ASIC Hardware Performance

Tim Good and Mohammed Benaissa

Department of Electronic and Electrical Engineering,
University of Sheffield, Mappin Street, Sheffield, S1 3JD, UK
{t.good,m.benaissa}@sheffield.ac.uk

Abstract. This chapter presents detailed hardware implementation results and performance metrics for the eSTREAM candidate stream ciphers remaining in the Phase 3 hardware profile. Performance assessment has been made in accordance with the eSTREAM hardware testing framework in terms of power, area and speed. An attempt has been made to quantify the flexibility and scalability dimensions of performance. The results are presented in tabular and graphical format together with summarising the utility of the candidates against two notional applications: one for 10Mbps wireless network and a second for 100kHz RFID. Where applicable to a particular cipher, guidance on any limitations on the choice of key or IV is given. The chapter concludes with a summary of the performance of each of the candidates and some general guidance for future low resource hardware stream cipher development.

1 Introduction

In 2004, a project under the Information Societies Technology (IST) Programme of the European Commission ECRYPT Network of Excellence called eSTREAM was started [1] tasked with seeking a strong stream cipher. Thirty-four candidate ciphers were submitted to either a software or hardware profile. From initial evaluations at SASC 2006 and SASC 2007, the commencement of Phase 3 saw the eSTREAM candidates reduced to eight in the software profile and eight in the hardware profile. There is no single cipher listed in both profiles. The aim of this chapter is to document the hardware performance aspects of those ciphers short-listed in the hardware profile.

A stream cipher, formally, is a symmetric cipher which generates a sequence of cryptographically secure bits called the key stream which is then combined with either the plaintext or ciphertext, at the bit level, using the exclusive-or operation. The basic topology (Fig. 1) of a stream cipher consists of a register to store the key and an initialisation vector (IV) together with a function for its update (typically some sort of feedback shift register). This register forms the current state of the cipher and is clocked for successive bits of the keystream. The next component is a non-linear reduction function which takes part or all of this state and combines the bits in a non-linear fashion normally to yield a single bit of the keystream. This bit is then exclusive-or'ed with the plain / cipher text. In a second form, the plain or cipher text may be incorporated into the state update feedback function to effectively create a cipher-feedback mode.

M. Robshaw and O. Billet (Eds.): New Stream Cipher Designs, LNCS 4986, pp. 267–293, 2008.

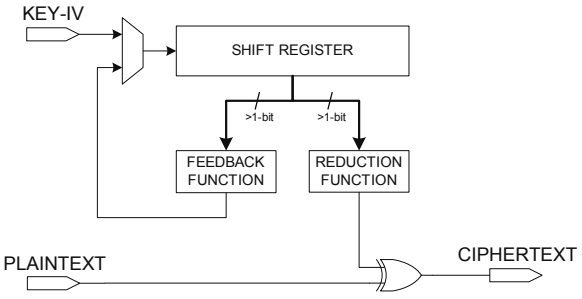

Fig. 1. Generic stream cipher

A vital function, in terms of security, is the period of the initial key and IV mixing to prevent key recovery attacks. In this period, a cryptographically strong feedback function is needed to operate upon the state for a number of iterations (basically hashing). The reduction function used to output the keystream can be somewhat weaker.

The same XOR per-bit property which does not change (permute) the ciphertext/plain-text order and provides stream ciphers with their simplicity also leads to an attack on all basic stream ciphers. This is the bit-flip attack where an active attacker may change the state of bit of plaintext at any positions within the ciphertext. The effectiveness of this attack depends on application and may be prevented using a message authentication code.

The original eSTREAM call for cipher primitives [1] made provision for two profiles, one for software, requiring equivalent security of 2^{128}, and one for hardware, requiring 80-bit (2^{80}) security. An extension to the basic cipher was also defined for those wishing to supply a message authentication code (MAC). The call recognised the importance of resource utilisation for both profiles in that the deployed environments for stream ciphers often have very restricted resources (eg smart cards). Subsequently, many of the hardware candidates have provided 128-bit key versions which have also been considered in this treatment.

Security analysis remains the overriding concern compared to hardware or software performance analyses; however, performance is both a technical requirement and economic imperative for any design including a cipher primitive. Put at its simplest, in the context of hardware design, it is the cost for each primitive in attaining the security. The aim of this chapter is to provide an independent set of hardware results for the promising candidates to further the understanding of their relative merits and to focus cryptanalysis efforts on the low resource candidates.

Hardware performance is multi-dimensional and the importance of the various quantities such as area, throughput and power depends on the specific application. The eSTREAM hardware testing framework [2] defines five dimensions: compactness, throughput, power/energy consumption, flexibility and simplicity. It was also stated that the Advanced Encryption Standard (AES) is to be used as the benchmark for comparison and candidates should be *smaller* and *faster* than the AES.

Economics plays a crucial role in contemporary consumer electronics, this leads designers to be concerned about design efficiency. Low-resource design is an increasingly important area due to customer appetite for feature-rich hand-held battery operated ICT devices. Typically, a system will have specified requirements in terms of timeliness or throughput which the designer must meet. The cost of achieving the required specification is measured broadly in terms of design costs, device cost (proportional to area occupied by the design) and energy consumption (battery life).

For any digital design the fundamental performance metrics are power, area and time. From these many other metrics are derived (eg throughput versus area ratio) the applicability of which depends on application. For this treatment two representative applications are selected, a wireless LAN operating at 10Mbps and a Radio Frequency Identification Device (RFID) operating at 100kHz clock.

2 Measuring Hardware Performance

For any digital design there is a small set of metrics which can be obtained from the design flow together with some simulations. It is this primary set of metrics which is used to calculate the other derived metrics which designers use as a convenient method for comparing different designs. The definitions used in this paper are given below:

Process: The fabrication technology used. The name normally indicates the smallest feature size, library usage and gate construction (e.g. $0.13\mu m$ standard cell CMOS).

Interface: Designs are invariably part of a larger system and thus require connections (on or off chip) with other designs. All the designs in this paper use a synchronous interface with handshaking and on-chip communication is assumed. In this paper, the interfaces differ by their bus widths. Thus the bus width in bits for I/O is included in the results.

Area: Amount of silicon used for the core design (excluding power rings and I/O cells). This result is typically expressed in μm^2 for a specified process. However, the more usable process independent method of expressing the area is to calculate the Gate Equivalence (GE) of the total area by dividing by the lowest power two-input NAND gate's area.

Load/Initialisation Cycles: The definition used here was from RESET going inactive, through loading key and IV, until the validity of the first output bit is signalled. Many would quote just the key/IV mixing cycles however this would fail to account for the impact on interfacing decisions on the latency.

Bits per cycle (running): For the simplest stream ciphers this is the number of bits of output keystream per clock cycle. However, many operate in a way that produces batches of output (e.g. a block cipher in output feedback mode) thus the definition has to include a second clause on sustainable output rate. Thus the better definition is number of bits of output for all subsequent batches/blocks of keystream divided by the number of cycles per batch/block.

Design frequency: This is the clock rate selected by the designer and applied as a constraint to the design tools. The tools will make decisions on driver strengths to meet this requirement. Thus the higher the constraint the more area will be consumed. For low resource design a modest rate must be selected.

Max. Clock frequency: Designs have many connections between inputs outputs and registers, each of these form a timing path (or arc). Simplistically, the slowest arc in the design is the critical path and sets an upper bound on the clock frequency. The design may be clocked at a significantly lower rate.

Power consumption: Ideally a chip would be manufactured and measurements made for a large set of operations, however, this would be both time-consuming and costly. The alternative is to use specialist tools which operate using estimations of parasitic parameters (resistance and capacitance) from the physical layout of a design together with switching activity from a set of random test vectors. For CMOS there are two components to the power: the static power (roughly proportional to area) and a second dynamic component proportional to the switching activity (probability of a switching event occurring and frequency of operation). Both components also depend on supply voltage. The typical core voltage for the process should be used. At low frequency the static power is significant whilst at the other extreme may be neglected. Power results can be scaled with an acceptable margin of error to other frequencies if the static and dynamic components are treated separately.

The primary metrics may be used to wholly describe a designs performance, however, as can be seen there are many dimensions to performance so engineers often use derived metrics to provide a single dimension for comparison. There is no universal agreement on which metric is the best. The true requirement is to meet all the application driven design constraints. The commonly used derived metrics are given below:

Throughput: The rate at which new output is produced with respect to time, typically expressed in bits-per-second. This definition is further clarified to be the sustainable rate once initialisation is completed at a given operating clock frequency. It is thus simply bits-per-cycle multiplied by the clock-frequency. The maximum throughput will occur at the maximum clock frequency, however, remember that the design tools were given a slack timing constraint to favour area so this metric must be used with care when considering low resource design performance.

Area-Time product: The product of the time taken to produce each new output bit and the area of the design. The reciprocal metric is presented as the **throughput-to-area ratio** (TPAR). Either representation is frequently used as a measure of design efficiency. However, once again, note that the metrics are at their best at the maximum clock frequency.

Energy-per-bit: This is calculated by dividing the total power consumption by the throughput. Care must be taken to ensure that the power and throughput figures used are for the same clock frequency. At first this measure may appear to be frequency independent, however, if modelled at a low frequency (eg 100kHz)

the static power will have a significant impact thus larger area designs will be *less efficient*. Conversely, at higher frequencies designs with large amounts of switching activity (including that from switching hazards to do path differences in the large fields of XOR gates present in most crypto-primitives) dominates the power.

Power-area-time product: This is the triple product formed from area-time product and the power consumption. As with energy per bit, this is maximised at the highest operating frequency due to the diminishing effect of the static power.

Power-Time product: Specifically, the product of power and latency (total time taken including initialisation and loading key and IV). This metric is particularly useful for measuring utility of a candidate in application such as RFID where both the power consumption and timeliness of response are important.

As has been frequently stated hardware performance analysis is multidimensional and application specific. Thus to resolve the impasse on which figures to quote the decision is made here to quote the following:

1. The primary design results for designs prepared with a slack timing constraint of 10MHz clock.
2. 'Best' metrics: Performance metrics for the designs operating at their maximum frequency given the 10MHz constraint.
3. High-end wireless: Performance metrics for an output rate of 10Mbps, taken as a typical estimate for future wireless LAN (proposed standards range between 1-100Mbps).
4. Low-end wireless: Performance metrics for a clock rate of 100kHz, as the low end of RFID/WSN tags which may be powered / clocked directly from the interrogating RF field.

The first three performance dimensions: compactness, throughput and power consumption may be readily compared quantitatively however the remaining two of flexibility and simplicity are much more subjective. There is little quantitative guidance in the testing framework so some definitions are offered here; admittedly the choice of metric is arbitrary but any *scale* is better than none.

Flexibility: It is assumed that a measure of the design space performance trade-offs is required. Herein defined as the (dimensionless) ratio of the throughput-to-area ratio for the maximum performance design variant ($TPAR_{max}$) divided by the corresponding ratio for a low-resource design operating at 100kHz ($TPAR_{100kHz}$)

Simplicity: It is assumed that the desired metric here is a measure of the design time (unfortunately the design work had to be fitted around existing work load thus this could not be reliably accounted for). There are a number of software-engineering metrics which are generally used to describe the complexity / simplicity of a source file. Metrics vary in sophistication and applicability to hardware design; one of the simplest, used here is the number of lines excluding blank lines and comments for all the design source (VHDL) files.

3 Candidate Ciphers

This section describes implementation specifics for the hardware design of the candidate ciphers in alphabetical order. All the designs are complete in that they contain a suitable finite state machine as controller and support usable handshaking. Further, for candidate ciphers requiring any padding of key/IV or specific initialisation constants these are performed within the architecture thus included in the stated results. However, for brevity only the critical aspects of the datapath are described.

3.1 Decim^{v2} and Decim-128

Decim has two variants for 80-bit and 128-bit key lengths. The datapath comprises four principle modules, a linear feedback shift register (LFSR), non-linear filter (NLF), an irregular decimation mechanism *ABSG* and a first-in first out (FIFO) buffer. The LFSR stores the internal state of the cipher, a total of 192 bits. The NLF combines 14 taps from the LFSR using the reduction XOR and conventional addition to yield a single bit output. This output is fed back to the LFSR during the mixing phase to produce non-linear feedback and in the running phase feeds the ABSG. The ABSG produces bits in an irregular pattern of clock cycles, thus the output and its control is feed to a FIFO which is read monotonically to restore a regular clocking pattern for the output keystream. The design of Decim permits up to ×4 parallelism by replicating the filter function at the expense of an ABSG of exponentially increasing complexity, the simplest solution being to use a lookup table for the ×4-ABSG. A compact circuit for the ×1 ABSG is shown in Fig. 2.

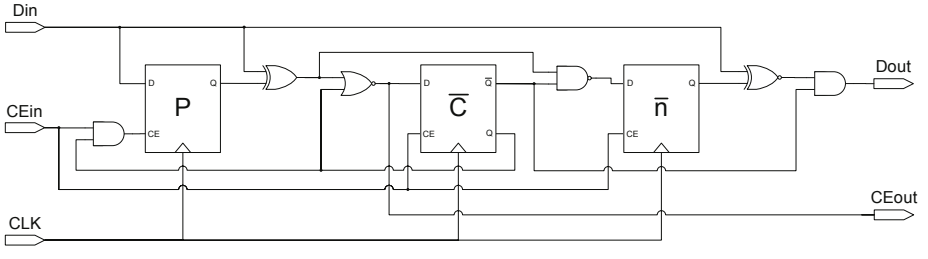

Fig. 2. Compact implementation of the Decim ABSG function

The initial internal state derived using arithmetic operations from the key (80-bit) and IV (64-bit), initially appears complex, however may be implemented using only a few conditionally applied XOR gates. The padding sequence required for Decim-128 may be conveniently generated using a toggle-flip-flop. The required circuit for the LFSR for both the 80-bit and 128-bit versions of Decim are shown in Fig. 3 and Fig. 4.

The initial phase of mixing (4×192 cycles) is followed by a second phase, of variable period, performed in multiples of 4 cycles, until the FIFO is full.

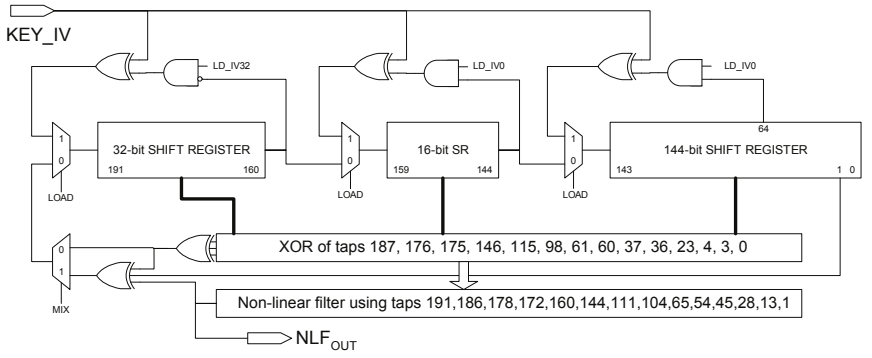

Fig. 3. Decim-80 LFSR supporting loading key, IV, padding, mixing and running phases

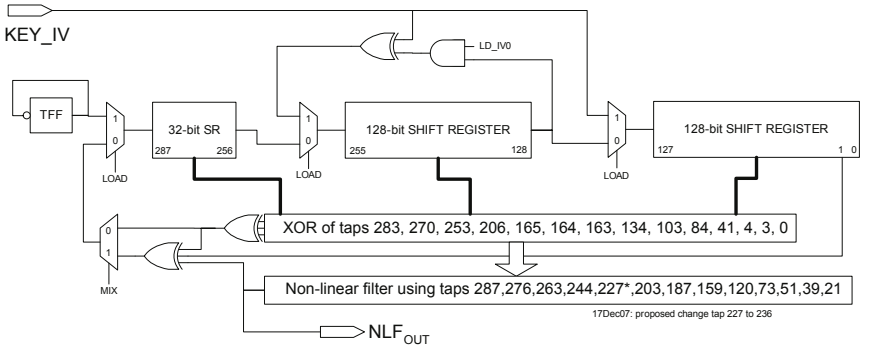

Fig. 4. Decim-128 LFSR supporting loading key, IV, padding, mixing and running phases

The FIFO is 32-bits for Decim-80 and 64-bits for Decim-128. Its implementation must support simultaneous or individual read/write operations inclusive of both buffer empty and buffer full conditions. The Decim specification requires a buffer refill mechanism, this halts the keystream output and refills the FIFO, however such an event is highly improbable (would be a distinguisher for the cipher) and would require the NLF to output a sequence buffer-length long of all zeros or all ones. From a hardware perspective, this presents a practical verification problem (finding a suitable test vector to test the buffer refill condition is improbable). In normal operation, the FIFO drops excess bits from the ABSG once full, incorrect implementation will appear as a single cycle misalignment of portions of the keystream from known test vectors.

3.2 Edon80

Edon80 by design was intended as an 80-element pipeline. However, the simple software definition for the initial mixing and running phases belies relatively high hardware complexity for its implementation. The nature and direction of

shifting between loading key, IV and padding, mixing phase and running phase changes resulting in a significant number of additional multiplexers and a need to duplicate the key register. The implementation (Fig. 6) requires an additional 80 cycles at the end of the initialisation phase to avoid requiring additional pipelining of control lines (saves 160 FF). Edon80 (pipelined) is the largest design in the hardware profile so a more iterative and lower area version was also designed (Edon80×4). This comprises only four *e-transformers* rather than the more usual 80, however, has relatively poor performance, thus only the pipelined version is described in further detail.

The datapath is area is dominated by the 80 pipeline processing elements each of which comprises several multiplexers and an *e-transformer* and two bits of internal state storage. For clarity, it should be noted that the majority of the values used in Edon80 comprise two bits. Due to correlative nature of the initial mixing phase a second temporary copy of the internal state (160 bits), the temporary register, is required. To meet the required different bit ordering for load, mixing and running both the pipeline and temporary registers require multiplexers to support shifting in either direction. The temporary register is also used to provide key storage using suitable feedback, however, the key bits are required to be cycled in both forward and reverse orders during mixing so a second key register is also required. The key is 40×2-bit elements and used consecutively to drive the 80 processing elements; this results in a conceptual

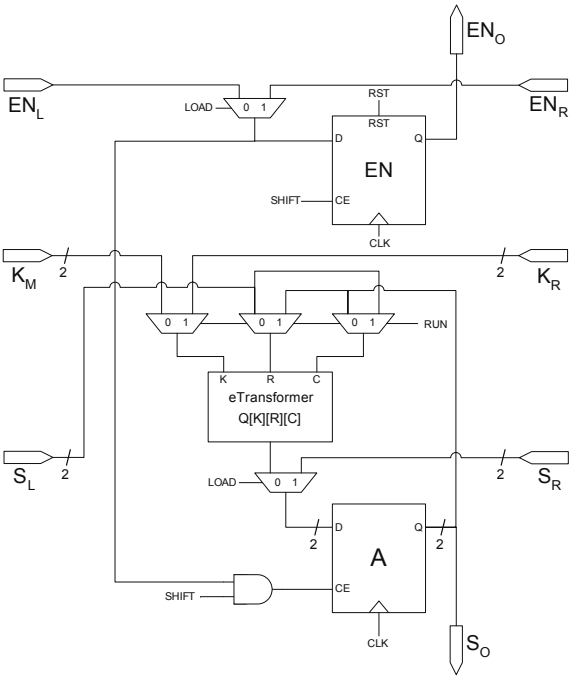

Fig. 5. An Edon-80 processing element (pipeline stage)

floorplan for the pipeline elements to be laid out around two revolutions of the perimeter of a circle.

The datapath used accepts the loading of a key (80-bits) followed by an IV (64-bits) and performs the necessary padding and bit-order reversal needed for the mixing phase. A suitable design for each pipeline element is shown in Fig. 5. A controller is used to generate the various control lines. For clarity, it should be noted that when the load and run multiplexer select lines are both low then mixing should be assumed.

The Edon80 reference code is written from a software efficiency perspective and the stored state for each clock cycle, during mixing, for a pipelined architecture is not readily available. The following fragment of C provides a suitable reference model for debugging such a hardware design:

```
#define mod80(x) (((x)+800)%80)
for (j=0;j<160; j++)
{
   Enable[0] = (j<80) ? TRUE : FALSE;
   NextState[0] = (Enable[0]==TRUE)
       ? ctx->Q[j%80][Temp[mod80(80-1-j)]][State[0]] : State[0];
   for (k=1; k<80; k++)
      NextState[k] = (Enable[k]==TRUE)
         ? ctx->Q[mod80(j-k)][State[k-1]][State[k]] : State[k];
   PrintInternalState(j,State);
   for (k=i-1; k>=0; k--) State[k]  = NextState[k];
   for (k=i-1; k> 0; k--) Enable[k] = Enable[k-1];
}
```

3.3 F-FCSR-16 and F-FCSR-H

There are two variants of the core F-FCSR design: F-FCSR-H supports an 80-bit key and F-FCSR-16 a 128-bit key. Both are conveniently implemented using shift registers for state storage. In comparison to many of the other designs, F-FCSR has a non-linear shift register and a linear reduction filter. The state register is updated according to a fixed polynomial, poly, which defines the inclusion of non-linear carry units which act on the feedback term to modify the state update. The key followed by the IV may be directly loaded into the state register with the feedback suppressed. As part of the mixing process, the original initial state is needed thus has to be stored in a second temporary shift register. This effectively doubles the size of the internal state. A side effect is that in order to minimise the cycles taken for mixing, the cipher needs to operate at the word level (8-bits for F-FCSR-H and 16-bits for F-FCSR-16) during the second phase of mixing (often convenient for key/IV loading too). However, due to the non-linear nature of the feedback operates using single-bit shifts during both the first phase of mixing and the running phase. This is performed in hardware using multiplexers between the state register elements. A suitable datapath is shown in Fig. 7, the design for F-FCSR-16 is similar except operates on 16-bit words and has a 256-bit internal state. An additional shift register may be used to convert the key/IV input and keystream output to/from serial format.

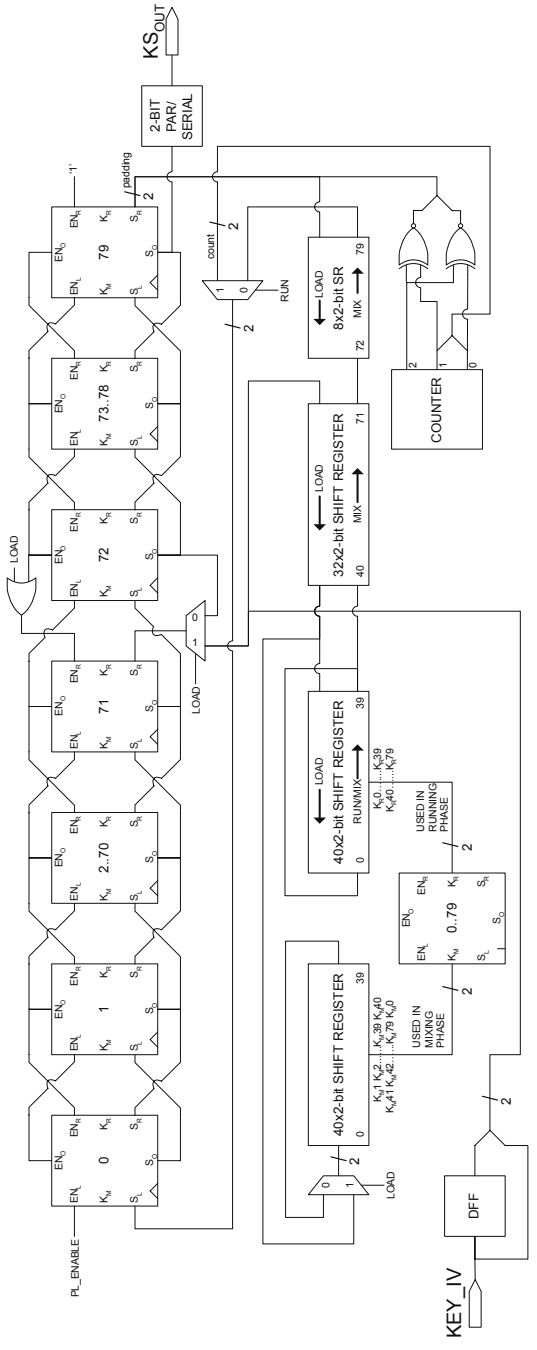

Fig. 6. A possible datapath architecture for pipelined Edon-80

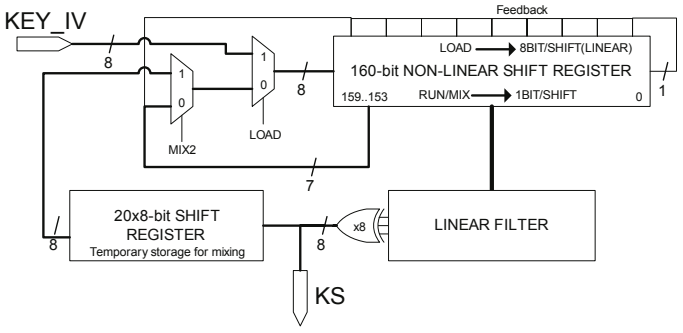

Fig. 7. Datapath for core of F-FCSR-H

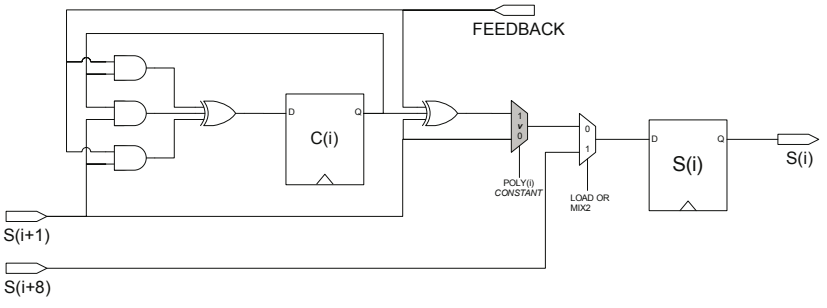

Fig. 8. Schematic of F-FCSR-H non-linear shift register element including *carry bit*

Fig. 8 shows a construction for the elements of the non-linear shift register, a virtual multiplexer (shaded and marked v) is shown to depict the presence of the cell according to the specified (constant) polynomial.

The cautious implementer should detect the all zero internal state (will be entered when both key and IV are all zero). If this state is entered it will persist (keystream all zeros) and result in the unencrypted output of plaintext! One low-resource solution is to detect the all zero IV and set the final bit to one. Without such a test the cipher's implementation would fail to demonstrate plaintext-ciphertext segregation to an assessor. This can be done with a single flip-flop and few combinatorial gates attached to the KEY/IV input.

3.4 Grain and Grain-128

Grain comes in two versions Grain-v1 for 80-bit key and Grain-128 for 128-bit key. For clarity these are referred to in this chapter as Grain-80 and Grain-128. Both support extensive parallelism at a cost of only replicating the filter functions, $\times 16$ is possible for Grain-80 and $\times 32$ for Grain-128. This feature, in common with Trivium, affords Grain a large and advantageous design space for trading between speed, area and power.

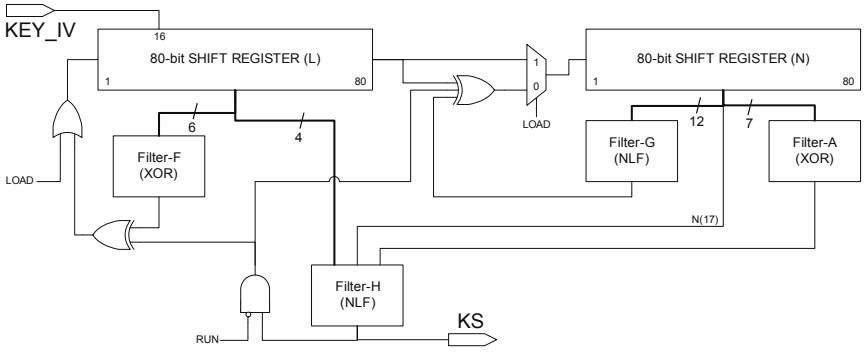

Fig. 9. Block diagram of Grain-80 datapath

During initialisation, the shift registers for Grain-80 are loaded with key (80-bits) and IV (64-bits) then padded with 16 ones. This padding may be done at the same time the key & IV are loaded. The mixing phase is then carried out for 160-bits of shifting before the cipher enters its running mode. The typical architecture for Grain-80 is shown in Fig. 9.

3.5 Mickey

Mickey is based on a pair of shift registers and has versions for 80-bit and 128-bit key and IV. One feature of the cipher is that the shift registers conditionally *jump* this is typically implemented by including the previous state bit in the feedback XOR sum and maintaining monotonic clocking. The two shift registers are each 100-bits for Mickey-80 and 160-bits for Mickey-128. Prior to key loading the state is initialised to zero, this is required as the feedback remains operative during key and IV loading. A number of constant polynomials are used in the description of Mickey to define the required feedback taps. In a hardware description language, conventional logic operations are used to define the required functionality however these are essentially virtual-gates as being constant will be removed during optimisation. Fig. 10 depicts a suitable datapath for Mickey-80 with the detailed composition of the shift registers in Fig. 11 in which virtual gates are shaded and marked by the letter-v and those which conditionally form inverters are marked *inv*. Mickey-128 is very similar except for the longer registers and different tap combinations used to form the *control-bits*. In the circuit shown it is necessary to load the IV first followed by the key.

The cautious hardware designer may additionally wish to detect the all-zero states in either the R or S registers. If either occurs it would be prudent to disable the ciphertext output and signal an error state (this would result in a distinguisher, but *not* of low complexity).

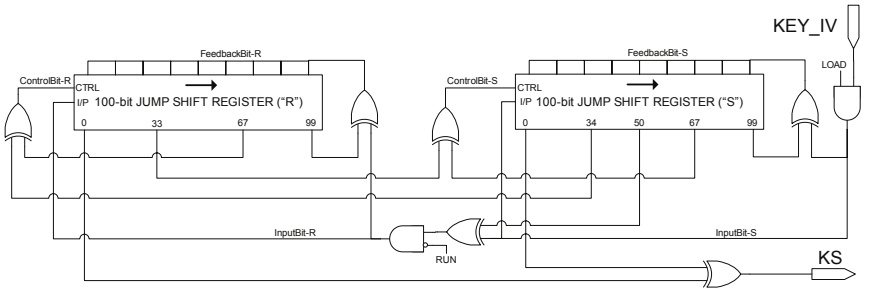

Fig. 10. Datapath for Mickey-80

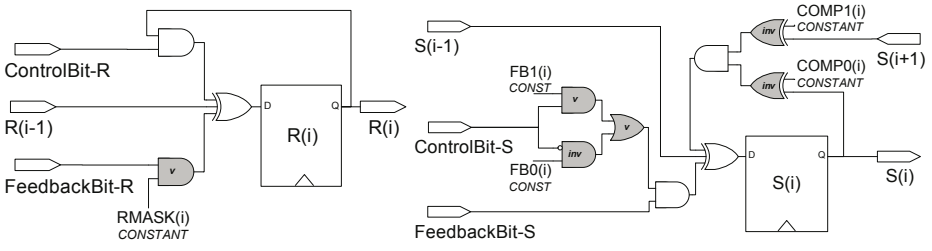

Fig. 11. Schematic detail for performing jump control functions within Mickey's shift registers

3.6 Moustique

Moustique is a self-synchronising stream cipher, which for some communications systems, is an advantageous property. The datapath compromises a 96-bit key register, a 128-bit non-linear shift register (CCSR) and a seven stage pipelined non-linear reduction filter. Three simple functions, g0, g1, g2, are used throughout the CCSR and filter, g1 & g2 providing the non-linearity.

It has a small design space in that a number of the *stages* may be performed iteratively to save area at the expense of latency. The key is contained in a static register thus could support the use of a one-time-programmable memory for key storage directly.

Once the 96-bit key has been entered, sequentially in this design, the 128-bit IV may loaded into the CCSR or in its self-synchronising mode, prefixed by a '0' and applied to the data-input along with ciphertext for decipherment. None of the flip-flops in the key or CCSR require resetting. The only exception is the flip-flop feeding the CCSR for encryption if the IV is not prefixed with a zero.

A suitable datapath is given in Fig. 12 which supports both enciphering and self-synchronising deciphering operations. Although the design in terms of gates would appear relatively simple, the permutative wiring within the stages results in hardware architectures which are dominated by the routing.

In a typical application, a self-synchronising stream cipher operates on the entire data stream, including synchonisation, header and framing bits. This is the principal advantage which can be gained by having the self-synchronising

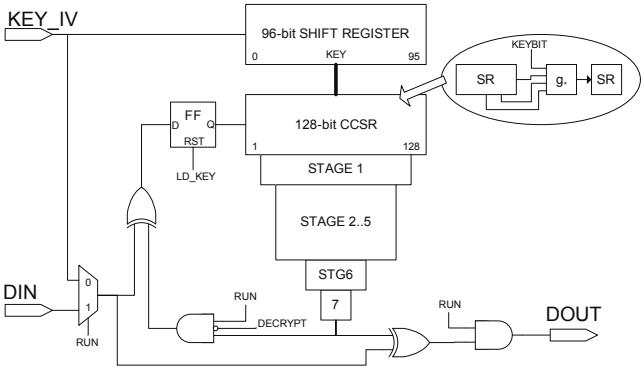

Fig. 12. Combined Moustique encipher/decipher datapath

property. There are a couple of circumstances that the cautious implementer should prevent from occurring.

The use of the all-zero IV for encryption followed by a sequence of plaintext zeros will result in all zero ciphertext irrespective of key. This may allow a potential attacker to gain some limited information and distinguish Moustique from other ciphers. The all-zero IV also results in the first byte of ciphertext being unencrypted. The all-one IV has similar undesirable properties. It is recommended that the cautious implementer prevent the use of IVs containing repeating sequences of low period.

The response of the decipher operation to an all-zero ciphertext, of N bits in length, is to output a portion of constant plaintext N-104 bits long.

3.7 Pomaranch

Pomaranch has 80-bit and 128-bit versions and consist of 6 or 9 sections each being an 18-bit jump-controlled linear feedback shift register (CJCSG), two different types defined, connected by non-linear function (Fig. 13). A jump-control term is derived using a complex key-dependant non-linear function and passed between the stages. This function is formed by the modulo-2 addition of 9 key bits followed by inversion in GF(2^9), further mod-2 addition of another 7 key bits to the middle 7-bits of the inversion result and finally a 7-bit reduction filter (PNLBF) to yield a single jump-control output bit. The final stage does not require the JCout term this is depicted (Fig. 14) by a shaded (virtual) multiplexer marked with the letter-v.

The ciphers authors provide the necessary field constructions for a composite field equivalent in GF($(2^3)^3$). Such construction is typically much more efficient than a conventional ROM look-up table or typical optimisers *random* logic implementation. However, only the middle 7-bits of the 9-bit inversions output are required thus the ROM is approx. $\times 4$ smaller than that needed to describe the GF(2^9) inversion so the advantage of using composite field arithmetic in this

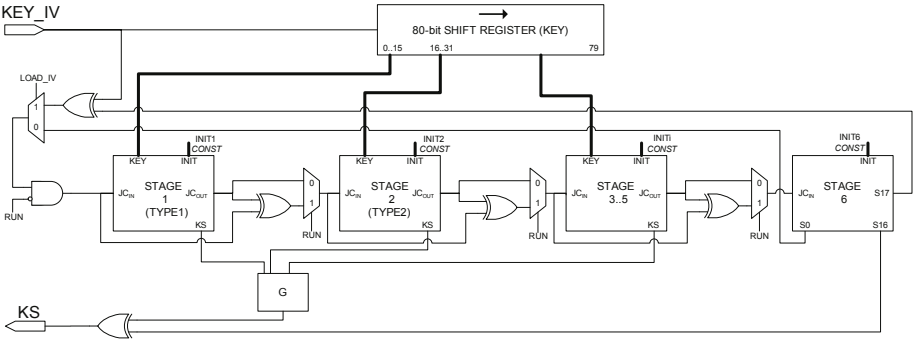

Fig. 13. Pomaranch-80 datapath architecture

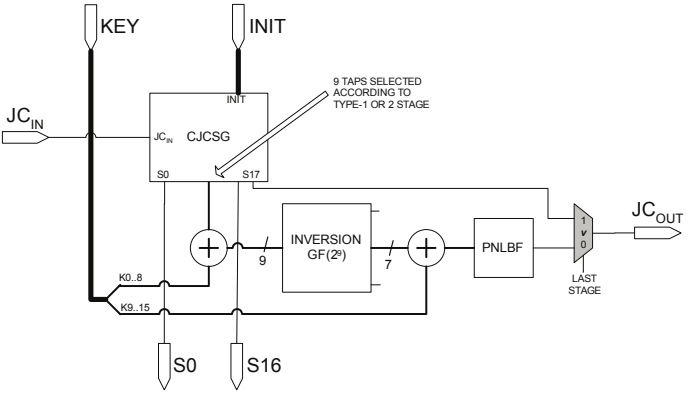

Fig. 14. Generic stage processing element for Pomaranch

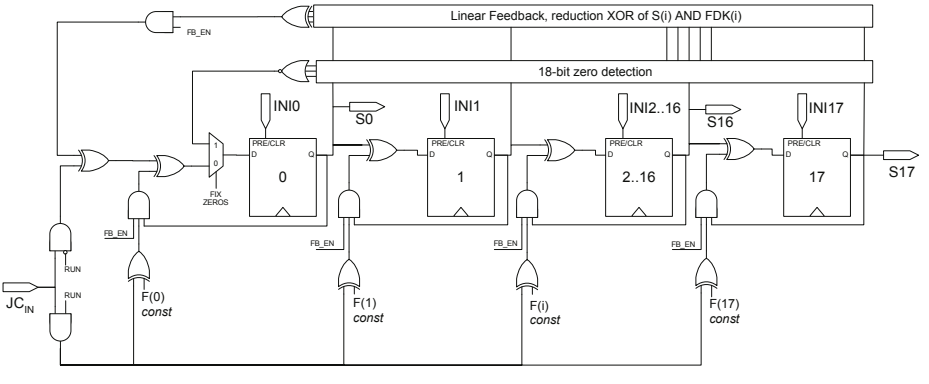

Fig. 15. Pomaranch CJCSG detail

instance is more marginal. The jump-control feedback between the sections frustrates attempts to roll the design into a single configurable stage supported by a suitable memory.

The state contained in CJCSG's must be initialised to a set of constants and the feedback terms being defined by constant polynomials for each type of CJCSG (Fig. 15). During IV loading the feedback must be disabled. To avoid the all-zero state in any CJCSG a step is required during initialisation to conditionally set the LSB. A second output from the CJCSG is one bit of the internal state from each which are combined by a function, G, to yield a single bit of keystream per cycle.

3.8 Trivium

The key feature of Trivium is its simplicity, it supports parallelism by replicating the filter functions from $\times 1$ to $\times 64$ which gives a large design space for trading between throughput, area and power. The internal state is 288-bits giving Grain the edge in terms of lowest area, however, its superior parallelism and simplicity give Trivium the edge in terms of throughput.

The datapath (Fig. 16) may be readily arranged with a few additional gates to perform the required padding during the key/IV loading phase without resorting to additional cycles. Many implementers will favour the $\times 8$ design giving 8-bit I/O, however even if serial output is desired working at say $\times 4$ or $\times 8$ and adding an SR for serialising the I/O to give superior power-area-time performance and reduce the initial mixing time. As an alternative the parallel input/output may be used directly. For $\times 32$ and $\times 64$ variants, the padding between the key and IV is most naturally input as part of the wider key/IV words.

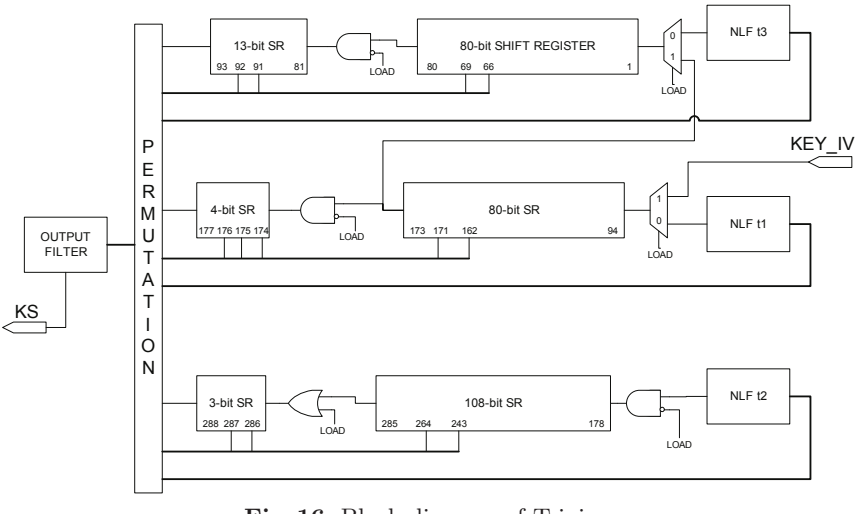

Fig. 16. Block diagram of Trivium

4 Results

This section summarises the ASIC performance results for the authors' hardware designs for the various stream cipher candidates submitted to the State-of-the-art stream ciphers conferences, SASC 2006 [3], SASC 2007 [4] and SASC 2008 [5]. The results include the complete set of Phase 3 candidates submitted to the hardware profile including any clarfications.

All the designs have been implemented using the *same design flow*. The natural bus-width for interfacing to each design was selected rather than forcing all designs to use the same bus-width in order to avoid skewing the results. Cadence tools were used together with ModelSim. The process selected was the same $0.13\mu m$ CMOS and standard cell library. Best-case worst-case timing analysis was carried out for a desired clock rate of 10MHz. The designs were taken through to physical layout (including clock tree synthesis, placement and routing). The final core area was converted to gate-equivalents. The resulting parasitic values were extracted and the netlist back annotated and simulated with known test vectors to validate the design. To estimate the power consumption, random test vectors were applied to the back annotated netlist and simulated to collect switching activity for a set of 100 different 1 kilobit keystream generations. The power modelling was done using the foundry typical values for the process ($1.2V_{core}$ $25°C$), the total power and static component are quoted in the results to permit scaling. The results (Tables 2, 3, 4 & 5) incorporate both initialisation and operational phases of the design under test.

For the notional future wireless network application, battery life, meeting throughput requirements and area are important to the designer. A good

Table 1. Flexibility and simplicity

Design	Flexibility	Simplicity[a]			
	$TPAR_{max} \div TPAR_{100k}$	VHDL (bytes)	Comment lines	Empty lines	VHDL code lines
Grain80 ($\times 1 - 16$)	39,472	5,415	31	10	158
Trivium ($\times 1 - 64$)	116,913	5.916	45	26	159
F-FCSR-H	3,922	4,923	22	33	152
Grain128 ($\times 1 - 32$)	58,224	4,703	21	29	138
Mickey128	4,132	6,399	41	34	127
F-FCSR-16	3,175	5,668	20	38	177
Mickey2(80)	4,545	5,645	20	37	149
Pomaranch80	1,245	23,378	71	156	578
Pomaranch128	1,049	23,378	71	156	578
Moustique	4,762	16,960	44	77	496
Decim80[b]	4,274	16,210	79	103	421
Decim128[b]	3,096	16,560	95	117	396
Edon80 ($\times 4 - 80_{PL}$)	19,632	20,704	95	149	618

[a] Figures quoted for designer's first validated draft.

[b] Decim with $\times 4$ versions are possible but not implemented by these authors. However, the estimated 'best-case' flexibility result will be less than 4 times the stated value.

measure for comparing designs is to consider the trade off between the Energy per bit and Throughput/Area metrics.

RFID applications place limits on power, area and latency directly, excesses in any would make a candidate unsuitable for the application. RFID tags must be fundamentally low cost thus low area. A good metric for performance would be power-latency product versus area.

These results are presented graphically in Figures 17, 18, 19 & 20.

Table 2. Our design results for $0.13\mu m$ Standard Cell CMOS

Design	Key bits	Interface bits	Load/Ini cycles	Bits/Cycle (running)	Max. clock freq. MHz	Area NAND GE, $gates$	Leakage power, μW	Total Power @10MHz, μW
Grain80	80	1	321	1	724.6	1294	2.224	109.4
Grain80×4	80	4	81	4	694.4	1678	3.243	126.6
Grain80×8	80	8	41	8	632.9	2191	4.634	150.7
Grain80×16	80	16	21	16	617.3	3239	7.399	200.5
Trivium	80	1	1314	1	327.9	2580	3.823	175.1
Trivium×2	80	2	660	2	574.7	2627	3.954	182.8
Trivium×4	80	4	332	4	473.9	2705	4.149	184.6
Trivium×8	80	8	168	8	471.7	2952	5.071	203.4
Trivium×16	80	16	86	16	467.3	3166	5.339	214.4
Trivium×32	80	32	45	32	350.9	3787	7.501	282.5
Trivium×64	80	64	24	64	348.4	4921	10.677	374.2
F-FCSR-H	80	8	225	8	392.2	4760	7.973	269.3
F-FCSR-16	128	16	308	16	317.5	8072	13.731	470.1
Grain128	128	1	513	1	925.9	1857	2.698	167.7
Grain128×4	128	4	129	4	584.8	2129	3.806	183.4
Grain128×8	128	8	65	8	581.3	2489	4.898	205.1
Grain128×16	128	16	33	16	540.5	3189	6.882	254.6
Grain128×32	128	32	17	32	452.5	4617	11.442	344.7
Mickey128	128	1	417	1	413.2	5039	8.144	310.7
Mickey2(80)	80	1	261	1	454.5	3188	5.195	196.5
Pomaranch80	80	1	472	1	124.5	5357	10.547	569.3
Pomaranch128	128	1	594	1	104.9	8039	16.185	878.4
Moustique	96	1	202	1	476.2	9607	16.078	464.0
Decim80	80	1	1012	0.25	427.3	2603	3.894	157.7
Decim128	128	1	1617	0.25	309.6	3819	6.052	242.2
Edon80×4	80	8	1869	0.0473	207.9	4969	7.775	280.1
Edon80pl	80	1	392	1	243.3	13010	20.467	478.9
AES [4]	128	32	50	2.37	131.2[a]	5398	-	-
AES [5]	128	8	1016	0.124	80.0[a]	3400	-	-

[a] Results are for different CMOS processes (Satoh 0.11, Feldhofer 0.35). Power cannot be scaled reliably between different processes and libraries. The area can be scaled to $0.13\mu m$ for comparison.

Table 3. Derived metrics for maximum clock frequency

Design	Max Throughput, $Mbps$	Estimated Power, μW	Energy/bit, pJ/bit	Area-Time, $\mu m^2 - \mu s$	Tput/Area, $kbps/\mu m^2$	Power-Area-Time $nJ - \mu m^2$
Grain80	724.6	7772	10.72	9.26	107.99	72.0
Grain80×4	2777.7	8569	3.08	3.13	319.33	26.8
Grain80×8	5063.2	9247	1.82	2.24	445.78	20.7
Grain80×16	9876.5	11929	1.20	1.70	588.26	20.3
Trivium	327.9	5618	17.14	40.79	24.51	229.2
Trivium×2	1149.4	10283	8.95	11.85	84.40	121.8
Trivium×4	1895.7	8559	4.51	7.40	135.17	63.3
Trivium×8	3773.6	9360	2.48	4.06	246.62	38.0
Trivium×16	7476.6	9777	1.31	2.20	455.50	21.5
Trivium×32	11228.0	9658	0.86	1.74	571.88	16.9
Trivium×64	22299.6	12677	0.56	1.14	874.13	14.5
F-FCSR-H	3137.2	10255	3.26	7.86	127.13	80.7
F-FCSR-16	5079.3	14503	2.85	8.23	121.38	119.5
Grain128	925.9	15283	16.50	10.39	96.20	158.9
Grain128×4	2339.1	10505	4.49	4.71	211.97	49.6
Grain128×8	4651.1	11646	2.50	2.77	360.52	32.3
Grain128×16	8648.6	13399	1.54	1.91	523.09	25.6
Grain128×32	14479.6	15093	1.04	1.65	604.92	24.9
Mickey128	413.2	12512	30.27	63.21	15.82	790.9
Mickey2(80)	454.5	8701	19.14	36.35	27.50	316.3
Pomaranch80	124.5	6969	55.96	223.01	4.48	1554.3
Pomaranch128	104.9	9063	86.37	397.15	2.51	3599.6
Moustique	476.2	21347	44.83	104.59	9.56	2232.7
Decim80	106.8	6577	61.55	126.28	7.91	830.6
Decim128	77.3	7316	94.52	255.80	3.90	1871.6
Edon80×4	9.8	5670	576.13	2617.43	0.38	14840.8
Edon80pl	243.3	11174	45.92	277.18	3.60	3097.3
AES [4]	311	-	-	90.12	11.10	-
AES [5]	10	-	-	1776.33	0.56	-
Better is:	higher	lower	lower	lower	higher	lower

Table 4. Derived metrics for an output rate of 10 Mbps (estimated typical future wireless LAN)

Design	Clock Frequency, MHz	Estimated Power, μW	Energy/bit, pJ/bit	Area-Time, $\mu m^2 - \mu s$	Tput/Area, $kbps/\mu m^2$	Power-Area-Time $nJ - \mu m^2$
Grain80	10.00	109.45	10.94	671	1.490	73.4
Grain80×4	2.50	34.07	3.40	870	1.150	29.6
Grain80×8	1.25	22.88	2.28	1136	0.880	26.0
Grain80×16	0.63	19.47	1.94	1679	0.596	32.7
Trivium	10.00	175.06	17.51	1337	0.748	234.1
Trivium×2	5.00	93.38	9.34	1362	0.734	127.2
Trivium×4	2.50	49.27	4.93	1402	0.713	69.1
Trivium×8	1.25	29.86	2.99	1530	0.654	45.7
Trivium×16	0.63	18.41	1.84	1641	0.609	30.2
Trivium×32	0.31	16.09	1.61	1963	0.509	31.6
Trivium×64	0.16	16.35	1.63	2551	0.392	41.7
F-FCSR-H	1.25	40.63	4.06	2468	0.405	100.3
F-FCSR-16	0.63	42.25	4.22	4185	0.239	176.8
Grain128	10.00	167.72	16.77	962	1.039	161.4
Grain128×4	2.50	48.69	4.87	1104	0.906	53.7
Grain128×8	1.25	29.92	2.99	1290	0.775	38.6
Grain128×16	0.63	22.36	2.23	1653	0.605	37.0
Grain128×32	0.31	21.85	2.18	2394	0.418	52.3
Mickey128	10.00	310.72	31.07	2612	0.383	811.6
Mickey2(80)	10.00	196.49	19.65	1652	0.605	324.7
Pomaranch80	10.00	569.34	56.93	2777	0.360	1581.2
Pomaranch128	10.00	878.38	87.83	4167	0.240	3660.6
Moustique	10.00	464.02	46.40	4980	0.201	2311.0
Decim80	40.00	619.10	61.91	1349	0.741	835.3
Decim128	40.00	950.52	95.05	1980	0.505	1882.0
Edon80×4	211.25	5761.22	576.12	2576	0.388	14840.5
Edon80pl	10.00	478.88	47.88	6744	0.148	3229.7
AES [4]	4.22	-	-	2798	0.357	-
AES [5]	80.63	-	-	1763	0.567	-
Better is:	lower	lower	lower	lower	higher	lower

Table 5. Derived metrics operating at 100kHz clock (low-end RFID/WSN applications)

Design	Throughput, $Mbps$	Estimated Power, μW	Energy/Bit, pJ/bit	Area-Time, $\mu m^2 - \mu s$	Tput/Area, $kbps/\mu m^2$	Power-Area-Time, $nJ - \mu m^2$	Latency, μs	Power-Area-Latency, $\mu J - \mu m^2$	Power-Latency, nJ
Grain80	0.100	3.29	32.96	67,098	0.0149	221	3,210	70.99	10.58
Grain80×4	0.400	4.47	11.19	21,747	0.0460	97	810	31.54	3.62
Grain80×8	0.800	6.09	7.61	14,198	0.0704	86	410	28.38	2.49
Grain80×16	1.600	9.33	5.83	10,493	0.0953	97	210	32.89	1.95
Trivium	0.100	5.54	55.36	133,747	0.0075	740	13,140	972.87	72.74
Trivium×2	0.200	5.74	28.71	68,092	0.0147	391	6,600	516.14	37.90
Trivium×4	0.400	5.95	14.89	35,061	0.0285	209	3,320	277.22	19.77
Trivium×8	0.800	7.05	8.82	19,127	0.0523	135	1,680	181.35	11.85
Trivium×16	1.600	7.43	4.64	10,259	0.0975	76	860	104.88	6.39
Trivium×32	3.200	10.25	3.20	6,135	0.1630	62	450	90.56	4.61
Trivium×64	6.400	14.31	2.23	3,986	0.2509	57	240	87.62	3.43
F-FCSR-H	0.800	10.58	13.23	30,847	0.0324	326	2,250	587.78	23.81
F-FCSR-16	1.600	18.29	11.43	26,153	0.0382	478	3,080	2357.93	56.34
Grain128	0.100	4.34	43.48	96,250	0.0104	418	5,130	214.70	22.30
Grain128×4	0.400	5.60	14.00	27,588	0.0362	154	1,290	79.74	7.22
Grain128×8	0.800	6.90	8.62	16,127	0.0620	111	650	57.86	4.48
Grain128×16	1.600	9.36	5.85	10,333	0.0968	96	330	51.06	3.08
Grain128×32	3.200	14.77	4.61	7,480	0.1337	110	170	60.12	2.51
Mickey128	0.100	11.17	111.69	261,204	0.0038	2,917	4,170	1216.64	46.57
Mickey2(80)	0.100	7.10	71.08	165,249	0.0061	1,174	2,610	306.58	18.55
Pomaranch80	0.100	16.13	161.35	277,724	0.0036	4,481	4,720	2115.12	76.15
Pomaranch128	0.100	24.80	248.07	416,742	0.0024	10,338	5,940	6140.88	147.35
Moustique	0.100	20.56	205.58	498,044	0.0020	10,239	2020	2068.22	41.53
Decim80	0.025	5.43	217.28	539,689	0.0019	2,931	10,120	741.69	54.97
Decim128	0.025	8.41	336.54	791,977	0.0013	6,663	16,170	2693.63	136.04
Edon80×4	0.005	10.49	2217.91	5,441,651	0.0002	57,132	18,690	5054.66	196.22
Edon80pl	0.100	25.05	250.51	674,421	0.0015	16895	3,920	6622.82	98.20
AES [4]	0.237	-	-	118,054	0.0085	-	500	-	-
AES [5]	0.001	-	-	1,421,064	0.0007	-	10,160	-	-
Better is:	higher	lower	lower	lower	higher	lower	lower	lower	lower

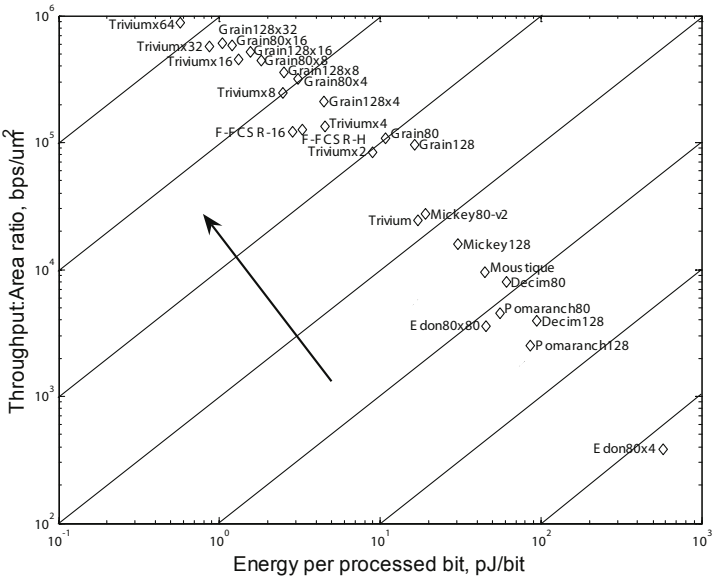

Fig. 17. 0.13μm Standard Cell CMOS design performance metrics at maximum throughput, arrow shows improving performance

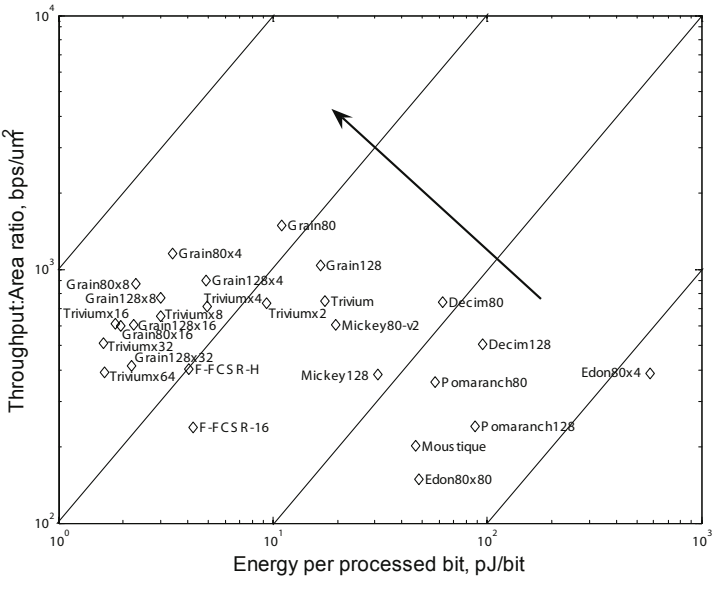

Fig. 18. Performance metrics for notional Wireless-LAN at 10Mbps throughput, arrow shows improving performance

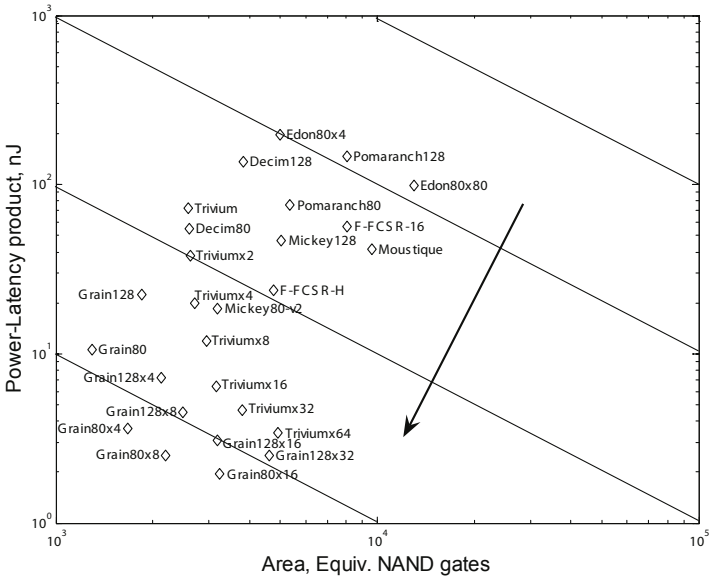

Fig. 19. Performance for low-end RFID/WSN application at 100kHz clock, arrow shows improving performance

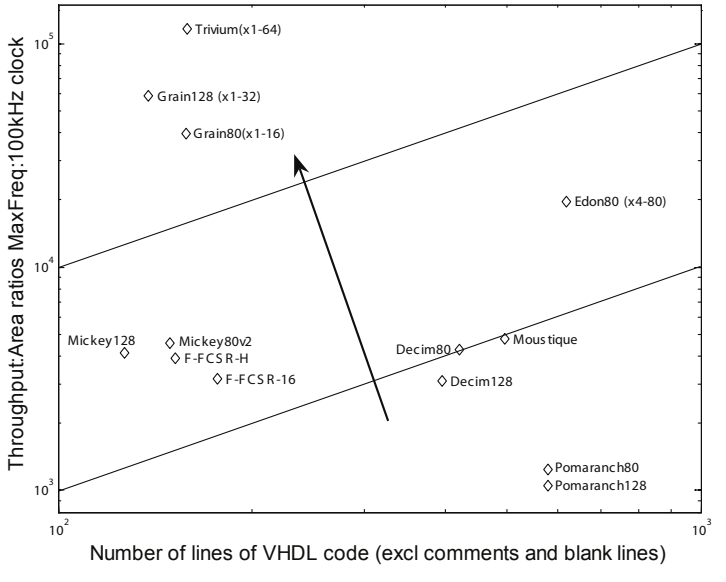

Fig. 20. Flexibility as Throughput: Area for MaxFreq:100kHz clock versus Simplicity as lines of VHDL

5 Require Even Lower Power?

For the primary set of results presented in this paper a typical general purpose standard cell library was used on a $0.13\mu m$ process with standard process options for all the designs. The previous section provides a set of readily comparable results between all of the Phase 3 hardware candidates. This section is only included to demonstrate the advantage to any hardware design by moving to a specialist low power library, selecting low-leakage process options and moving to a more advanced design flow significant power savings can be achieved at the expense of considerable additional design effort. At relatively low clock rates relative to the critical path the core voltage may be reduced accepting longer propagation delays thus further reducing the power consumption. As an example, Table 6 shows the results for Grain and Trivium.

Table 6. Results using a specialist low power library

Design	Grain80×8	Grain128×16	Trivium×8
Interface bits	8	16	8
Core voltage, V	0.8	0.8	0.8
Area, NAND GE	2796	4057	3244
Clock for 10Mbps, MHz	1.25	0.625	1.25
Power (10Mbps), μW	10.710	8.761	15.108
Energy/Bit (10 Mbps), pJ/bit	1.071	0.876	1.511
Power-Area-Time, $nJ - \mu m^2$	11.5	13.6	18.8
Power (100 kHz clk), μW	0.857	1.403	1.209
Power-Latency (100kHz clk), pJ	352	463	2056

At 100kHz Grain80×8 shows approximately a factor of 7 improvement in power-latency product (for the same VHDL source) by changing the library, process options and flow. These results have been included as a reminder that comparison in absolute units between different designs must be made using the same technology, libraries and process options and to demonstrate the low resource nature of stream ciphers using an advanced flow and process options for those who wish to make absolute comparisons with other designs.

6 Evaluation ASIC for Stream Ciphers

At time of writing there has been little published work on the side-channel attacks such as differential power analysis, differential EM-analysis and fault injection techniques. To assist this effort, prototype quantities of an ASIC containing all the Phase 3 hardware candidates have been designed and submitted for fabrication on $0.18\mu m$ CMOS (Fig. 21). All the designs share a common synchronous serial interface (including handshaking) with multiplexers and clock-gating to select the cipher for testing. A total of 15-designs are included as tabulated in Table 7.

Table 7. Cipher implementations

1 Moustique	6 F-FCSR-H	11 Mickey128
2 Edon80	7 F-FCSR-16	12 Pomaranch80
3 Trivium	8 Grain80	13 Pomaranch128
4 Decim80	9 Grain128	14 Grain80 (×8 internally)
5 Decim128	10 Mickey2(80)	15 Trivium (×8 internally)

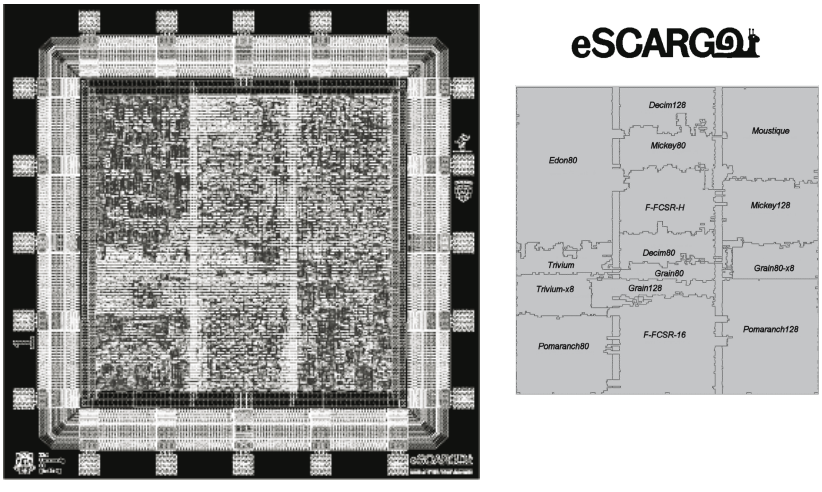

Fig. 21. Layout for eSCARGOt (European Stream Ciphers Are Ready (to) GO)

7 Conclusions

This treatment has considered the entire set of Phase 3 candidates in the hardware profile. Using the two sample applications of a notional future wireless network (WLAN) and low-end radio frequency identification tags / wireless sensor network nodes (RFID/WSN). The table below provides the *first documented attempt to summarise quantifiable results* for all the performance dimensions specified in [2] for each of the candidate ciphers. The authors' overall view relative to the AES is summarised by the left hand column.

And finally, from the results obtained, it is clear that the overall area is dominated by the flip-flops used to store the internal state; this leads to the following general guidance for the development of low-resource hardware stream ciphers:

- the internal state should be composed from key/IV with minimum padding to prevent constant keystream cases,
- the internal state should be stored using a shift register composed of simple D-type flip-flops (without reset/preset, etc),
- there should be a non-linear (filter) function which is trivial in terms of area,

Table 8. Summary of comparative results

	Power-Area-Time Max. clock	Power-Area-Time WLAN	Power-Area-Time RFID/WSN	Flexibility (design space)	Simplicity (code lines)
⋆	Trivium(×64)	Grain80(×8)	Grain80(×8)	Trivium	Mickey128
‿̈	Grain80(×16) Grain128(×32) F-FCSR-H F-FCSR-16	Grain128(×16) Trivium(×8−32) F-FCSR-H	Grain128(×16) Trivium(×8−32)	Grain128 Grain80	Grain128 Mickey2(80) Grain80 Trivium F-FCSR-H F-FCSR-16
‾̈	Mickey2(80) Mickey128 Moustique[a]	F-FCSR-16 Mickey2(80)	F-FCSR-H Mickey2(80) Decim80	Edon80 Decim80 Decim128 Moustique[a]	Decim128 Decim80 Moustique[a]
⁔	Decim80 Edon80 Pomaranch80 Decim128 Pomaranch128	Mickey128 Decim80 Pomaranch80 Decim128 Pomaranch128 Moustique[a] Edon80	Mickey128 Pomaranch80 F-FCSR-16 Moustique[a] Decim128 Edon80 Pomaranch128	F-FCSR-H F-FCSR-16 Mickey2(80) Mickey128 Pomaranch80 Pomaranch128	Pomaranch80 Pomaranch128 Edon80

[a] Moustique is the only self synchronising stream cipher so should be considered of significant merit irrespective of other performance metrics.

- feedback tap selection for the SR should allow for replication of the filter function(s) to permit a more parallel state update and output thus increasing the available P-A-T tradeoffs,
- the trade-off between filter function complexity and mixing phase cycles should be evaluated at an early stage of development, and
- *any* N-bit word *S-boxes* should be avoided as they are a significant consumer of area and power.

Acknowledgement

The authors wish to thank the developers of the candidate ciphers for all their commitment and effort in continuing to refine their submission and further for their assistance in understanding and resolving minor discrepancies between the descriptions and reference designs.

References

1. ECRYPT, Call for Stream Cipher Primitives (April 12, 2005), `http://www.ecrypt.eu.org/stream/call/`
2. Batina, L., Kumar, S., Lano, J., Lemke, K., Mentens, N., Paar, C., Preneel, B., Sakiyama, K., Verbauwhede, I.: Testing Framework for eSTREAM Profile II Candidates. In: SASC (2006), `www.ecrypt.eu.org/stream`

3. Good, T., Chelton, W., Benaissa, M.: Review of stream cipher candidates from a low resource hardware perspective. In: SASC (2006), www.ecrypt.eu.org/stream
4. Good, T., Benaissa, M.: Hardware results for selected stream cipher candidates. In: SASC (2007), www.ecrypt.eu.org/stream
5. Good, T., Benaissa, M.: Hardware performance of phase-III stream cipher candidates. In: SASC (2008), www.ecrypt.eu.org/stream
6. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Nagi, K. (ed.) Transactional Agents. LNCS, vol. 2249, pp. 230–254. Springer, Heidelberg (2001)
7. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES Implementation on a Grain of Sand. IEE Proceedings on Information Security 152, 13–20 (2005)