different architectures.
– A straightforward implementation of the Mixcolumn (M_1) of this circuit requires 108 xor gates [1]. In a recent article, an implementation with 92 xor gates (M_2) was proposed [13]. This represents the most compact implementation of the Mixcolumn circuit known till date. In [12], the authors propose a circuit with 105 gates but low circuit depth (M_3). To do a fair comparison, we included all of these circuits in our experiments.
– Another approach used quite often to get fast software implementations is using T-tables which are a set of look-up tables, that combine the S-box and Mixcolumn operations. The AES T-tables are essentially a set of four look-up tables T_0, T_1, T_2, $T_3 : \{0,1\}^8 \rightarrow \{0,1\}^{32}$, with

$$T_0(x) = [2 \cdot S(x),\ 1 \cdot S(x),\ 1 \cdot S(x),\ 3 \cdot S(x)]$$
$$T_1(x) = [3 \cdot S(x),\ 2 \cdot S(x),\ 1 \cdot S(x),\ 1 \cdot S(x)]$$
$$T_2(x) = [1 \cdot S(x),\ 3 \cdot S(x),\ 2 \cdot S(x),\ 1 \cdot S(x)]$$
$$T_3(x) = [1 \cdot S(x),\ 1 \cdot S(x),\ 3 \cdot S(x),\ 2 \cdot S(x)]$$

Here, $S(x)$ denotes the S-box mapping. If x_0, x_1, x_2, x_3 are the 4 bytes in a column after the shiftrow operation, it is not too difficult to see that $\sum_i T_i(x_i)$ is the AES round function output for that column. Since the circuit combines the S-box and part of the Mixcolumn operation, it is natural to try to implement this circuit to minimize circuit depth and hence critical path. However, the downsides of this approach are obviously the large silicon area required to construct four 8- to 32-bit tables, which would have a negative effect on not only area but also power and energy consumption.

However, when we constructed the SNOW-V circuit with all possible mix and match of the above component architectures, for multiple standard cell libraries, we observed that the

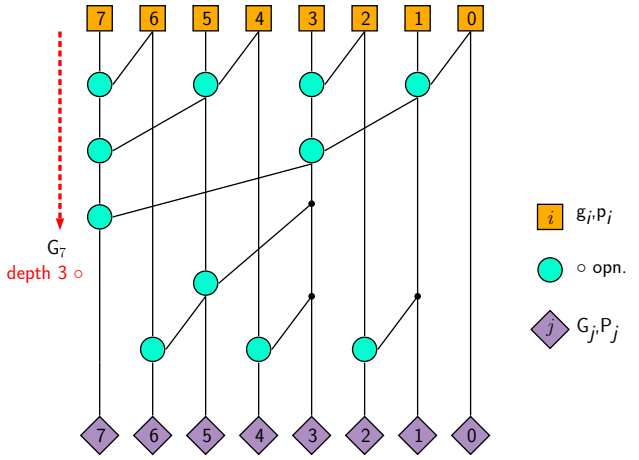


Fig. 7 8-bit Brent–Kung adder architecture. The final stage carry G_7 carry requires 3° operations. The orange boxes and the purple rhombuses denote the g_i, p_i and G_i, P_i signals. The green circles denote the ∘ operation

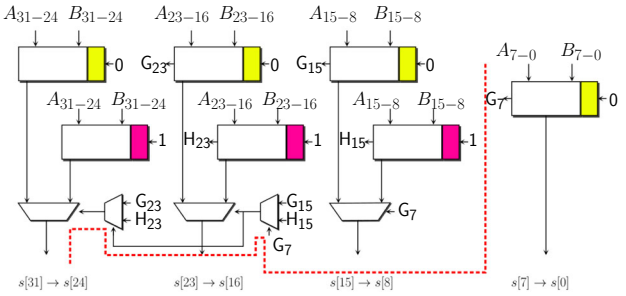


Fig. 8 Hybrid adder architecture combining the Brent–Kung and Carry select approaches. the path in the red dotted line represents the critical path of the circuit. The rectangular blocks coloured yellow/pink on the side represent 8-bit Brent–Kung adders which take 0/1 as carry in. G_i/H_i represent the final carry-outs generated by the respective byte adders

critical path is not always necessarily due to the AES round function. When the circuit synthesizer is asked to a multiple target optimization using the `compile_ultra` option, the compiler tries to concurrently optimize timing, power and area. It does so by using the vast array of gates and flip-flops available to it in the standard cell library. Now depending on the functionality and physical/electrical characteristics of the gates in the library the compiler may be able to optimize one part of the circuit better than the others. For example, while implementing the SNOW-V circuit in the TSMC 90 nm library we found that the 32-bit adder constituted the critical path of the circuit, whereas in the STM 90 nm library the AES round function constituted the critical path. Also in circuits compiled with both the libraries, timing delays of the AES round function and the 32-bit adder were not widely different, which led us to the opinion that it was important to optimize both these components for speed.

When no optimization for the adder is written into the RTL code, the circuit compiler invariably synthesizes it as a ripple-carry adder which is known to have circuit depth linear in the size of the adder. However, computer literature provides us with numerous adder architectures with logarithmic delay. One example is the Brent–Kung (A1) topology [5]. Although this type of adder is well known, we give a preliminary description for the benefit of the reader. If $a_0, a_1, \ldots, a_{n-1}$ and $b_0, b_1, \ldots, b_{n-1}$ are the two bit strings that are to be added, then the circuit first computes the signals $g_i = a_i \cdot b_i,\ p_i = a_i \oplus b_i,\ \forall\, i \in [0, n-1]$. An operation ∘ is defined as follows: $(x_1, y_1) \circ (x_2, y_2) = (x_1 \vee (y_1 \cdot x_2), y_1 \cdot y_2)$. Two strings of variables G_i, P_i are then defined as:

$$(G_i, P_i) = \begin{cases} (g_i, p_i), & \text{if } i = 0, \\ (g_i, p_i) \circ (G_{i-1}, P_{i-1}), & \text{otherwise} \end{cases}$$

For n being a power of 2, G_{n-1}, P_{n-1} can be computed in a tree-like manner as shown in Fig. 7, by using the associativity of the ∘ operator. It is not difficult to see that the carry and the sum generated at the ith stage is G_i and $P_i \oplus G_{i-1}$, respectively. The height of the tree is $log_2(n)$ which is therefore the circuit depth for computing G_{n-1}. Although the final carry-out of the adder, i.e. G_{n-1}, is computed in logarithmic time, for generating the intermediate sum/carry bits more ∘ operations are required. In fact for $n = 32$, the total circuit depth from the initial g_i, p_i signals to the final G_i, P_i, $\forall\, i$ signals around 8° operations. Assuming that the delay in computing ∘ is around 2 units and the delay in each xor/and is 1 unit, then the total depth in this topology is 1 (initial xor/and needed to generate g_i/p_i) + 8 ∗ 2 + 1 (to generate the sum $P_i \oplus G_{i-1}$) = 18 units.

Since for any n equal to a power of 2, G_{n-1} only takes logarithmic time, it is therefore best to combine this topology with the Carry-Select architecture, to decrease the overall critical path, albeit at the cost of a little more silicon area. The carry bit G_7 of an 8-bit Brent–Kung adder can be generated in 3° operations. Combining it with the Carry-Select architecture would require seven 8-bit block adders as shown in Fig. 8. The total critical path as shown by the dotted red line in Fig. 8 is equal to the time taken to compute G_7 and the time taken to ripple through three multiplexers till the most significant byte of the sum is calculated. Assuming that each multiplexer has two-unit delay (this is a reasonable hypothesis, since in the absence of a dedicated multiplexer in the library, a circuit compiler would combine basic NAND gates to construct it, ex: (a NAND s) NAND (b NAND s′)), the total delay in this architecture is only $1 + 3 * 2 + 3 * 2 = 13$ units. We tried this hybrid architecture with three different standard cell libraries (TSMC 90 nm, STM 90 nm and NanGate 15 nm). In all the three libraries, the hybrid architecture exhibited around 25% reduction in critical path over a 32-bit Brent–Kung adder circuit.

The Brent–Kung adder is predated by a similar construction by Kogge and Stone that uses more ∘ components to reduce the fan-out of the nets, which naturally increases the

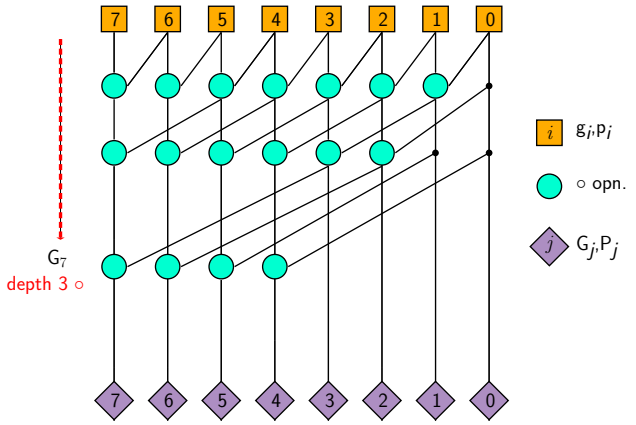


Fig. 9 8-bit Kogge–Stone adder architecture. The final stage carry G_7 carry requires 3° operations. The orange boxes and the purple rhombuses denote the g_i, p_i and G_i, P_i signals. The green circles denote the ∘ operation

circuit area [11]. More specifically, instead of arranging the ∘ in a binary-tree fashion as shown in Fig. 7, the Kogge–Stone circuits add computes a ∘ signal for all adjacent wires. A diagram for an 8-bit Kogge–Stone adder is depicted in Fig. 9. For the 32-bit variant, note that the MSB will have a critical path delay of 5° operations, which makes the total delay of these adders equal to $1 + 2 * 5 + 1$ (for the final xor) $= 12$ units.

4.3 Component selection: low energy

One of the conclusions of [1] was that the most energy/intensive operation in the AES circuit was the S-box layer. The paper also concluded that the most energy efficient architecture of the S-box circuit could be realized by the Decode–Switch–Encode (DSE) architecture (S4) which is a generic way to construct any $n \rightarrow m$-bit function for arbitrary integers m, n. For $n = m = 8$, as in the case of the AES S-box, the circuit first decodes the n input bits to produce 2^n parallel signals such that only the tth signal is logically HIGH and the rest are all LOW (where t is the integer equivalent of the n-bit binary input). Such a set of signals is also sometimes referred to as "1-hot lines" for obvious reasons. Then, the shuffle layer permutes the 2^n lines as per the function mapping: for the case of the AES S-box $S(\cdot)$, each tth line is shifted to $S(t)$th position ($\forall\, t$). The encoder is logically the inverse of the decoder operation which converts the 2^n signals back to n output lines. It was shown in [1] that such circuits have very low switching probabilities, with only 25% gates likely to switch per change of input bit. Hence, such circuits are remarkably energy efficient.

However, all the other atomic components of the SNOW-V circuit (such as Mixcolumn/adders/constant multipliers) take at least 16 bits of input or more. Thus means that to employ the DSE philosophy to implement the other circuit components would require exponential circuit complexity to construct only the 2^n lines of the Decoder layer. This is heavily counterproductive from an energy consumption and area point of view, and thus, as far as energy consumption is concerned, we did not tinker with any other circuit component.

5 Results: discussion

5.1 First dataset

We first present synthesis results for the single-round unrolled circuit ($r = 1$) for three standard cell libraries TSMC 90 nm, STM 90 nm and NanGate 15 nm. We have experimented with four different kinds of AES S-box circuits, three Mixcolumn circuits, a T-table circuit that combines these two functionalities, and two different kinds of adder circuits. The following design flow was adhered to: first the design was implemented in VHDL. Then, a functional verification at the RTL level was first done using Mentor Graphics Modelsim software. The designs were synthesized using the three libraries just mentioned, with the Synopsys Design Compiler, with the compiler optimization set to `compile_ultra`. 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. Note that the frequency of operation was fixed at 10 MHz because as established in [1,10], at high frequencies the energy consumption of block ciphers is frequency independent. 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 "Total Energy" is calculated as the product of average power and time taken for producing 4000 blocks of keystream. We also provide an entry for the normalized average energy required to produce one keystream block.

The initial results are presented in Tables 5, 6 and 7. This represents a huge trove of data spanning one and a half pages. However, it is not too difficult to analyse the data once the results are isolated for each library. For the TSMC 90 nm library, the results were along expected lines, the Kogge–Stone adder architectures always have a lower critical path, whereas the DSE S-box architecture is most energy efficient.

For the STM 90 nm library, there is more variance in the critical path with change in the AES architecture and does not seem to be affected with change in adder architecture. The best configuration in terms of both energy and speed is the implementation with DSE S-box, a simple functional Mixcolumn circuit (M_1) and a Brent–Kung adder architecture. It is also understandable that the Brent–Kung architecture tends to consume lesser power/energy than the hybrid architecture

Table 5 Synthesis results of the SNOW-V circuit for the TSMC 90 nm library. Power computed at 10 MHz. "Total Energy" represents the energy required to produce 4000 blocks of keystream

A: TSMC 90 nm Library

S-box	Mixcolumn	Adder	Area (GE)	Core latency (ns)	Max TPUT (Gbit/s)	Power (mW)	Total energy (nJ)	Energy/block (pJ/128 bits)
LUT	Low depth (M_3)	Brent	28048	6.21	20.27	0.470	191.155	47.789
LUT	Low depth (M_3)	Hybrid	28413	4.97	25.32	0.478	194.572	48.643
LUT	Low depth (M_3)	K-Stone	29590	3.8	33.12	0.4811	195.711	48.928
LUT	Smallest (M_2)	Brent	28044	6.21	20.27	0.479	194.735	48.684
LUT	Smallest (M_2)	Hybrid	28416	4.97	25.32	0.487	198.234	49.558
LUT	Smallest (M_2)	K-Stone	29583	2.9	43.40	0.4899	199.291	49.823
LUT	Simple (M_1)	Brent	28047	6.21	20.27	0.471	191.399	47.850
LUT	Simple (M_1)	Hybrid	28412	4.97	25.32	0.479	194.817	48.704
LUT	Simple (M_1)	K-Stone	29586	2.82	44.63	0.4816	195.915	48.979
Fast (S_2)	Low depth (M_3)	Brent	18670	6.12	20.57	0.575	233.991	58.498
Fast (S_2)	Low depth (M_3)	Hybrid	18945	5.46	23.05	0.581	236.188	59.047
Fast (S_2)	Low depth (M_3)	K-Stone	20208	4.12	30.55	0.5678	230.981	57.745
Fast (S_2)	Smallest (M_2)	Brent	18620	6.10	20.63	0.550	223.577	55.894
Fast (S_2)	Smallest (M_2)	Hybrid	19027	5.36	23.48	0.592	240.704	60.176
Fast (S_2)	Smallest (M_2)	K-Stone	20143	4.22	29.82	0.5629	228.988	57.247
Fast (S_2)	Simple (M_1)	Brent	18654	6.08	20.70	0.574	233.340	58.335
Fast (S_2)	Simple (M_1)	Hybrid	18918	5.46	23.05	0.546	221.991	55.498
Fast (S_2)	Simple (M_1)	K-Stone	20176	5	25.17	0.5785	235.334	58.833
Compact (S_1)	Low depth (M_3)	Brent	20873	6.21	20.27	0.652	265.356	66.339
Compact (S_1)	Low depth (M_3)	Hybrid	21088	5.44	23.14	0.649	264.176	66.044
Compact (S_1)	Low depth (M_3)	K-Stone	22400	4.54	27.72	0.6758	274.915	68.729
Compact (S_1)	Smallest (M_2)	Brent	20874	6.20	20.30	0.678	275.770	68.942
Compact (S_1)	Smallest (M_2)	Hybrid	21207	5.36	23.48	0.665	270.685	67.671
Compact (S_1)	Smallest (M_2)	K-Stone	22359	4.73	26.61	0.6866	279.309	69.827
Compact (S_1)	Simple (M_1)	Brent	20816	6.10	20.63	0.639	259.782	64.946
Compact (S_1)	Simple (M_1)	Hybrid	20998	5.59	22.52	0.650	264.257	66.064
Compact (S_1)	Simple (M_1)	K-Stone	22254	4.58	27.48	0.6744	274.346	68.586
Tradeoff (S_3)	Low depth (M_3)	Brent	20667	6.10	20.63	0.637	259.254	66.339
Tradeoff (S_3)	Low depth (M_3)	Hybrid	20799	5.15	24.44	0.642	260.962	66.044
Tradeoff (S_3)	Low depth (M_3)	K-Stone	22038	4.69	26.84	0.6464	262.956	65.739
Tradeoff (S_3)	Smallest (M_2)	Brent	20598	6.13	20.53	0.647	263.118	68.942
Tradeoff (S_3)	Smallest (M_2)	Hybrid	20686	5.24	24.02	0.643	261.613	67.671
Tradeoff (S_3)	Smallest (M_2)	K-Stone	21983	4.48	28.09	0.6331	257.545	64.386
Tradeoff (S_3)	Simple (M_1)	Brent	20593	6.10	20.63	0.634	257.911	64.946
Tradeoff (S_3)	Simple (M_1)	Hybrid	20805	5.64	22.32	0.631	256.650	66.064
Tradeoff (S_3)	Simple (M_1)	K-Stone	22139	4.56	27.60	0.6287	255.755	63.939
DSE (S_4)	Low depth (M_3)	Brent	29022	6.16	20.43	0.337	137.051	**34.263**
DSE (S_4)	Low depth (M_3)	Hybrid	29385	4.94	25.48	0.345	140.387	35.097
DSE (S_4)	Low depth (M_3)	K-Stone	30560	2.75	**45.77**	0.3489	141.933	35.483
DSE (S_4)	Smallest (M_2)	Brent	29017	6.15	20.47	0.342	139.126	34.781
DSE (S_4)	Smallest (M_2)	Hybrid	29389	4.94	25.48	0.350	142.502	35.626

Table 5 continued

A: TSMC 90 nm Library

S-box	Mixcolumn	Adder	Area (GE)	Core latency (ns)	Max TPUT (Gbit/s)	Power (mW)	Total energy (nJ)	Energy/block (pJ/128 bits)
DSE (S_4)	Smallest (M_2)	K-Stone	30560	2.81	44.79	0.3542	144.089	36.022
DSE (S_4)	Simple (M_1)	Brent	29022	6.15	20.47	0.337	137.214	34.303
DSE (S_4)	Simple (M_1)	Hybrid	29390	4.94	25.48	0.345	140.509	35.127
DSE (S_4)	Simple (M_1)	K-Stone	30572	3.39	37.13	0.3494	142.136	35.534
T-Table		Brent	47832	6.16	20.43	0.472	191.888	47.972
T-Table		Hybrid	48212	4.94	25.48	0.480	195.223	48.806
T-Table		K-Stone	49379	3.21	39.21	0.4835	196.688	49.172

Table 6 Synthesis results of the SNOW-V circuit for the STM 90 nm library

B: STM 90 nm Library

S-box	Mixcol	Adder	Area (GE)	core latency (ns)	Max TPUT (Gbit/s)	Power (mW)	Total Energy (nJ)	Energy/block (pJ/128-bit)
LUT	Low depth (M_3)	Brent	26111	4.35	28.93	1.143	465.013	116.253
LUT	Low depth (M_3)	Hybrid	26347	4.69	26.84	1.150	467.901	116.975
LUT	Low depth (M_3)	K-Stone	26692	4.35	28.93	1.1678	475.061	118.765
LUT	Smallest (M_2)	Brent	26114	4.38	28.74	1.150	467.861	116.965
LUT	Smallest (M_2)	Hybrid	26353	4.47	28.16	1.155	470.017	117.504
LUT	Smallest (M_2)	K-Stone	26662	4.49	28.03	1.1728	477.095	119.274
LUT	Simple (M_1)	Brent	26133	4.06	31.00	1.140	463.711	115.928
LUT	Simple (M_1)	Hybrid	26375	4.31	29.20	1.145	465.745	116.436
LUT	Simple (M_1)	K-Stone	26707	4.05	31.08	1.1602	471.969	117.992
Fast (S_2)	Low depth (M_3)	Brent	16774	5.59	22.52	1.259	511.958	127.989
Fast (S_2)	Low depth (M_3)	Hybrid	17011	5.63	22.36	1.268	515.863	128.966
Fast (S_2)	Low depth (M_3)	K-Stone	17340	5.61	22.44	1.2748	518.589	129.647
Fast (S_2)	Smallest (M_2)	Brent	16774	5.57	22.60	1.264	514.195	128.549
Fast (S_2)	Smallest (M_2)	Hybrid	17011	5.58	22.56	1.274	518.100	129.525
Fast (S_2)	Smallest (M_2)	K-Stone	17340	5.57	22.60	1.2804	520.867	130.217
Fast (S_2)	Simple (M_1)	Brent	16800	5.50	22.88	1.255	510.493	127.623
Fast (S_2)	Simple (M_1)	Hybrid	17037	5.54	22.72	1.265	514.439	128.610
Fast (S_2)	Simple (M_1)	K-Stone	17366	5.52	22.80	1.2713	517.165	129.291
Compact (S_1)	Low depth (M_3)	Brent	16382	5.42	23.22	1.159	471.278	117.819
Compact (S_1)	Low depth (M_3)	Hybrid	16619	5.46	23.05	1.173	477.014	119.253
Compact (S_1)	Low depth (M_3)	K-Stone	16948	5.42	23.22	1.1833	481.366	120.342
Compact (S_1)	Smallest (M_2)	Brent	16382	5.44	23.14	1.165	473.719	118.430
Compact (S_1)	Smallest (M_2)	Hybrid	16619	5.44	23.14	1.179	479.414	119.853
Compact (S_1)	Smallest (M_2)	K-Stone	16948	5.44	23.14	1.1891	483.726	120.931
Compact (S_1)	Simple (M_1)	Brent	16408	5.32	23.66	1.156	470.139	117.535
Compact (S_1)	Simple (M_1)	Hybrid	16645	5.36	23.48	1.169	475.427	118.857
Compact (S_1)	Simple (M_1)	K-Stone	16974	5.35	23.53	1.1792	479.699	119.925
Tradeoff (S_3)	Low depth (M_3)	Brent	16248	4.66	27.01	1.085	441.215	117.819
Tradeoff (S_3)	Low depth (M_3)	Hybrid	16485	4.69	26.84	1.098	446.504	119.253

Table 6 continued

B: STM 90 nm Library

S-box	Mixcol	Adder	Area (GE)	core latency (ns)	Max TPUT (Gbit/s)	Power (mW)	Total Energy (nJ)	Energy/block (pJ/128-bit)
Tradeoff (S_3)	Low depth (M_3)	K-Stone	16812	4.66	27.01	1.1075	450.531	112.633
Tradeoff (S_3)	Smallest (M_2)	Brent	16248	4.60	27.36	1.088	442.680	118.430
Tradeoff (S_3)	Smallest (M_2)	Hybrid	16485	4.62	27.24	1.101	447.968	119.853
Tradeoff (S_3)	Smallest (M_2)	K-Stone	16812	4.62	27.24	1.1111	451.995	112.999
Tradeoff (S_3)	Simple (M_1)	Brent	16274	4.64	27.13	1.085	441.419	117.535
Tradeoff (S_3)	Simple (M_1)	Hybrid	16511	4.67	26.95	1.098	446.748	118.857
Tradeoff (S_3)	Simple (M_1)	K-Stone	16838	4.65	27.07	1.108	450.734	112.684
DSE (S_4)	Low depth (M_3)	Brent	28393	3.24	38.85	0.611	248.514	62.129
DSE (S_4)	Low depth (M_3)	Hybrid	28640	3.33	37.80	0.622	252.948	63.237
DSE (S_4)	Low depth (M_3)	K-Stone	28969	3.28	38.37	0.6321	257.138	64.285
DSE (S_4)	Smallest (M_2)	Brent	28393	3.24	38.85	0.613	249.206	62.301
DSE (S_4)	Smallest (M_2)	Hybrid	28644	3.39	37.13	0.623	253.436	63.359
DSE (S_4)	Smallest (M_2)	K-Stone	28969	3.24	38.85	0.6335	257.708	64.427
DSE (S_4)	Simple (M_1)	Brent	28419	3.16	**39.83**	0.609	247.904	**61.976**
DSE (S_4)	Simple (M_1)	Hybrid	28666	3.33	37.80	0.620	252.175	63.044
DSE (S_4)	Simple (M_1)	K-Stone	28995	3.18	39.58	0.6303	256.406	64.102
T-Table		Brent	41076	3.75	33.56	1.054	428.686	107.171
T-Table		Hybrid	41250	3.72	33.83	1.067	433.934	108.483
T-Table		K-Stone	41562	3.69	34.11	1.084	440.971	110.243

Power computed at 10 MHz. "Total Energy" represents the energy required to produce 4000 blocks of keystream

since it uses lesser silicon resources, but the difference is not very large. In the hybrid architecture, most of the glitching occurs in the adders which are obviously at a lower depth than the full block Brent–Kung adder, and so as explained at the beginning of the previous section, this results in lower dynamic power/energy consumption and so both adder architectures have relatively similar power/energy budgets.

For the NanGate 15 nm library, the core latency is almost constant across the table, and the most energy efficient architectures are also the ones with lowest area. The results are extremely interesting in as much we found that the critical path was consumed by the controller circuit of the stream cipher. Since the underlying transistors constructing the standard cells of this library are composed of transistors with extremely small feature size (15 nm), these standard cells are extremely fast. A clock input of 10 MHz (i.e. period 100 ns) causes the synthesizer to construct the control circuit suboptimally since the total delay across it is only around 300 ps and since this is much much smaller than the clock period, the synthesizer does not really put any effort behind optimizing this circuit component. However, in the later experiments, which we will shortly describe, we used a higher frequency clock with period close to the critical path. Once we did that, the synthesizer did optimize the control circuit and returned a netlist in which the critical path was again due to the adder.

Being a low feature size library, the effect of leakage power in NanGate 15 nm circuits is also more pronounced. Since the amount of leakage power is directly proportional to the silicon area of the circuit, and independent of frequency of the clock, circuits with low total area also tend to consume low leakage power. For example, in the SNOW-V circuit with the compact S-box (S_1), the smallest Mixcolumn M_2 and the hybrid adder the contribution of the leakage power to the total power consumption is close to 50 %. The same figure is less than 5% for the other libraries. The leakage power is primarily the reason why low area circuits for the NanGate library consume less energy at 10 MHz.

However, as the frequency of the clock increases, the total physical time taken to compute an equivalent amount of keystream decreases proportional to the clock period (since the total time is just some integer multiple of the clock period). When this happens, the contribution of the leakage power to the total energy consumed (which is the product of the frequency independent leakage power and the physical time) also decreases proportionally. This is not the case for the dynamic component of the power consumption (which is really a measure of the amount of $0 \rightarrow 1/1 \rightarrow 0$ transitions of the circuit per unit time). It is generally well known that the dynamic component of the power consumption varies directly as the clock frequency of the circuit and

Table 7 Synthesis results of the SNOW-V circuit for the NanGate 15 nm library

C: NanGate 15 nm Library								
S-box	Mixcol	Adder	Area (GE)	Core latency (ps)	Max TPUT (Gbit/s)	Power (mW)	Total energy (nJ)	Energy/block (pJ/128-bit)
LUT	Low depth (M_3)	Brent	35380	298.88	421.11	0.278	113.172	28.293
LUT	Low depth (M_3)	Hybrid	35864	298.87	421.12	0.274	111.504	27.876
LUT	Low depth (M_3)	K-Stone	36264	298.88	421.11	0.2837	115.409	28.852
LUT	Smallest (M_2)	Brent	35395	298.88	421.11	0.278	113.172	28.293
LUT	Smallest (M_2)	Hybrid	35879	298.87	421.12	0.274	111.504	27.876
LUT	Smallest (M_2)	K-Stone	36279	298.87	421.12	0.2837	115.409	28.852
LUT	Simple (M_1)	Brent	35438	298.88	421.11	0.280	113.945	28.486
LUT	Simple (M_1)	Hybrid	35982	298.88	421.11	0.276	112.277	28.069
LUT	Simple (M_1)	K-Stone	36322	298.87	421.12	0.2856	116.182	29.046
Fast (S_2)	Low depth (M_3)	Brent	23980	298.87	421.12	0.207	84.086	21.021
Fast (S_2)	Low depth (M_3)	Hybrid	24464	298.88	421.11	0.203	82.377	20.594
Fast (S_2)	Low depth (M_3)	K-Stone	24864	298.87	421.12	0.2121	86.282	21.571
Fast (S_2)	Smallest (M_2)	Brent	23996	298.87	421.12	0.207	84.086	21.021
Fast (S_2)	Smallest (M_2)	Hybrid	24480	298.88	421.11	0.203	82.377	20.594
Fast (S_2)	Smallest (M_2)	K-Stone	24880	298.88	421.11	0.2122	86.323	21.581
Fast (S_2)	Simple (M_1)	Brent	24038	298.88	421.11	0.209	84.858	21.215
Fast (S_2)	Simple (M_1)	Hybrid	24522	298.88	421.11	0.204	83.150	20.787
Fast (S_2)	Simple (M_1)	K-Stone	24922	298.88	421.11	0.2141	87.096	21.774
Compact (S_1)	Low depth (M_3)	Brent	22948	298.88	421.11	0.197	80.018	20.004
Compact (S_1)	Low depth (M_3)	Hybrid	23432	298.88	421.11	0.193	78.309	19.577
Compact (S_1)	Low depth (M_3)	K-Stone	23832	298.88	421.11	0.2021	82.214	20.554

Table 7 continued

C: NanGate 15 nm Library

S-box	Mixcol	Adder	Area (GE)	Core latency (ps)	Max TPUT (Gbit/s)	Power (mW)	Total energy (nJ)	Energy/block (pJ/128-bit)
Compact (S_1)	Smallest (M_2)	Brent	22964	298.87	421.12	0.197	80.018	20.004
Compact (S_1)	Smallest (M_2)	Hybrid	23448	298.88	421.11	0.193	78.309	**19.577**
Compact (S_1)	Smallest (M_2)	K-Stone	23848	298.88	421.11	0.2021	82.214	20.554
Compact (S_1)	Simple (M_1)	Brent	23006	298.88	421.11	0.199	80.790	20.198
Compact (S_1)	Simple (M_1)	Hybrid	23490	298.88	421.11	0.194	79.082	19.770
Compact (S_1)	Simple (M_1)	K-Stone	23890	298.88	421.11	0.204	82.987	20.747
Tradeoff (S_3)	Low depth (M_3)	Brent	22764	298.87	421.12	0.195	79.367	20.004
Tradeoff (S_3)	Low depth (M_3)	Hybrid	23248	298.87	421.12	0.191	77.617	19.577
Tradeoff (S_3)	Low depth (M_3)	K-Stone	23648	298.88	421.11	0.2005	81.563	20.391
Tradeoff (S_3)	Smallest (M_2)	Brent	22780	298.87	421.12	0.195	79.367	20.004
Tradeoff (S_3)	Smallest (M_2)	Hybrid	23264	298.88	421.11	0.191	77.658	19.577
Tradeoff (S_3)	Smallest (M_2)	K-Stone	23664	298.87	421.12	0.2005	81.563	20.391
Tradeoff (S_3)	Simple (M_1)	Brent	22822	298.88	421.11	0.197	80.140	20.198
Tradeoff (S_3)	Simple (M_1)	Hybrid	23306	298.88	421.11	0.193	78.390	19.770
Tradeoff (S_3)	Simple (M_1)	K-Stone	23706	298.87	421.12	0.2024	82.336	20.584
DSE (S_4)	Low depth (M_3)	Brent	35652	298.88	421.11	0.222	90.391	22.598
DSE (S_4)	Low depth (M_3)	Hybrid	36136	298.87	421.12	0.218	88.642	22.160
DSE (S_4)	Low depth (M_3)	K-Stone	36536	298.87	421.12	0.2276	92.588	23.147
DSE (S_4)	Smallest (M_2)	Brent	35668	298.87	421.12	0.222	90.432	22.608
DSE (S_4)	Smallest (M_2)	Hybrid	36152	298.87	421.12	0.218	88.682	22.171
DSE (S_4)	Smallest (M_2)	K-Stone	36552	298.87	421.12	0.2276	92.588	23.147
DSE (S_4)	Simple (M_1)	Brent	35710	298.87	421.12	0.224	91.205	22.801
DSE (S_4)	Simple (M_1)	Hybrid	36194	298.87	421.12	0.220	89.455	22.364
DSE (S_4)	Simple (M_1)	K-Stone	36594	298.87	421.12	0.2296	93.401	23.350
T-Table		Brent	54474	298.87	421.12	0.378	153.730	38.432
T-Table		Hybrid	54958	298.88	421.11	0.374	151.980	37.995
T-Table		K-Stone	55358	298.88	421.11	0.3833	155.926	38.982

Power computed at 10 MHz. "Total Energy" represents the energy required to produce 4000 blocks of keystream

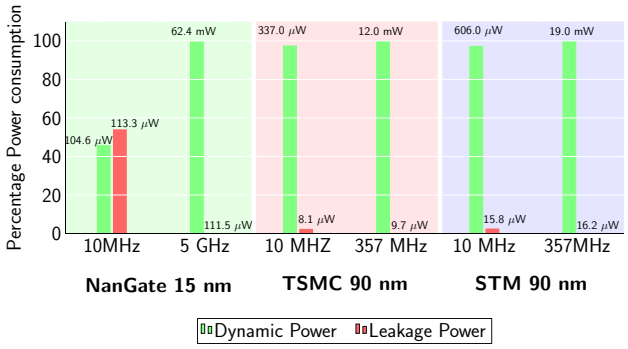


Fig. 10 Percentage contributions of dynamic and leakage powers at different frequencies

hence inversely as the clock period. Since the total physical time to complete any operation, all other things being the same, varies directly as the clock period, the dynamic component of the energy consumption (product of dynamic power and total time) is generally constant with respect to change in clock period or frequency. Therefore, at higher frequencies the dynamic energy of the same circuit remains a constant as the contribution of the leakage energy becomes lesser and lesser and this was what led [1,10] to conclude that at high frequencies the total energy (sum of dynamic and leakage energies) consumption of block ciphers is more or less a constant and frequency-independent. In the later experiments, where we increase the frequency of the clock to the NanGate circuit, we did notice much reduced energy consumption due to the effect of the leakage energy becoming insignificant. The effect is illustrated in Fig. 10, where we illustrate the percentage consumption of the dynamic and leakage power (which is just energy consumed per unit time) contributions to the total contributions, both at 10 MHz and at a frequency at which the clock period is close to the critical path of the circuit. For the 90 nm libraries, the leakage power is of the order of tens of microwatts and less than 3% of the total consumption for all frequencies. For the NanGate 15 nm library, the leakage power is over 100 μW and contributes over 50% of the total consumption at 10 MHz. At a higher frequency, the contribution of the leakage power reduces to 2% although its physical value is more or less the same

5.2 Second dataset

From the experiments, we have conducted in the previous subsection, we identified configurations of the various components that help to construct fast and energy efficient circuits. For the TSMC and STM libraries, we did identify that the DSE architecture is best for energy consumption. Also the particular architecture of the Mixcolumn circuit does not seem to be a very critical factor in the final scheme of things, and so we select the low-depth Mixcolumn M_2 architecture. We also select the hybrid architecture for the adder since the total energy consumptions of both architectures are comparable. Although this is not the most efficient configuration for the NanGate library, for uniformity we decided to use the same architecture for all the libraries, also since the energy consumed in the configuration is within 10% of the optimum architecture for NanGate. We used this configuration to construct a state update function and used the generic unrolling technique to construct multiple-round unrolled circuits. We present the simulation and synthesis results in Table 8.

We have already seen that the energy consumptions of stream ciphers have a parabolic relation wrt unrolling, which is to say that the energy at first decreases with increased unrolling, and reaches an optimum and increases again if unrolled beyond this point. For ciphers like Trivium which have very simple update functions, the optimum occurs at values of r as high as 160. However, SNOW-V, which has a more complex update function we find the optimum occurs at $r = 2$, for the TSMC library but $r = 1$ for the STM library. For the NanGate library, we see a steady decrease in energy consumption wrt unrolling, which is primarily due to the fact that the NanGate circuits produce a comparatively lower volume of glitches compared to the 90 nm libraries, which leads to lower increase of dynamic energy with increasing unrolling. While the energy graph wrt unrolling for this library is still expected to be parabolic, the minimum most likely occurs for values of $r > 4$.

5.3 Third dataset

In the next set of experiments that we performed, we tried to achieve maximum throughput by increasing the clock frequency so that the clock period was equal to or just more than the critical path as computed by the circuit compiler (for the same circuit components as in the second dataset except for the adder: we used the Kogge–Stone adder for this set of experiments). When we synthesize circuits for higher clock frequencies, the compiler sometimes substitutes gates of higher drive strength into the circuit for which the area of the circuit increases sometimes. After synthesis, to verify correctness, we further did a timing analysis with 10000 randomly chosen test vectors on the synthesized netlist and satisfied ourselves that we did not encounter any timing violation, which indicates that all the output signals stabilize much before the clock cycle ends. The results are shown in Table 9, along with comparison with fully unrolled AES-128 and AES-256, which shows that in terms of speed SNOW-V is much superior to both these block ciphers. It is noteworthy that in the NanGate 15 nm library a two-round unrolled SNOW-V reaches throughput over and above 1 Tbps! A look at the energy consumption of the NanGate circuits in Table 9 reveals a that the energy consumption increases

Table 8 Synthesis results of the multiple-round unrolled SNOW-V circuit. Power computed at 10 MHz. "Total Energy" represents the energy required to produce 4000 blocks of keystream

Lib	# Unrolled	Area (GE)	Core latency (ns)	Max TPUT (Gbit/s)	Power (mW)	Total energy (nJ)	Energy/block (pJ/128-bit)
TSMC 90 nm	1	30861	5.05	24.92	0.348	141.688	35.422
TSMC 90 nm	2	55911	8.99	27.97	0.622	126.660	31.665
TSMC 90 nm	3	81541	12.35	30.48	1.086	147.642	36.910
TSMC 90 nm	4	105373	15.26	32.89	1.669	170.238	42.560
STM 90 nm	1	27487	5.63	22.36	0.580	235.863	58.966
STM 90 nm	2	50528	9.55	26.33	1.243	253.116	63.279
STM 90 nm	3	73924	12.83	29.34	2.402	326.713	81.678
STM 90 nm	4	95211	16.53	30.37	4.064	414.569	103.642
NanGate 15 nm	1	35793	0.32833	383.33	0.207	84.370	21.093
NanGate 15 nm	2	63472	0.62518	402.24	0.352	71.647	17.912
NanGate 15 nm	3	91769	0.94992	396.32	0.498	67.782	16.946
NanGate 15 nm	4	118183	0.9312	539.05	0.637	64.964	16.241

Table 9 Simulation results for SNOW-V when clock period is close to the critical path

Library	# Unrolled	Area (GE)	Core latency (ns)	Clk period (ns)	Actual TPUT (Gbit/s)	Power (mW)	Total energy (nJ)	Energy/block (pJ/128-bit)
TSMC 90 nm	1	31191	2.5	2.8	44.91	12.024	137.094	34.273
TSMC 90 nm	2	57011	4.5	5.0	50.20	11.417	116.453	29.113
TSMC 90 nm	3	83479	7.0	7.5	50.05	13.443	137.519	34.380
TSMC 90 nm	4	108026	9.0	10.0	50.00	15.554	159.268	39.817
STM 90 nm	1	27743	2.5	2.8	44.91	18.974	216.335	54.084
STM 90 nm	2	51004	4.5	5.0	50.20	22.188	226.318	56.579
STM 90 nm	3	73969	7.0	7.5	50.05	29.099	297.686	74.421
STM 90 nm	4	96873	9.0	10.0	50.00	39.095	400.331	100.083
NanGate 15 nm	1	34847	0.20	0.20	628.68	62.505	50.904	12.726
NanGate 15 nm	2	62957	0.24	0.24	**1045.75**	106.387	52.087	13.022
NanGate 15 nm	3	97010	0.36	0.36	1042.68	157.110	77.147	19.287
NanGate 15 nm	4	119129	0.48	0.48	1041.67	185.453	91.154	22.788
Comparisons								
TSMC 90 nm	AES-128	168994	16	16	8.00	99.581	1593.288	1593.288
STM 90 nm		145299	12	12	10.67	298.146	3577.750	3577.750
NanGate 15 nm		163337	1.20	1.20	106.67	374.999	449.998	449.998
TSMC 90 nm	AES-256	236833	22.0	24.0	5.33	147.390	3537.367	3537.367
STM 90 nm		204096	16.0	16.0	8.00	456.523	7304.360	7304.750
NanGate 15 nm		230468	1.60	1.60	80.00	595.349	952.559	952.559

with unrolling. This is completely opposite to the trend given in Table 8 where the energy consumptions decreased with unrolling. The difference stems from the fact that the figures in Table 9 are generated for a clock period close to the circuit critical path. When this happens, the compiler is forced to make certain adjustments to the final netlist to meet the slack requirements, so that synthesized netlist for the multiple unrolled circuits does not exactly match the geometric description in Fig. 6b, due to some intra-block optimizations within successive round functions. Hence, we find that the most energy-efficient SNOW-V architecture is for the one-round unrolled architecture constructed with the standard cells of the NanGate 15 nm library (12.726 pJ per block). This figure is much lower than that calculated at 10 MHz in Table 8, because as already explained in Fig. 10, the contribution of the leakage energy at higher frequencies becomes insignificant, whereas dynamic energy stays more or less constant.

6 Conclusion

In this paper, we proposed area, speed and energy-efficient architectures for the SNOW-V stream cipher. We used the byte-serial Atomic-AES architecture to construct serialized low-area circuits that consume as little as 4776 GE with the STM 90 nm library. To implement high-speed/low-energy circuits, we take the help of generic high-level techniques like round unrolling as well as specific low-level techniques like optimization of individual components like the AES S-box, Mixcolumn and the 32-bit integer adder circuit. As a result, with the NanGate 15 nm library we report **a)** an implementation consuming around 12.7 pJ that produces one block of 128-bit keystream and **b)** an implementation that produces a throughput of over 1 Tbps. These are the first hardware results reported for this stream cipher.

Funding Open Access funding provided by EPFL Lausanne

References

1. Banik, S., Bogdanov, A., Regazzoni, F.: Exploring energy efficiency of lightweight block ciphers. In: Selected Areas in Cryptography—SAC 2015—22nd International Conference, Sackville, NB, Canada, August 12–14, 2015, Revised Selected Papers. pp. 178–194 (2015)
2. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES: A compact implementation of the AES encryption/decryption core. In: Progress in Cryptology—INDOCRYPT 2016—17th International Conference on Cryptology in India, Kolkata, India, December 11–14, 2016, Proceedings. pp. 173–190 (2016). https://doi.org/10.1007/978-3-319-49890-4_10
3. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES v 2.0. IACR Cryptology ePrint Archive p. 1005 (2016). http://eprint.iacr.org/2016/1005
4. Banik, S., Mikhalev, V., Armknecht, F., Isobe, T., Meier, W., Bogdanov, A., Watanabe, Y., Regazzoni, F.: Towards low energy stream ciphers. IACR Trans. Symmetr. Cryptol. **2**, 1–19 (2018)
5. Brent, R.P., Kung, H.T.: A regular layout for parallel adders. IEEE Trans. Comput. **3**, 260–264 (1982). https://doi.org/10.1109/TC.1982.1675982
6. Cannière, C.D., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986, pp. 244–266. Springer (2008)
7. Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: A new SNOW stream cipher called SNOW-V. IACR Trans. Symmetr. Cryptol. **3**, 1–42 (2019). https://doi.org/10.13154/tosc.v2019.i3.1-42
8. ETSI/SAGE: Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2, document 2: SNOW 3G specification, version 1.1, 2006. (2006)
9. Jean, J., Moradi, A., Peyrin, T., Sasdrich, P.: Bit-sliding: a generic technique for bit-serial implementations of SPN-based primitives—applications to AES, PRESENT and SKINNY. In: Cryptographic Hardware and Embedded Systems—CHES 2017—19th International Conference, Taipei, Taiwan, September 25–28, 2017, Proceedings. pp. 687–707 (2017). https://doi.org/10.1007/978-3-319-66787-4_33
10. Kerckhof, S., Durvaux, F., Hocquet, C., Bol, D., Standaert, F.: Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint. In: Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings. pp. 390–407 (2012). https://doi.org/10.1007/978-3-642-33027-8_23
11. Kogge, P.M., Stone, H.S.: A parallel algorithm for the efficient solution of a general class of recurrence equations. IEEE Trans. Computers **22**(8), 786–793 (1973). https://doi.org/10.1109/TC.1973.5009159, https://doi.org/10.1109/TC.1973.5009159
12. Li, S., Sun, S., Li, C., Wei, Z., Hu, L.: Constructing low-latency involutory MDS matrices with lightweight circuits. IACR Trans. Symmetr. Cryptol. **1**, 84–117 (2019). https://doi.org/10.13154/tosc.v2019.i1.84-117
13. Maximov, A.: AES mixcolumn with 92 XOR gates. IACR Cryptology ePrint Archive p. 833 (2019). https://eprint.iacr.org/2019/833
14. Maximov, A., Ekdahl, P.: New circuit minimization techniques for smaller and faster AES sboxes. IACR Trans. Cryptogr. Hardw. Embed. Syst. **4**, 91–125 (2019). https://doi.org/10.13154/tches.v2019.i4.91-125