

Review of stream cipher candidates from a low resource hardware perspective

T. Good, W. Chelton and M. Benaissa

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

Abstract. This paper presents hardware implementation and analysis of a carefully selected sub-set of the candidate stream ciphers submitted to the European Union eStream project. Only the submissions without licensing restrictions have been considered. The sub-set of six was defined based on memory requirements versus the Advanced Encryption Standard and any published security analysis. A number of complete low resource designs for each of the candidates are presented together with FPGA results for both Xilinx Spartan II and Altera Cyclone FPGAs, ASIC results in terms of throughput, area and power are also included. The results are presented in tabular and graphical format. The graphs are further annotated with different cost functions in terms of throughput and area to simplify the identification of the lowest resource designs. Based on these results, the short-listed six ciphers are classified.

Keywords. Stream Ciphers, Hardware, FPGA, ASIC, Performance Evaluation.

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 tasked with seeking a strong stream cipher. Thirty-four candidate ciphers have been submitted and are currently being evaluated in terms of security.

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.

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.

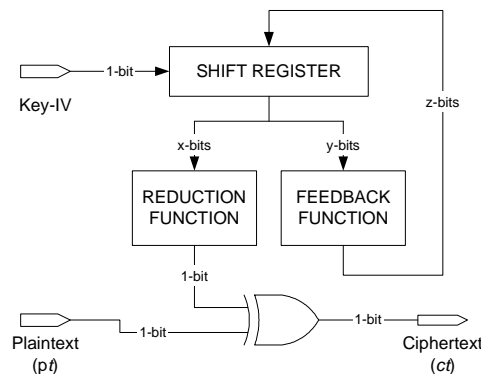


Fig. 1. Generic stream cipher

The 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). To aid this, the call defined the Advanced Encryption Standard (AES) as a benchmark and submissions should use less resource and be “faster” than the AES.

There has been very little discussion or comparison of the hardware implementations of the candidate stream ciphers to date. The majority of the effort has been correctly directed towards the cryptanalysis of the algorithms. However, as will be shown in this paper, even early hardware results can provide a timely method for selecting a sub-set of the candidates for more intensive scrutiny. It is hoped that this paper will allow effort to be directed towards the “low resource” hardware submissions so that these are proved secure or broken more or less in the order of hardware “performance”.

Section 2 of this paper describes how the list of ciphers was sifted to locate a smaller set for hardware evaluation. Briefly, this was in terms of the ciphers commercial status (i.e. “free-for-all”), the amount of internal state and identification of any large look-up tables (S-boxes). This is followed, in section 3, by details of the method used to evaluate the hardware performance and in section 4, the results for the selected ciphers. Where the hardware results were affected by the developers’ choice of initialisation, this is highlighted, as tweaks to initialisation are permitted within the scope of the eStream call. Finally, in section 5 some conclusions are drawn. Appendix A (due to page number restrictions) gives details of each of the designs together with additional suggestions on possible ‘tweaks’ aimed at further reducing the hardware requirements.

Considering the results for Xilinx FPGA, Altera FPGA and power results for 0.13 μ m Standard Cell ASIC allows some strong conclusions to be drawn. Of the ciphers considered, this analysis excluded the “commercial” ones, Grain and Trivium can be ranked as the most efficient followed by Sinks, Mosquito and Hermes. The raw results have been included to permit others to choose their own metrics and allow for further comparisons.

2 Selection Process

Any selection process will be inevitably coloured by the authors own position and beliefs. In the interest of academic fairness we will state ours:

1. We have no affiliation or predisposition to any of the candidate algorithms or their authors.
2. We would like to see the successful stream cipher be “free for all” to use. Thus we have not directed any efforts towards any of the “non-free for all” candidates.
3. We are concerned only with low resource hardware results and believe that both FPGA and ASIC results are important.
4. We do not wish to make any security claims about any of the candidates and where ciphers have been disregarded from this analysis on the grounds of security weakness we have relied on our interpretation of the results posted on the eStream web site.

There are some 34 candidate primitives submitted. From the information provided on the eStream web site [1], nine of the candidates were subject to some form of licensing or restriction so excluded from our analysis. For a further seven of the candidates the published cryptanalysis had highlighted, in the authors’ view sufficient weakness for it to be excluded from this analysis. To reiterate, this is the authors’ view for the purpose of reducing the number of candidates to implement in hardware and is not in any way concerned with the formal selection process by the eStream project.

The developers submitted their algorithms to either the software, hardware or both profiles. From initial examination of a few of the “software” submissions it was recognised that although these had not been submitted to both the software and hardware profiles they may have a low resource hardware implementation so should be considered.

For the eighteen remaining candidates, the reference designs and papers were examined in detail to determine any hardware results reported by the developers together with the amount of internal storage (in bits) and any memory requirement for “S-box” substitution operators. It was further noted, if any S-box had known or likely logic implementation which could be utilised to avoid a relatively large memory. In the case where the “S-box” is generated using the key or otherwise manipulated making implementation as a ROM not possible it was considered as part of the internal state.

A view was taken on what would be acceptable as low resource with the aim of approximately reducing the remaining candidates to produce our “top six”. As a baseline for comparison we considered an earlier low resource FPGA implementation of the AES [2] which supported three feedback modes (OFB, CTR and CFB)

thus represents a well understood and relatively secure stream cipher. For a second baseline, the standard cell ASIC design of Feldhofer [3] was selected.

FPGA results can be obtained more rapidly than ASIC results so the evaluation was started with consideration of the FPGA performance. The FPGA AES baseline case was viewed as the limiting case that candidates must outperform. This implementation [2] required 704 bits of internal state and a 2kbit S-box (implemented using composite field logic). This design supported three feedback modes, if a single mode were selected such a design would only require approximately 400 bits of storage. Consequently, baseline limits of 400 bits of internal state and 2kbit of fixed-valued S-box were selected.

This selection process may at first glance appear relatively crude, however, in hardware, the area occupied by a D-type Flip Flop, which is the most likely means of storage of internal state bits, is relatively large compared with combinational gates, thus, will account for a significant proportion (>50%) of the area of any low-resource implementation. A similar argument can be made for the area consumed by requiring a few kilo-bits of memory (either RAM or ROM).

Table 1 lists all the candidates in alphabetical order together with the authors' reasons for selection or non selection for further analysis.

Table 1. Summary of selection of candidates

Cipher	Profile	Free for all	Internal state (bits)	Key & IV bits	S-box bits	Cut	Notes
ABC	1	yes	160 +KE (1024)	128 128	0	✗	For software broken but still $>2^{80}$ so ok for hardware, however, Key Expansion of 32x32-bit words (1024 bits) and not sure from paper or code what is the "standard key expansion".
Achterbahn	2	yes	-	-	-	✗	broken, linear ca in 2^{73}
CryptMT/ Fubuki	1	no	-	-	-	✗	not free for all
Decim	2	no	-	-	-	✗	broken, 2^{29} IV to recover key
Dicing	1	yes	768	128/256 128/256	2k	✗	disputed ca, large internal state
Dragon	1	yes	192	128/256 128/256	16k	✗	large "randomly" generated s-boxes, disputed ca
Edon80	2	no	-	-	-	✗	key period doubts, not free for all
F-FCSR	1&2	yes	-	-	-	✗	broken, key recovery attack
Frogbit	1A	no	-	-	-	✗	not free for all
Grain	2	yes	160	80 63	0	✓	ok, linear ca which required $2^{61.4}$ bits of keystream
HC-256	1	yes	64k	256 256		✗	2 x huge s-boxes (64 kbit) 1024 bit subtraction
Hermes8	1&2	yes	224	80 184	2k or logic	✓	ok, uses AES s-box
LEX	1&2	no	-	-	-	✗	not free for all, key recovery in 2^{61} IVs
MAG	1&2	yes	-	-	-	✗	broken, low complexity distinguishing attack
MICKEY	2	yes	-	-	-	✗	key stream entropy loss
Mir-1	1	yes	2432	128 64	2k or logic	✗	too much internal state, uses AES s-box with key to generate own s-box

Table 1 continued...

Cipher	Profile	Free for all	Internal state (bits)	Key & IV bits	S-box bits	Cut	Notes
Mosquito	1A&2A	yes	128	96 104	0	✓	ok
NLS	1A&2A	yes	1184 ?	256 256	8k	✗	two S-boxes (total 8x32 bit s-box). too much internal state & s-box!
Phelix	1A&2A	yes	352	256 128	0	✓	ok
Polar Bear	1&2	yes	168	128 <248	2k or logic	✗	5 round AES + RC4, one round of AES is still relatively large
Pomaranch	1&2	yes	184 ?	128 144	4k or logic	?	Unsure of status of broken then fixed submissions
Py ("Roo")	1	yes	10400	256 128	0	✗	too much internal state
Rabbit	1&2	no	-	-	-	✗	not free for all
salsa20	1	yes	512	256 64	0	✗	Too much internal state, disputed ca
Sfinks	2A	yes	256	80 80	64k or logic	✓	ok
Sosemanuk	1	yes	512	128/256 128	0.5k	✗	Too much internal state
SSS	1A&2A	yes	-	-	-	✗	broken by J. Daemen 10 secs on a PC
Trbdk3 yaea	1&2	no	-	-	-	✗	not free for all
Trivium	2	yes	288	80 80	0	✓	ok, linear ca to date shows strength
TSC-3	2	yes	-	-	-	✗	broken, linear ca in 4 mins on a PC
VEST	2A	no	-	-	-	✗	not free for all
WG	2	yes	-	-	-	✗	broken, chosen IV attack
Yamb	1&2	yes	>3k	256 128	0	✗	Too much internal state
ZK-Crypt	2	No	-	-	-	✗	not free for all

This first-pass selection, as illustrated in Table 1 above, has resulted in a short list of six for further investigation:

Grain, Hermes-8, Mosquito, Phelix, Sfinks and Trivium.

3 Hardware Implementation

3.1 Method

Of the remaining six selected candidates, three are in the 1A and 2A profiles which offer a message authentication code (MAC) in addition to the stream cipher output. In order to achieve a fair comparison against the other candidates, the designs were implemented as pure stream ciphers out without any of the additional resources required for supporting MAC generation.

Some of the developers quoted hardware results to different degrees of confidence from “rough estimates” to detailed implementation details. However, there was no consistent methodology used and results differed greatly depending on a variety of factors such as the number of gates or transistors required to make up a D-type flip-flop and the supported interfacing. Such variations make it impossible to directly compare the developers’ hardware results. Thus the decision was taken to develop an independent set of hardware results.

Stream ciphers are required to operate on a stream of bits thus the decision was taken to use a synchronous serial style of interface for input and output of the plaintext and ciphertext. More flexibility was adopted for entry of the key to be either serially or utilise a short word parallel format (eg 8 or 32 bits at a time).

The designs were developed for low resources, sacrificing throughput in the interests of saving area. First, results were obtained for Xilinx FPGA using ISE version 6.3 and Altera FPGA using Quartus II version 5.0 both of which use 0.13 μm CMOS processes. In line with the low resource nature of eStream, the smaller Spartan-II devices were selected (XC2S15, XC2S30 and XC2S50). The smallest available Altera Cyclone (EP1C3T100C7) is considerably larger than the smallest Spartan II parts thus the same part was used for all the designs. ASIC results for a commercial 0.13 μm standard cell process were obtained using a Cadence Physically Knowledgeable Synthesis (PKS) version 5.14 design flow for Synthesis, Place and Route (SP&R) using PKS, BuildGates, AmbitWare, standard cell technology library and SiliconEnsemble. The flow incorporated worst-case parasitic extraction and back annotation using foundry data. Verification included static timing analysis, design rule checks, generation of expected switching data using ModelSim and power results from Cadence LPS.

3.2 Defining Performance

The results quoted for the FPGAs are actual post place and route results (not synthesis estimates). The maximum clock rate for the design together with the selected FPGA device and its area utilisation are given. However, due to the richness of modern FPGA fabric this alone would not be representative of the likely device performance for ASIC so a further gate based analysis is given.

For this analysis, throughput performance was measured in millions of bits per second (Mbps) for the output of ciphertext neglecting any initialisation time. The area of an FPGA is normally measured in terms of its cell usage: slices for Xilinx and Logic Elements (LE) for Altera.

To avoid specific metrics for individual devices, it is proposed to use the “gates” metric for measuring area. In this paper, one “gate” is equivalent to the area occupied by a two input NAND gate (6 transistors). Thus a two input XOR gate typically occupies an area equivalent to 2.33 NAND gates (14 transistors). The implementation of a D-type flip-flop is much more variable depending on what auxiliary inputs (eg preset, clear, clock enable) are required. In this paper an 8 gate equivalent for the flip-flop was chosen.

To allow readers to calculate their own gate count for different gates-per-flip-flop, the quoted gate results are separated into two figures one for flip-flops and the second for all other gates. On many processes, by sacrificing flip-flop functionality such as preset, reset and clock-enable, the overall “gate” count may be reduced.

These relatively modern FPGA devices have a rich fabric supporting a number of distributed memory storage primitives. The effectiveness of these, in particular the Xilinx SRL16, depends on precisely how the algorithm uses its memory storage elements. Some ciphers make good use of such FPGA-area saving components and others less so. There is a further complication in that the FPGA synthesis tools generally attempt to yield the most “adaptable” design fitting within the given speed and area constraints. This is done to minimise the impact of relatively minor design changes for a waterfall development cycle often used in prototyping. The area constraint is typically defined with a rectangle thus for low resource designs the area utilised is dominated by how well the design tessellates with the chosen rectangle rather than its minimum resource utilisation.

To overcome this issue, an alternative approach was taken rather than to simply quote the number of “slices” reported by the post place and route report. In our second approach, the map report was examined and the number of LUT Function Generators (FG) and associated resource such as carry-chains (counted as equivalent to an FG) were extracted together with the number of D-type flip-flops (FD). On some FPGAs, the LUTs may also be configured as memory resources (ROM or RAM) these figures were also obtained from the map report. Of particular concern was how to correctly account for the use of the SRL16 (16x1 bit shift register) resource. The decision was taken to only account for this in terms of the actual number of bits used for the given design. For example, if only a 6 bit SR was needed then account for this as 8x6 gates rather than 8x16. This approach is believed to be equivalent to a gate level analysis and is more representative of the likely ASIC results.

From the FD, LUT and memory (MEM) values (SRL, ROM & RAM) an equivalent ASIC 2-input NAND gate count was estimated as follows:

$$\text{gates} = 6 \times \text{FG} + 8 \times \text{FD} + 8 \times \text{bits} \times \text{MEM}$$

In general terms, there are two different goals for a “low resource” design. Firstly, designers may be concerned with minimising the peak power consumption. This is typical in inductively powered contact-less smart cards. However, for battery powered systems it is more important to minimise the total energy consumption. For the latter, a typical goal is the minimisation of the power-area product.

Both design objectives are sensitive to area, so here, as this is a first attempt at comparison between the stream ciphers it was chosen to simply minimise the area. A typical academic metric for efficiency would be to minimise the area-time product. “Time” being the time taken to perform the cryptographic operation. The power consumption, for CMOS, is dominated by the number of transitions per second (thus datapath width and the clock frequency).

The simple area-time product metric favours highly parallel pipelined loop unrolled designs which generally would not be described as “low resource”. Area alone could be considered as the performance metric for a low resource design however would not discriminate between two designs of differing throughput which required the same area. A suitable metric must include both throughput and area weighted in such a way to avoid favouring simply unrolling a design to improve its “performance”.

One option is to select a throughput based loosely on the currently emerging wireless data standards, say 5 to 15 Mbps and develop implementations of the ciphers to meet this rate by selecting the appropriate clock frequency. However, it is also common to design for a higher rate, say 100Mbps and then calculate power consumption at a reduced clock rate.

The formulation of such a metric would be entirely subjective, thus the decision was taken to present the results graphically with a set of lines indicating constant metric value for different formulations of the metric and leave it for potential readers to make their own judgements.

The ASIC power results were obtained by stimulating a cell-level back-annotated simulation model of the design under test with random test vectors. ModelSim was used to obtain switching data in terms of a value change dump. This data was converted to a suitable format and combined with foundry supplied power models for the cells to yield the expected modelled power results. A basic MonteCarlo analysis was carried out by repeating the results a number of times with different test vectors in order to validate the accuracy of the results (<1% error). The results incorporate both initialisation and operational phases of the design under test.

4 Results

4.1 FPGA Implementation Results

Table 2 summarises the results obtained for each of the selected ciphers. In the interest of completeness, the original developers’ results are also presented where available. Details of the designs adopted and any design modifications made are illustrated in Appendix A for each of these ciphers. For readers interested solely in FPGA design then their attention is drawn to the device, slices and LE results. The smallest available Xilinx Spartan II device is the XC2S15, only the AES-B, Trivium-1, Grain-1 and Mosquito-B designs will fit within this device. Fig. 2, effectively shows throughput versus area for the Xilinx Spartan II FPGA (0.13 μm process) and Fig. 3 the corresponding results for the Altera Cyclone FPGA (0.13 μm process).

For these designs, the Altera results are generally the faster and in terms of throughput and the relative performance of the different designs more closely follows the gate level analysis. An approximate equivalence of 2 LE = 1 SLICE may be used to perform a crude comparison in terms of area. Thus, the processor style architectures (Phelix-C, Hermes8) occupy less area on the Xilinx FPGA.

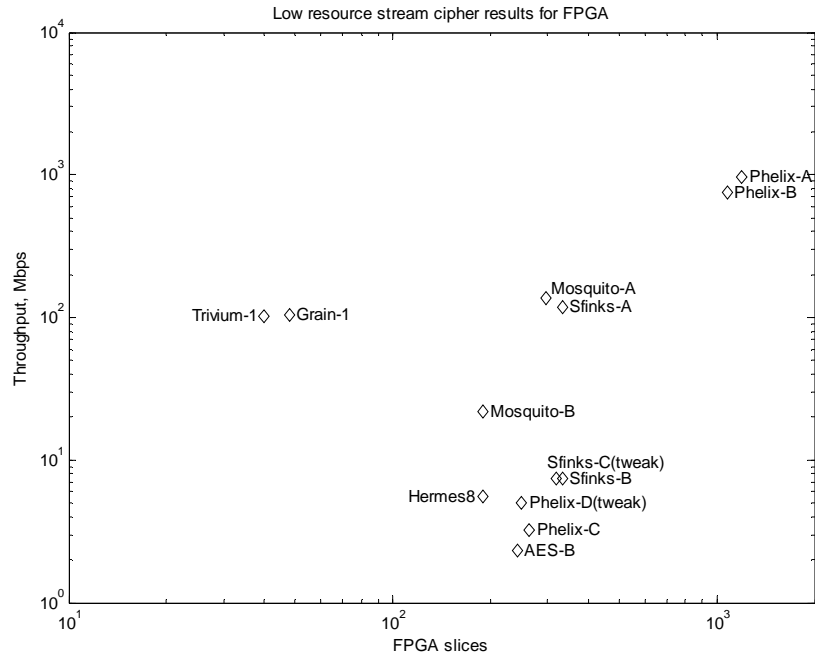


Fig. 2. Xilinx FPGA results

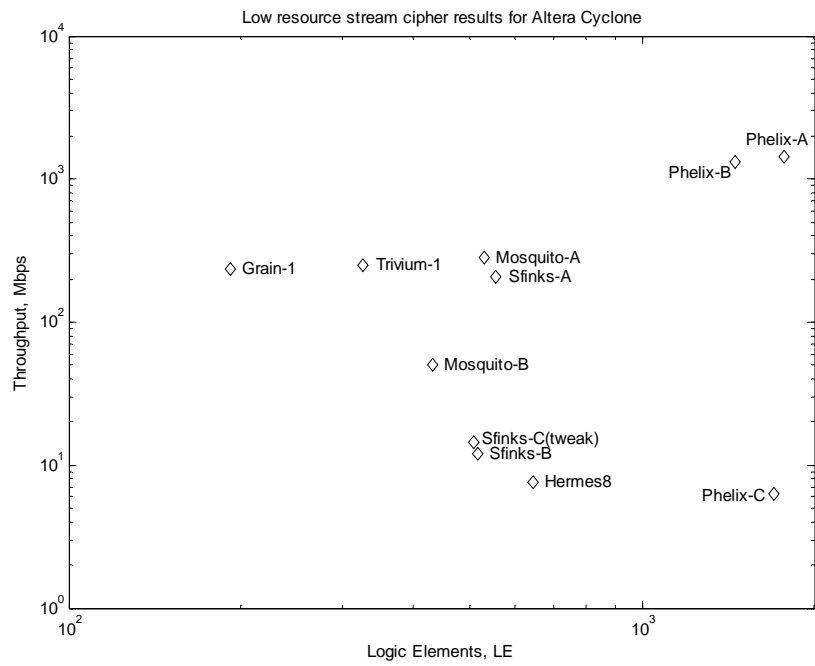


Fig. 3. Altera FPGA results

However, for ASIC designers, the “gates” column is more likely to be of interest. This clearly shows that Grain-1 and Trivium-1 are by far the smallest, yet still provide good throughput figures.

Table 2. FPGA results and gate level analysis

Cipher Design	Authors hardware results	Xilinx Spartan II FPGA	Altera Cyclone FPGA	Equiv. gates estimate	Notes
AES-A	0.35 μ m CMOS (Philips) 9 Mbps, 3,500 “gates” [3]	(ASIC) 9 Mbps	no result	6000	Gate count increased by 2500 to allow for feedback mode support
AES-B	Our basis for comparison [2]	XC2S15-5 2.34 Mbps 242 slices	no result	10426	Our ASIP design supporting OFB, CTR and CFB modes
Trivium-1	3488 gates [4]	XC2S15-5 102 Mbps 40 slices	EP1C3T-C7 249 Mbps 327 LE	2682	
Grain-1	ALTERA: 1435 “gates” MAX3000A 49Mbps MAX-II 200 Mbps Cyclone 282 Mbps [5]	XC2S15-5 105 Mbps 48 slices	EP1C3T-C7 335 Mbps 191 LE	1714	
Mosquito-A	Xilinx Virtex I 179 Mbps, 252 CLB & other FPGA results [6]	XC2S30-5 137 Mbps 298 slices	EP1C3T-C7 280 Mbps 530 LE	6844	(A) Pipelined as developers’ paper
Mosquito-B		XC2S15-5 22 Mbps 190 slices	EP1C3T-C7 50 Mbps 431 LE	4178	(B) Our resource shared design (common hardware for logic stages 2-5)
Phelix-A	“Rough” estimates of 2Gbps, 20,000 gates [7]	XC2S100-5 960 Mbps 1198 slices	EP1C3T-C7 1440 Mbps 1772 LE	20404	(A) Full-round 160-bit design, as per developers paper
Phelix-B		XC2S100-5 750 Mbps 1077 slices	EP1C3T-C7 1312 Mbps 1455 LE	18080	(B) Our half-round 160-bit design
Phelix-C		XC2S30-5 3.26 Mbps 264 slices	EP1C3T-C7 6.31 Mbps 1697 LE	12314	(C) Our 32-bit datapath, control adversely affects area
Phelix-D		XC2S30-5 ~5 Mbps ~250 slices	no result	~8800	(D) Estimate initialisation was tweaked to simplify architecture
Sfinks-A	5265 gates (excluding MAC) [8]	XC2S30-5 118 Mbps 334 slices	EP1C3T-C7 207 Mbps 556 LE	5904	(A) Pipelined as per developers’ paper
Sfinks-B		XC2S30-5 7.4 Mbps 334 slices	EP1C3T-C7 12.0 Mbps 517 LE	4910	(B) Our design comprising resource sharing in inversion – frustrated by requirements of initialisation (thus not efficient design)
Sfinks-C		XC2S30-5 7.4 Mbps 319 slices	EP1C3T-C7 14.6 Mbps 508 LE	3946	(C) Tweaked to remove feedback delay needed for initialisation
Hermes8	0.35 CMOS 4,026 gates (std cell) [9]	XC2S30-5 5.6 Mbps 190 slices	EP1C3T-C7 7.6 Mbps 645 LE	5022	Our 8-bit datapath architecture inclusive of control.

The results can be even more clearly expressed graphically in terms of throughput and area. In terms of area the further left, the smaller the design. In terms of speed the higher up the faster the design. As discussed in the method section of this paper, some “performance” metric would be most expedient.

Fig. 4 depicts the results with the dashed lines show constant speed versus area for each given design, however, this metric favours loop-unrolled and pipelined architectures so may not be considered the most appropriate.

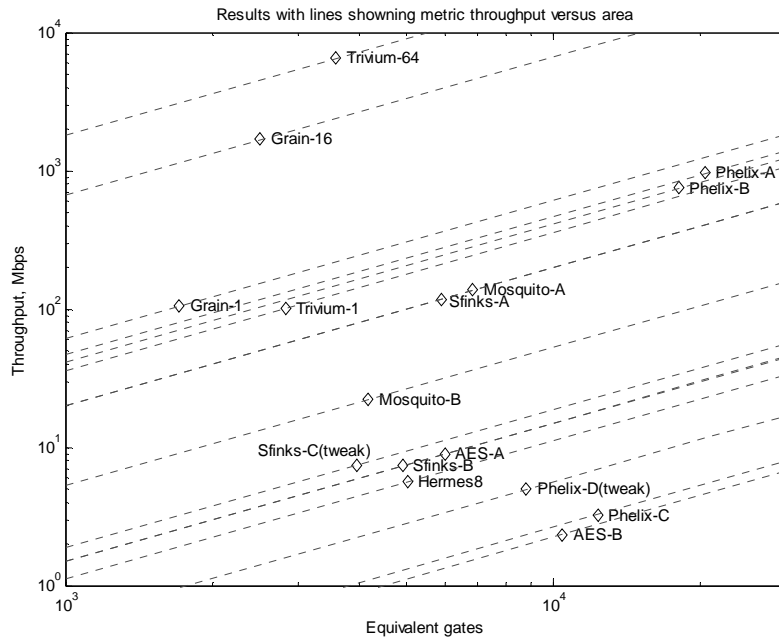


Fig. 4. Results annotated with lines of constant throughput versus area

The performance metric can be skewed more in favour of area by raising the area to a higher power than the throughput. Fig. 5 once again shows the low resource designs however this time the dashed lines are lines of constant area^2 versus speed.

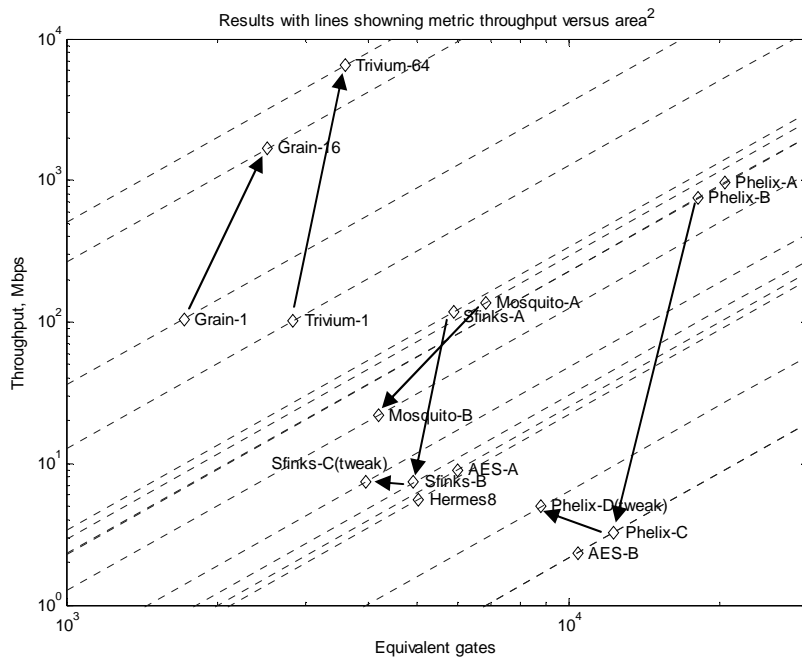


Fig. 5. Results annotated with lines of constant throughput versus area^2

However, as can be seen by the Grain-n and Trivium-n designs, the metric still favours unrolling and parallelism. The area is now raised to a still higher power ($\text{area}^{7.3}$) such that the resulting metric is now approximately neutral to the parallel construction of the smallest candidate. The resulting graph is presented as Fig. 6.

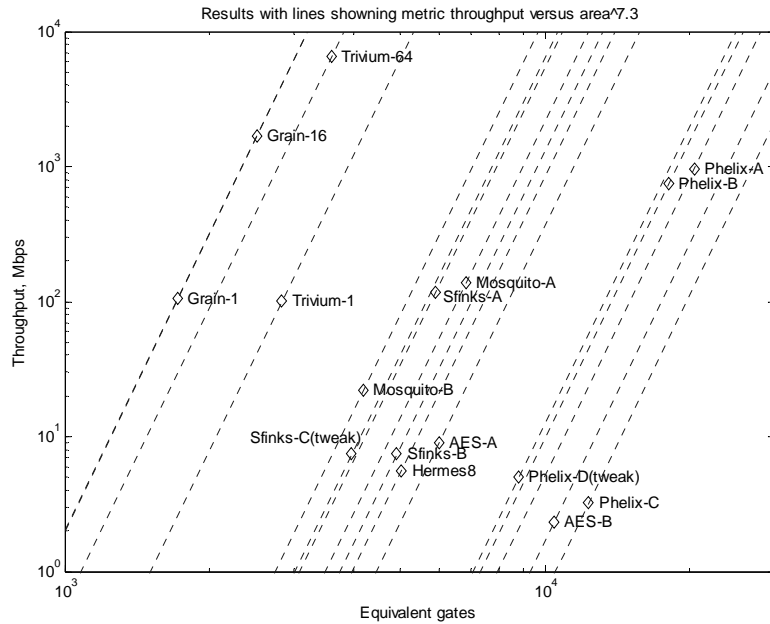


Fig. 6. Results annotated with lines of constant throughput versus area^{7.3}

This still may not be considered to be sufficiently area skewed so a final graph, Fig. 7, is presented for raising the area to the fifteenth power (as an extreme example). This is done to further illustrate that irrespective of the choice of cost function that both Grain and Trivium stand out as the lowest resource.

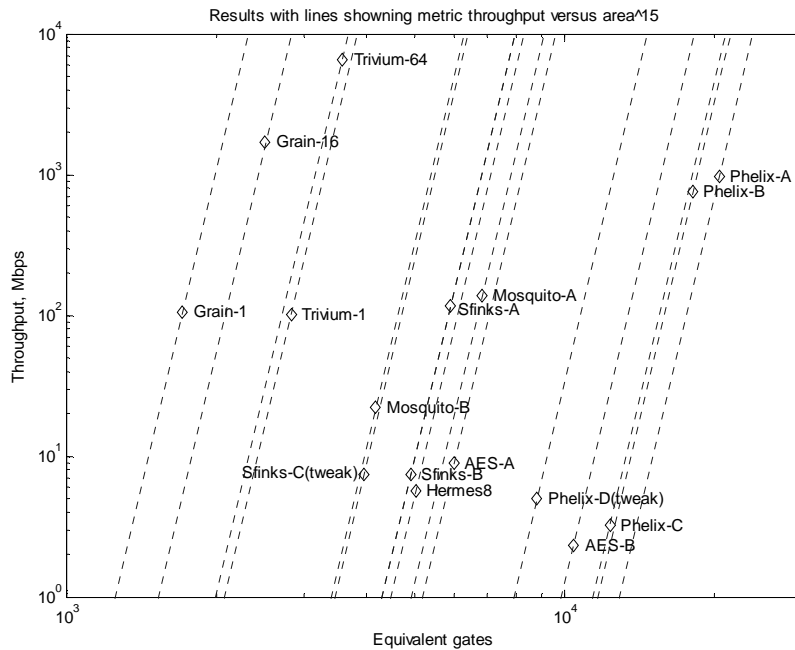


Fig. 7. Results annotated with lines of constant throughput versus area¹⁵

For low resource design the metric “areaⁿ * time” has been presented. The choice of a suitable value of n is subjective thus illustrative examples have been given for selected values between 1 and 15. It has been shown for the smallest candidate that a value of n=7.3 makes the metric neutral to pipelining. This value should be considered as the upper limit for n. A sensible choice would be to choose a value somewhere between the extremes of area * time (n=1 and 7.3), say, on a purely subjective basis n=2. However, irrespective of the precise value of n, as shown by the different results graphs, conclusions can be drawn and the selected ciphers categorised.

4.2 ASIC Results

To confirm the gates analysis above, obtain power results, and also for the sake of completeness, ASIC results were also obtained for a 0.13 μm standard cell process using the Cadence Physically Knowledgeable Synthesis (PKS) flow. The results shown (Table 3) are the expected modelled results for the technology. The area is the occupied core area including routing. For readers wishing an ASIC 2-input NAND gate estimate simply multiply the area in μm^2 by 0.193. The power results were obtained using switching data resulting from loading the key and IV followed by initialisation and the encryption of a 10kbit stream of random data. Statistics from three different runs were compared in a basic Monte-Carlo analysis to validate the power results.

Table 3. ASIC throughput-area-power results

Design	Throughput, Mbps	Clock Period, ns	Critical path delay, ns	Area, μm^2	Power, mW
Trivium-1	1	1000	2.39	15,058	0.0347
	10	100			0.227
	100	10			2.154
Grain-1	1	1000	2.18	8,073	0.0238
	10	100			0.156
	100	10			1.476
Mosquito-A	1	1000	3.11	52,155	0.178
	10	100	3.15	52,023	1.027
	100	10	3.15	52,023	9.520
Mosquito-B resource shared	1	200	2.16	24,903	0.137
	10	20	2.14	24,903	1.136
	100	(2)		(no result)	
Sfinks-A	1	1000	9.43	33,167	0.253
	10	100			2.207
	100	10			21.75
Sfinks-B resource shared	1	200	12.05	32,702	2.211
	10	20	12.01	32,702	21.83
	100	(2)		(no result)	
Hermes8	1	125	7.36	35,672	0.429 (tbc)
	10	12.5	7.34	35,773	3.834 (tbc)
	100	(1.25)		(no result)	

The results are summarised in terms of power versus area in Fig. 8. This figure shows that in terms of power-area efficiency Grain is the most efficient closely followed by Trivium. It also clearly shows the advantage of utilising a resource shared design for Mosquito. The power results again highlight the difficulty in attempting resource sharing for Sfinks (point too far off graph to plot).

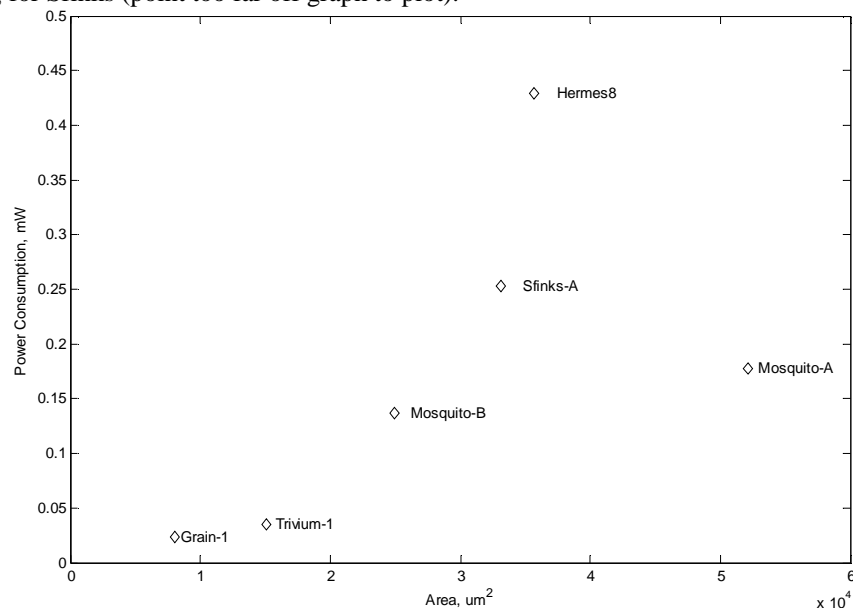


Fig. 8. ASIC results of power consumption versus area for designs operating at 1Mbps

5 Conclusions

Irrespective of how the results are presented Grain and Trivium are the smallest and most efficient designs and have straight forward parallel implementation which may ultimately be desirable to further enhance throughput and achieve improved energy per bit performance. The authors wish to urge those interested in the stream cipher project to analyse these thoroughly from a security perspective.

There is much more debate on which are the next ciphers to “perform” the best so they have been simply grouped together (Mosquito, Sfinks, Hermes8). More controversial, would be where to rank Phelix, in this paper it has been categorised as “moderate resource” due to its size not withstanding its higher throughput.

The authors of this paper do not wish to pass any comment (or expend effort) on those candidate ciphers which are not “free-for-all”. It is left to others to carry out a similar analysis.

In summary, in terms of “low resource” hardware the considered candidates may be conveniently and fairly grouped as follows:

Category	Candidate ciphers for low resource <u>hardware</u>
Lowest Resource, High speed (~100Mbps)	Grain, Trivium
Low resource, moderate speed (~10Mbps)	Mosquito, Sfinks, Hermes-8
Moderate resource (~1000Mbps)	Phelix
High resource or broken	ABC, Achterbahn, Dicing, Dragon, F-FCSR, HC-256, MAG, MICKEY, Mir-1, NLS, Polar Bear, Pomaranch, Py (“Roo”), salsa20, Sosemanuk, SSS, TSC-3, WG, Yamb
Commercial i.e. “not free for all” (not considered in this treatment)	CryptMT, Decim, Edon80, Frogbit, Lex, Rabbit, Trbdk3, Vest, ZK-crypt

In summary, the purpose of this analysis is to encourage the security analysis community to direct their efforts towards analysing the security of the lowest resource candidates first before moving on to those requiring more resources. The benefits to this approach are two fold: firstly avoids wasted effort analysing a candidate which may not be considered to be “low resource” and secondly early rejection of those with low resource on security grounds will enable the hardware engineers to focus on adding side channel resistance to the remaining lower resource ciphers again avoiding wasted effort.

References

- 1 The eStream web site, <http://www.ecrypt.eu.org/stream/>
- 2 T. Good and M. Benaissa, "AES as a stream cipher on a small FPGA", to appear ISCAS 2006.
- 3 M. Feldhofer, J. Wolkerstorfer and V. Rijmen, "AES implementation on a grain of sand", IEE Proc. Info. Sec, Vol. 1, pp 13-20, 2005
- 4 C. de Canniere and B.Preneel, "Trivium Specifications", <http://www.ecrypt.eu.org/stream/>
- 5 M.Hell, T.Johansson and W.Meier, "Grain – a stream cipher for constrained environments", <http://www.ecrypt.eu.org/stream/>
- 6 J. Daemen and P. Kitsos, "Submission to ECRYPT call for stream ciphers: the self-synchronizing stream cipher Mosquito", <http://www.ecrypt.eu.org/stream/>
- 7 D. Whiting, B.Schneier, S.Lucks and F.Muller, "Phelix: fast encryption and authentication in a single cryptographic primitive", <http://www.ecrypt.eu.org/stream/>
- 8 A.Braeken, J.Lano, N.Mentens, B.Preneel and I.Verbauwhede, "SFINKS: A synchronous stream cipher for restricted hardware environments", <http://www.ecrypt.eu.org/stream/>
- 9 U. Kaiser, "Hermes-8", <http://www.ecrypt.eu.org/stream/>
- 10 NIST, "Recommendation for block cipher modes of operation", Special Publication 800-38A, 2001, <http://www.nist.gov/>
- 11 NIST, "The Advanced Encryption Standard", FIPS-197, <http://www.nist.gov/>

Acknowledgements

Funding by the UK Engineering and Physical Sciences Research Council (EPSRC) is acknowledged.

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

Appendix: A. Design Details

In the following sections, a brief description of each of the considered candidate algorithm is given and should be read in conjunction with the developers' original paper. Our designs and implementation results for each are given including where appropriate suggestions of possible tweaks to initialisation which may permit reduction of the required hardware resources.

A.1 AES (baseline)

The AES in a suitable feedback mode (eg Output Feedback) could be used as a "tried-and-tested" stream cipher. However, it is evident from the call that for "low resource" there is an aspiration to do better. Thus the AES forms one of the best baselines to date in terms of known security and as it was stated in the call as a suitable basis for comparison for the software profile it would be a sound judgement to use its low resource hardware implementations as a basis for comparison for the hardware ones too.

A previous FPGA design by the authors [2] looked at an 8-bit ASIP which supported three of the recognised [10] feedback modes for the Advanced Encryption Standard [11]. The modes were Output Feedback (OFB), Counter (CTR) and Cipher FeedBack (CFB) all of which generate a key stream which is then combined with the plain/cipher text using the XOR operation. This is an example of using a block cipher (such as AES) in a feedback mode to make it suitable for stream cipher applications. The use of block memory can be allowed for by adjusting the slice count with a cost of 32bits/slice for block memory usage.

A recently published [3] design for the AES showed that it is possible to construct a low resource ASIC to perform the core functions of the AES. With suitable additional memory, logic and interfacing it could operate autonomously in one of the feedback modes (OFB, CTR or CFB) to provide a low resource stream cipher. The additional logic including shift registers to support serial I/O and additional storage for key and IV required by a feedback mode such as OFB or CFB is estimated to total an additional 2500 gates.

Table 4. Implementation results for the AES

Design	Details	FPGA results	Gate level analysis (for Xilinx)
AES-A	Feldhofer's ASIC design, 3500 gates @ 9 Mbps on 0.35um ASIC Additional logic for feedback mode and serial I/o ~2500 gates	(ASIC result)	throughput: 9 Mbps approx. flip flop gates: 4848 approx. other gates: 1152 approx. total gates: 6000
AES-B	all: FG 211 FD 184 RAM 608bits ROM 200x16 bits	Xilinx (ISE): device: XC2S15-5 clock: 70 MHz bits/cycle: 128/3828 slices: 242 (120 + 2xBLKRAM)	throughput : 2.34 Mbps block RAM gates: 5220 block ROM gates: 2468 other flip flop gates: 1472 other gates : 1266 total FPGA gates: 10426

A.2 Trivium

Trivium [4] is a stream cipher consisting of three shift registers with interconnected non-linear feedback functions to form a recognisable Substitution-Permutation-Network and a final linear function is used to create the keystream. The shift registers are of different lengths (93, 84 and 111 bits) and all the feedback functions only combine five taps.

The feedback and output functions may be expressed in terms of their constituent taps as follows:

$$\begin{aligned} t1(S) &= S_{66} + S_{91}.S_{92} + S_{93} + S_{171} \\ t2(S) &= S_{162} + S_{175}.S_{176} + S_{177} + S_{264} \\ t3(S) &= S_{69} + S_{243} + S_{286}.S_{287} + S_{288} \\ z(S) &= S_{66} + S_{93} + S_{162} + S_{177} + S_{243} + S_{288} \end{aligned}$$

To load the key the bit stream is (externally) prepared by padding out the key and IV to the required 288 bits as follows:

$$S_{1..288} = K_{1..80}, '0'_{14}, IV_{1..80}, '0'_{111}, '1', '1', '1'$$

The control was implemented using a state machine supported by an 11-bit counter to generate the necessary control (key loading, clocking and output latching) and handshaking signals. Loading the key-IV-padding word takes 288 cycles followed by 1152 cycles (4x288) of key mixing with the output suppressed. After initialisation one bit of keystream is output every cycle.

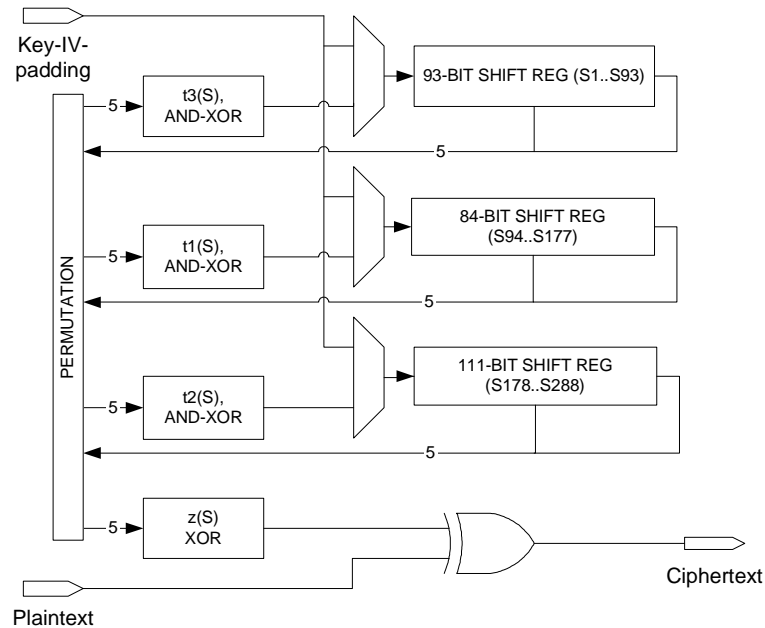


Fig. 9. Block diagram of Trivium

The design is very small and offers little scope for optimisation other than the usual logic and gate level manipulation which most synthesis tools perform automatically.

Table 5. Implementation results for Trivium

Design	Details	FPGA results	Gate level analysis (for Xilinx)
Trivium-1	sr: FD 29 SRL 21 (or FD 288) funcs(t1,t2,t3,z): FG 27 ctrl: FG 36 FD 19 all: FG 63 FD 48 SRL 21 (or FG 63 FD 307)	Xilinx (ISE): device: XC2S15-5 clock: 102 MHz bits/cycle: 1 slices: 40 Altera (Quartus II): device: EP1C3T144C7 clock: 249 MHz area: 327 LE t'put: 249 Mbps	throughput: 102 Mbps flip flop gates: 2456 other gates : 378 total FPGA gates: 2834
Trivium-n	sr: FD 288 funcs: FG 25+2n ctrl: FG 36 FD 19 all: FG 61+2n FD 307	Xilinx (ISE): device: Spartan 2 clock: 102 MHz bits/cycle: n ($n_{max} = 64$)	Estimate for parallel generation throughput: 102n Mbps for n=64: 6528 Mbps total FPGA gates: 2822+12n for n=64: 3590

As shown in [4] it is possible to use parallel computation to enhance throughput without increasing the flip-flop count (up to x64). This will improve the throughput versus area metric but the overall area will be increased.

A.3 Grain

The grain submission [5] is a key stream generator comprising two 80-bit shift registers and three combinatorial functions, $f(x)$, $g(x)$ and $h(x)$. The first, $f(x)$ is a 7th degree linear feedback polynomial for the first shift register. The second, $g(x)$ is a non linear feedback polynomial utilising 11 taps of the second shift register with a maximum of 6 taps being ANDed together. The final nonlinear function, $h(x)$ combines a total of 6 taps, here $h(x)$ defined to include the XOR with the final output of shift register N, is used to create the keystream.

$$\begin{aligned}
f(x) &= x^0 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80} \\
g(x) &= x^0 + x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + x^{80} \\
&\quad + x^{17} \cdot x^{20} + x^{43} \cdot x^{47} + x^{65} \cdot x^{71} \\
&\quad + x^{20} \cdot x^{28} + x^{35} + x^{47} \cdot x^{52} \cdot x^{59} \\
&\quad + x^{17} \cdot x^{35} \cdot x^{52} \cdot x^{71} + x^{20} \cdot x^{28} \cdot x^{43} \cdot x^{47} + x^{17} \cdot x^{20} \cdot x^{59} \cdot x^{65} \\
&\quad + x^{17} \cdot x^{20} \cdot x^{28} \cdot x^{35} \cdot x^{43} + x^{47} \cdot x^{52} \cdot x^{59} \cdot x^{65} \cdot x^{71} \\
&\quad + x^{28} \cdot x^{35} \cdot x^{43} \cdot x^{47} \cdot x^{52} \cdot x^{59} \\
h(x) &= N^0 + L^{55} + N^{17} + L^{77} \cdot L^{16} + L^{34} \cdot L^{16} + L^{16} \cdot N^{17} \\
&\quad + L^{77} \cdot L^{55} \cdot L^{34} + L^{77} \cdot L^{34} \cdot L^{16} + L^{77} \cdot L^{34} \cdot N^{17} + L^{55} \cdot L^{34} \cdot N^{17} + L^{34} \cdot L^{16} \cdot N^{17}
\end{aligned}$$

For initialisation the shift registers are loaded with key and IV (padded with ones to 80 bits). Initial key-IV mixing is then carried out for 160 cycles with the “keystream output bit” being fed back to both shift registers (using XOR).

Control was implemented using a finite state machine supported by an 8-bit counter. The overall design may be summarised by the following diagram.

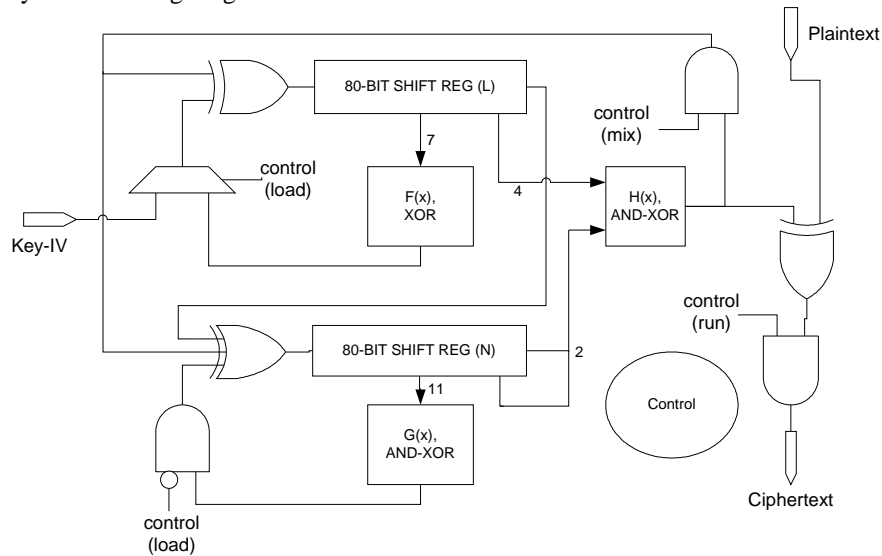


Fig. 10. Block diagram of Grain

The design is relatively simple and offers little scope for optimisation above the usual logic/gate-level optimisations that modern synthesis tools will automatically perform. The implementation had synchronous serial interfaces for cipher/plain text I/O and a separate serial input for loading key-IV.

Table 6. Implementation results for Grain

Design	Details	FPGA results	Gate level analysis (for Xilinx)
Grain-1	sr: FD 22 SRL 19 funcs(f,g,h): FG 26 ctrl: FG 29 FD 13 all: FG 55 FD 35 SRL 19 (or FG 55 FD 173)	Xilinx (ISE): device: XC2S15-5 clock: 105 MHz bits/cycle: 1 slices: 48 Altera (Quartus II): device: EP1C3T144C7 clock: 235 MHz area: 191 LE t'put: 235 Mbps	throughput: 105 Mbps flip flop gates: 1384 other gates : 330 total FPGA gates: 1714
Grain-n	sr: FD 160 funcs: FG 16+10n ctrl: FG 29 FD 13 all: FG 45+10n FD 173	Xilinx (ISE): device: Spartan 2 clock: 105 MHz bits/cycle: n (n _{max} = 16)	Estimate for parallel generation throughput: 105n Mbps for n=16: 1680 Mbps total FPGA gates: 1550+60n for n=16: 2510

The original paper on the design [5] described how the feedback functions can be paralleled (up to x16) to improve the throughput-area metric however this is at the expense of additional area.

A.4 Mosquito

The Mosquito self-synchronising stream cipher [6] is based around a non-linear shift register followed by a combinatorial function which yields a single bit of the keystream. The “conditional complementing shift register”, CCSR, connects each storage element with a small logic function derived from the proceeding element together with a key bit, K, and two further proceeding bits of the CCSR. Here, this is referred to as stage 0 and is defined by

$$G_i^{<0>} = G_{i-1}^{<0>} + K_{i-1} + G_v^{<0>} \cdot (G_w^{<0>} + 1) + 1, \quad 0 \leq i < 128$$

where v and w are both functions of the bit index i. These values are defined in table 1 of the Mosquito specification [6].

This equation essentially expresses, the elemental non-linear logic function (2xXOR, 1xNAND) used for all the logic stages. It has a convenient form in terms of FPGA implementation in that it is a 4-input 1-output function thus is described by a single LUT.

This function is repeated for seven combinational logic stages to produce the keystream bit, z, as described in table 7.

Table 7. Mosquito logic stages

Stage	Equation
1	$G_{4i \bmod 53}^{<1>} = G_{128-i}^{<0>} + G_{i+18}^{<0>} + G_{113-i}^{<0>} \cdot (G_{i+1}^{<0>} + 1) + 1, \quad 0 \leq i < 53$
2 to 5	$G_{4i \bmod 53}^{<j>} = G_i^{<j-1>} + G_{i+3}^{<j-1>} + G_{i+1}^{<j-1>} \cdot (G_{i+2}^{<j-1>} + 1) + 1, \quad 0 \leq i < 53$
6	$G_i^{<6>} = G_{4i}^{<5>} + G_{4i+3}^{<5>} + G_{4i+1}^{<5>} \cdot (G_{4i+2}^{<5>} + 1) + 1, \quad 0 \leq i < 12$
7	$G_i^{<7>} = G_{4i}^{<6>} + G_{4i+1}^{<6>} + G_{4i+2}^{<6>} + G_{4i+3}^{<6>}, \quad 0 \leq i < 3$
output	$z = G_0^{<7>} + G_1^{<7>} + G_2^{<7>}$

In this implementation, the resources for stages 2-5 share a single round based implementation, saving of 212 LUTs, at the cost of a 53-bit register and a 53-bit two-way multiplexer (53 LUTs and 53 DFFs). This is an equivalent saving of 530 gates at the cost of a factor of five reduction in throughput.

Once the 80-bit key has been entered serially together with the 128-bit IV (loaded into the CCSR), the initial key mixing of 105 iterations (each of 5 clock cycles) is performed with the plaintext input and ciphertext output zeroed. Subsequently, a new bit of keystream is available once in every 5 clock cycles. The control was implemented using a state machine supported by a 7-bit counter.

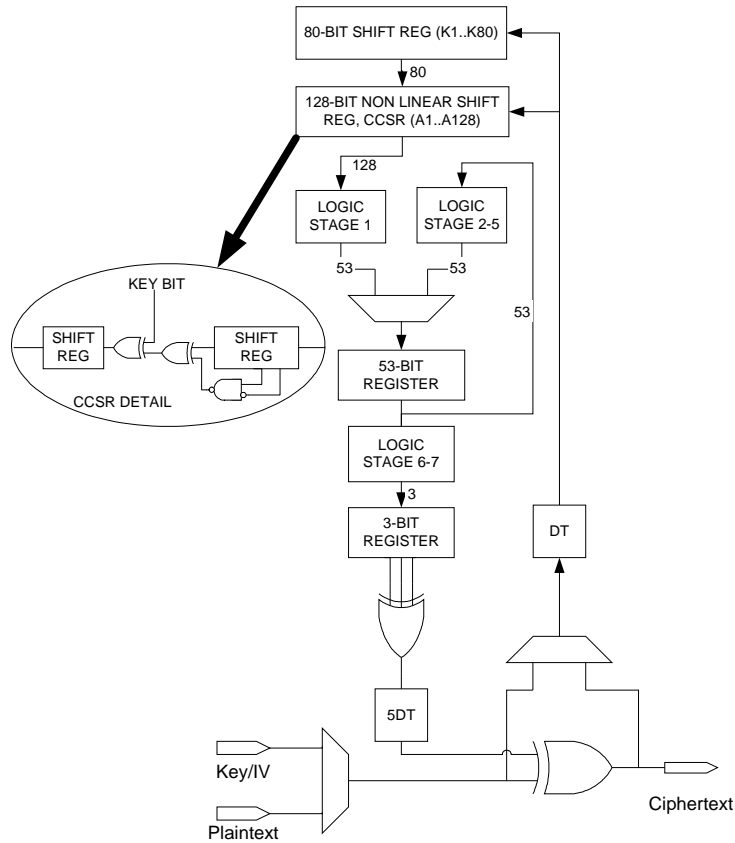


Fig. 11. Mosquito

The implementation results are tabulated below (Table 8), firstly for a repetition of the developers design (Mosquito-A) followed by the above resource shared architecture (Mosquito-B).

In these designs the key and IV were loaded serially, a tweak to the definition for initialisation would permit direct loading of the IV into the CCSR accepting the complementing due to the key. This would simplify the CCSR design avoiding needed the larger flip-flops with a reset capability.

Table 8. Implementation results for Mosquito

Design	Details	FPGA results	Gate level analysis (for Xilinx)
Mosquito-A	Design as per developers paper all: FG 450 FD 518	Xilinx (ISE): device: XC2S30-5 clock: 137 MHz bits/cycle: 1 slices: 298 Altera (Quartus II): device: EP1C3T144C7 clock: 280 MHz area: 530 LE t'put: 280 Mbps	throughput: 137 Mbps flip flop gates : 4144 other gates: 2700 total FPGA gates: 6844
Mosquito-B	keyreg: FD 80 ccsr: FG 130 FD 128 stages: FG 122 FD 56 ctrl: FG 27 FD 23 other: FG 4 FD 23 all: FG 283 FD 305 SRL 1 (or FG 283 FD 310)	Xilinx (ISE): device: XC2S15-5 clock: 110 MHz bits/cycle: 1/5 slices: 190 Altera (Quartus II): device: EP1C3T144C7 clock: 254 MHz area: 431 LE t'put: 50 Mbps	throughput: 22 Mbps flip flop gates: 2480 other gates : 1698 total FPGA gates: 4178

A.5 Phelix

Phelix [7] consists of five strands each of 32-bit data which are twisted together using shifting and arithmetic operations to form a helix like structure (hence its name). The cipher is supplied with a 256 bit key and 128-bit IV (nonce). Its operation could be viewed as a Feistel block cipher operating in a hybrid counter - cipher feedback mode to provide a keystream. However, as pointed out by the developers, when the MAC is not used then there exists a low complexity differential cryptanalysis against a CFB based decryptor. To avoid this, the “plaintext” applied to Quarter Round A should always be zero making the keystream generation a hybrid OFB-CTR mode. The datapath may be decomposed into a simple operator consisting of a programmable shift and 32-bit add/xor operation, thus a 32-bit processor style architecture could be considered as 20 rounds of this simple operator plus a key schedule computed using the same datapath. Additionally, Phelix supports a message authentication code which was not considered in these hardware results.

First, initial consideration is given to folding the round by a factor of two and four to exploit symmetry within the datapath, forming 160-bit half round and quarter round implementations respectively.

Folding in half is relatively straight forward and gains the expected approximate factor of two reduction in area from the unrolled baseline design. However, if a second fold is made to use a flexible quarter round function then the multiplexers required to select between hardwired shifts would negate the advantage. Thus the flexible quarter-round design has not been progressed further.

The half round function based design is approximately half the size of the unrolled design and would produce approximately half the throughput. However, the area required would be best described as “moderate resource” rather than low resource when compared with the AES but its expected throughput is much higher (due to its simple operations and wide datapath) than the other candidates.

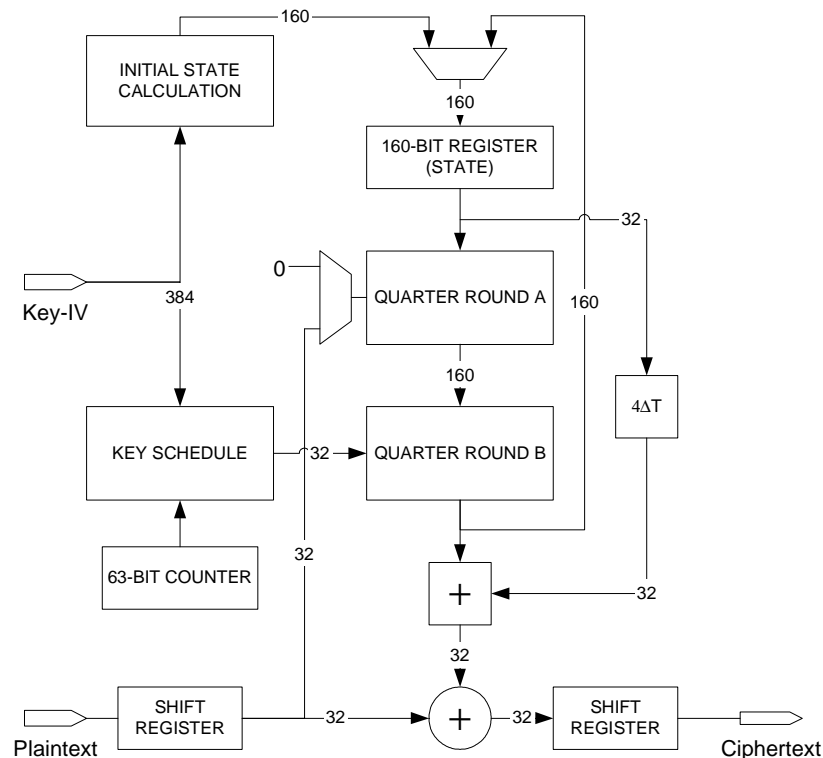


Fig. 12. Phelix half-round implementation

The next option was to consider a processor style architecture with a 32-bit datapath and controlled by state machines. The datapath consisted of a small register file (32x32-bit), a sequential shifter and configurable xor/add operation. The “height” in slices of the smaller FPGAs made it difficult to implement a fast ripple carry adder thus this was implemented as four separate 8-bit adders with registered carry propagation.

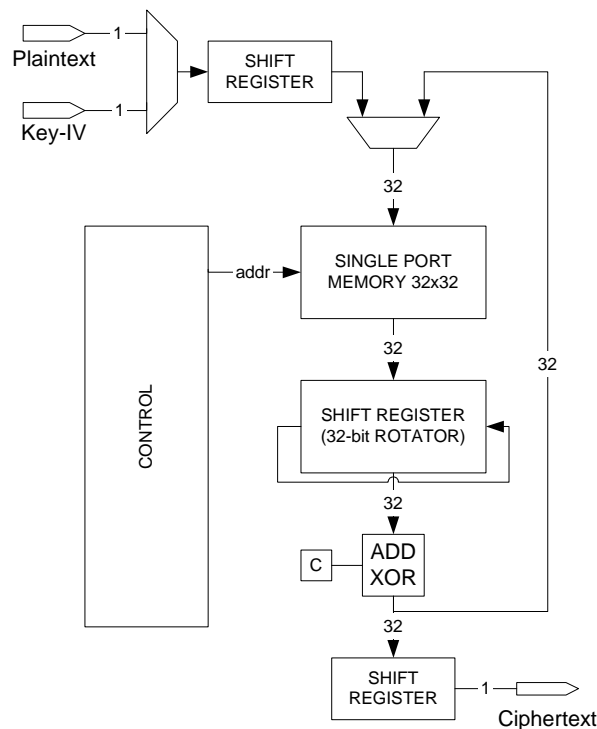


Fig. 13. Phelix datapath architecture

The register file consisted of 32 locations and contained the following registers:

- Eight registers to hold the padded key;
- Eight registers to hold the expanded nonce/IV;
- Five registers for the current state (Z_0, Z_1, Z_2, Z_3, Z_4);
- Four registers to keep the previous four states of Z_4 ;
- Two registers to hold a 64-bit counter value;
- Two temporary storage locations;
- and three constants (zero, one and key length).

A 22-bit instruction word was defined to control the datapath and instructions supplied from a set of subroutines contained within a 64x22-bit ROM (implemented using random-logic) and controlled by a second state machine. In order to allow for a constant rate of data bits in and out a third state machine together with shift registers was used to control the input and output.

The main operations required for the “helix” structure are straight forward to implement however, the key schedule, although carried out using the same datapath, is more difficult and dominates the area for the controller. There may be some scope for “tweaks” to be considered for the initialisation of the key schedule to simplify its hardware implementation. One option would be a change to the key schedule so that a simple 64-bit counter could be loaded with the IV then simply incremented and incorporated as part of the key schedule together with some simplification of setting the initial “ $Z(-8)$ ” state.

It should be stressed that these are initial results and some further optimisation may be possible. The ciphers area is dominated by the 512 bits required to store the expanded key and nonce. If some tweaks were permitted then the nonce could be loaded into the counter saving 256 bits of memory!

Considering an ASIC implementation, the constants and any constant bit (eg in key-length) may be hard wired further reducing the flip-flop count by an additional 93 bits. In total the saving, in flip-flops alone is estimated to be 2,792 equiv. gates. Further, if 32-bit I/O was acceptable then a further 710 gates could be saved. This results in a final estimate for a tweaked “Phelix” datapath based implementation of 8,800 gates. However, the 160-bit half round function only requires 7128 gates to implement which has simpler control and would be two orders of magnitude faster. However, it is the implementation of the keyschedule which is problematic and requires a number of 32-bit multiplexers and 32-bit binary adders together with two, partly overlapping 32-bit counters. The authors of this paper would urge the developers of Phelix to consider a revised keyschedule making more use of XOR rather than binary addition and being defined such that can be operated using a counter initially loaded with the nonce.

Table 9. Implementation results for Phelix

Design	Details	FPGA results	Gate level analysis (for Xilinx)
Phelix-A	160-bit Whole round design datapath: FG 1282 FD 320 keysched: FG 972 FD 540 all: FG 2254 FD 860	Xilinx (ISE): device: XC2S100-5 clock: 30 MHz bits/cycle: 32 slices: 1198 Altera (Quartus II): device: EP1C3T144C7 clock: 45 MHz area: 1772 LE t'put: 1440 Mbps	throughput: 960 Mbps flip flop gates: 6880 other gates: 13524 total FPGA gates: 20404
Phelix-B	160-bit Half round design helicies: FG 544 mux/reg: FG 260 FD 197 MEM 4x32 keysched: FG 1036 FD 555 all: FG 1840 FD 752 MEM 4x32 (or FG 1840 FD 880)	Xilinx (ISE): device: XC2S100-5 clock: 47 MHz bits/cycle: 32/2 slices: 1077 Altera (Quartus II): device: EP1C3T144C7 clock: 82 MHz area: 1455 LE t'put: 1312 Mbps	throughput : 750 Mbps flip flop gates: 7040 other gates: 11040 total FPGA gates: 18080
Phelix-C	32-bit Datapath Architecture datapath: FG 128 FD 68 regfile: MEM32x32 ctrl: FG 240 FD 81 I/O: FG 33 FD 64 all: FG 403 FD 213 MEM32x32 (or FG 403 FD 1237)	Xilinx (ISE): device: XC2S30-5 clock: 30 MHz bits/cycle: 32 / 294 slices: 264 Altera (Quartus II): device: EP1C3T144C7 clock: 58 MHz area: 1697 LE t'put: 6.31 Mbps	throughput: 3.26 Mbps flip flop gates: 9896 other gates: 2418 total FPGA gates: 12314
Phelix-D (Tweak)	non-compliant estimate allowing for some tweaks datapath: FG 128 FD 68 regfile: FD 675 ctrl: FG 240 FD 81 all: FG 368 FD 824	Xilinx (ISE): device: Spartan 2 clock: 30 MHz bits/cycle: 32 / 192 slices: 250 ESTIMATES	throughput: ~ 5 Mbps flip flop gates: 6592 other gates: 2208 total FPGA gates: 8800 ESTIMATES

In summary, Phelix, as currently defined, is difficult to implement efficiently in a rolled-up architecture however, in its half-round form performs with high throughput so may be worthy of further consideration.

A.6 Sfinks

The Sfinks [8] cipher comprises a 256-bit shift register together with a 16-bit multiplicative inversion in $GF 2^{16}$. This inversion derives its 16-bit input from a set of taps within the shift register. A single bit of its output is combined with a further bit from the shift register is used to generate the keystream. However, all 16-bits are utilised in a permuted order during the initial key mixing process. Thus the inversion is used in its complete form to create a “strong” SPN network for key-IV mixing and as the reduction operation for generation of the keystream bits. The paper [8] included message authentication code, however, in order to be consistent with the pure stream cipher model adopted in this paper it was omitted.

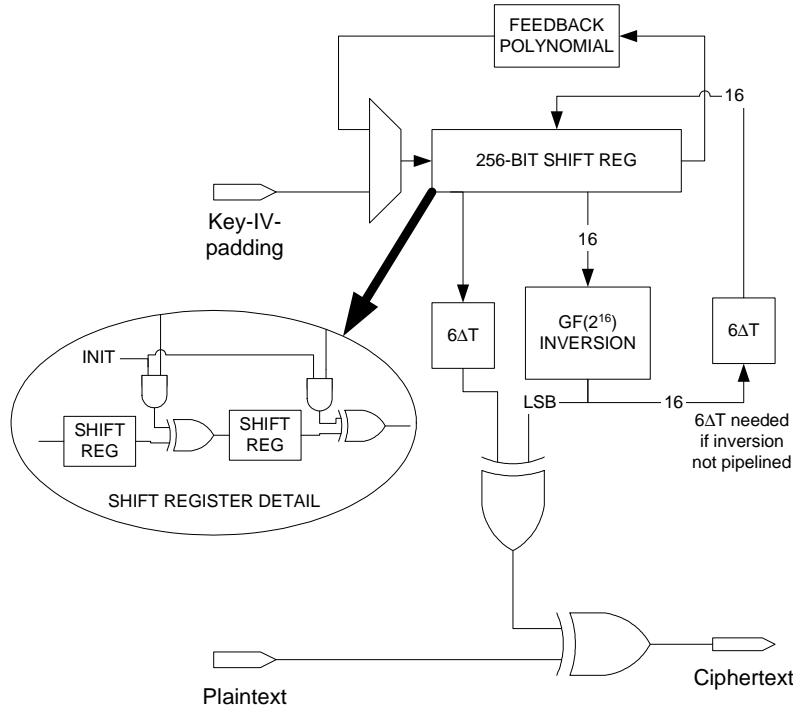


Fig. 14. Sfinks architecture

As shown by the AES, such multiplicative inverses can be efficiently implemented using composite field arithmetic thus the inverse was computed in the 16-bit $GF(((2^2)^2)^2)$ field with suitable isomorphisms. There are a number of opportunities for sharing resources within this inverse to reduce area at the expense of throughput. In the paper [8] the inversion was pipelined to enhance throughput, here the opposite approach is taken in that the 8-bit $GF(((2^2)^2)^2)$ multipliers and the 4-bit $GF(2^2)$ in the $GF(((2^2)^2)^2)$ inversion are resource shared using appropriate multiplexing and registers. Thus the inversion takes 5 cycles to complete.

The field construction was as follows:

Table 10. Sfinks composite field construction

Field	Polynomial	Binary representation
$GF(2)$	n/a	b_0
$GF(2^2)$	$P_u(u) = u^2 + u + 1$	$b_1u + b_0$
$GF((2^2)^2)$	$P_z(z) = z^2 + z + u$	$z(b_3u + b_2) + b_1u + b_0$
$GF(((2^2)^2)^2)$	$P_y(y) = y^2 + y + (uz + z)$	$y(b_7zu + b_6z + b_5u + b_4) + b_3zu + b_2z + b_1u + b_0$
$GF((((2^2)^2)^2)^2)$	$P_x(x) = x^2 + x + uzy$	$x(b_{15}yzu + b_{14}yz + b_{13}yu + b_{12}y + b_{11}zu + b_{10}z + b_9u + b_8) + b_7yzu + b_6yz + b_5yu + b_4y + b_3zu + b_2z + b_1u + b_0$

$$x_{GF2^{16}}^{-1}(v) \equiv \delta^{-1}(x_{GF2^{2.2.2.2}}^{-1}(\delta(v)))$$

and the isomorphisms between $GF(2^{16})$ and $GF(((2^2)^2)^2)$ may be represented in hexadecimal form as:

$$\delta(x) = 0001.x_0 + 7C91.x_1 + 4604.x_2 + 43DA.x_3 + 6C13.x_4 + 7E9D.x_5 + 6B49.x_6 + 1190.x_7 + 5A36.x_8 + 707F.x_9 + 454F.x_{10} + B430.x_{11} + 5EFD.x_{12} + D6CA.x_{13} + 104D.x_{14} + 6A24.x_{15}$$

$$\delta^{-1}(x) = 0001.x_0 + ACCB.x_1 + 90C4.x_2 + 86FA.x_3 + C583.x_4 + AE57.x_5 + 7C62.x_6 + 8684.x_7 + 444A.x_8 + 161C.x_9 + C1D6.x_{10} + 2D90.x_{11} + 2A5D.x_{12} + C215.x_{13} + 470A.x_{14} + 4A4A.x_{15}$$

The resource sharing allows the reduction in gate count for the “operational” phase of the cipher but is somewhat frustrated by the initial key mixing stage. The algorithm requires that all 16-bits of the inversion to be fed back with a delay of 6-clocks which matches the developers’ pipelined inversion. A resource-shared or non-pipelined version of the inversion would require an additional 96 flip-flops to implement the required delay to match the algorithm definition for initialisation. This effectively overcomes any advantage in area from using resource sharing in the inversion and mandates a 6-stage pipelined inversion. This may be one area where a “tweak” could be considered to permit more flexibility in terms of implementation (lower area or higher throughput).

The “Sfinks-A” design is essentially follows the developers intended architecture. Sfinks-B is a compliant design with resource sharing as described above. Finally, Sfinks-C shows the area saving if the design key mixing was changed to avoid the need to delay the inverse.

Table 11. Implementation results for Sfinks

Design	Details	FPGA results	Gate level analysis (for Xilinx)
Sfinks-A	FPGA results for pipelined design lfsr: FG 22 FD 262 inv: FG 289 FD 92 SRL 16 ctrl: FG 41 FD 18 SRL 1 all: FG 352 FD 372 SRL 17 (or FG 352 FD 474)	Xilinx (ISE): device: XC2S30-5 clock: 118 MHz bits/cycle: 1 slices: 334 Altera (Quartus II): device: EP1C3T144C7 clock: 207 MHz area: 556 LE t’put: 207 Mbps	throughput: 118 Mbps flip flop gates: 3792 other gates: 2112 total FPGA gates: 5904
Sfinks-B	compliant with SFINKS paper lfsr: FG 22 FD 262 inv: FG 177 FD 27 feedback: SRL 17 ctrl: FG 42 FD 42 all: FG 241 FD 331 SRL 17 (or FG241 FD 433)	Xilinx (ISE): device: XC2S30-5 clock: 37 MHz bits/cycle: 1/5 slices: 334 Altera (Quartus II): device: EP1C3T144C7 clock: 60 MHz area: 517 LE t’put: 12.0 Mbps	throughput: 7.4 Mbps flip flop gates: 3464 other gates: 1446 total FPGA gates: 4910
Sfinks-C (tweak)	initialisation “tweaked” lfsr: FG 22 FD 262 inv: FG177 FD 27 ctrl: FG 40 FD 25 all: FG 239 FD 314	Xilinx (ISE): device: XC2S30-5 clock: 37 MHz bits/cycle: 1/5 slices: 319 Altera (Quartus II): device: EP1C3T144C7 clock: 73 MHz area: 508 LE t’put: 14.6 Mbps	throughput: 7.4 Mbps flip flop gates: 2512 other gates: 1434 total FPGA gates: 3946

A.7 Hermes-8

The Hermes-8 [9] has been designed around an 8-bit SPN architecture. The choice for the substitution operation was the well known 8-bit AES S-box. The permutation was carried out at the byte level (rather than the more usual bit level) by selecting differing indexes into the state and key registers.

The algorithm requires modulo arithmetic in order to carry out the indexed addressing (modulo-7, 10 and 23). In the running phase this can be accomplished by simply incrementing specific modulo counters which automatically reset when the correct modulus is reached. However, for initialisation these counters must be loaded with a modulo-value derived from the XOR of a number of key-bytes. The low resource implementation of this is to use conditional subtraction by the required modulus either for a fixed number of iterations or terminate when no further modulo reduction is required. The latter would leak significant side channel information during initialisation so would be most undesirable. Looking for an alternative for initialising the modulo counters would be a good starting point for a “tweak” to simplify hardware implementation.

The controller is split into two state machines, one specifically to control the datapath given an instruction word and the second to carry out the global control and generate the required sequence of instruction words. A

single port memory was used for the register file containing both key and state values (36x8 bits total). The controller also contained a number of counters: two off mod-23, mod-10 and mod-7. The values of these counters were used to provide all the necessary addresses for indexing into the register file.

The datapath consists of an 8-bit XOR operation, the AES S-box implemented using composite field arithmetic in $GF((2^2)^2)$ with resource sharing of the 4-bit $GF((2^2)^2)$ multiplier and a dedicated unit for performing modulo reduction (conditional subtraction of modulus) of an 8-bit value and is only used in the initialisation phase.

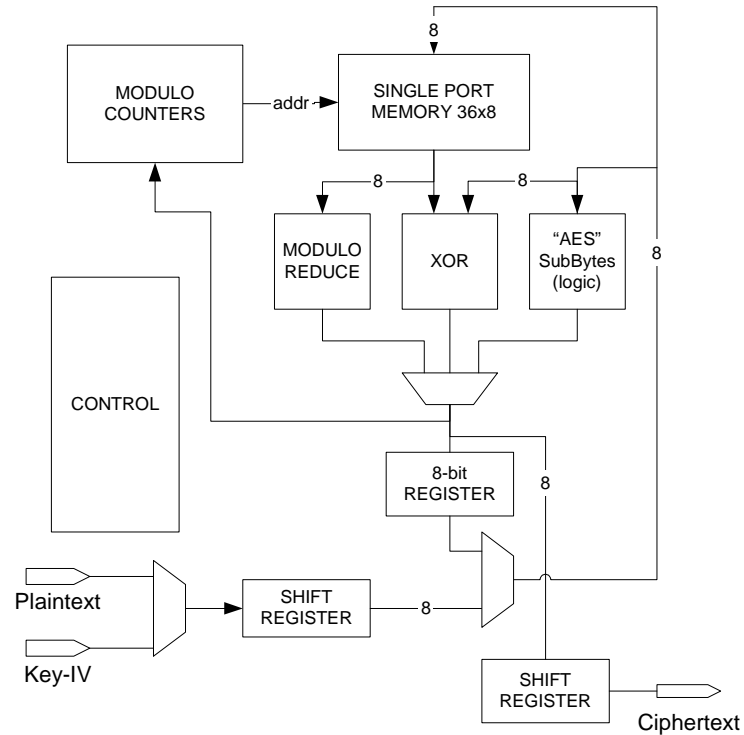


Fig. 15. Hermes-8 architecture

Table 12. Implementation results for Hermes-8

Design	Details	FPGA results	Gate level analysis (for Xilinx)
Hermes8	regfile: RAM32x16 datapath-less-sbox: FG 63 FD 8 sbox: FG 52 FD 21 ctrl: FG 148 FD 101 other: FG 14 FD 2 all: FG 277 FD 132 RAM32x16 (or FG 277 FD 420)	Xilinx (ISE): device: XC2S30-5 clock: 45 MHz bits/cycle: 64 / 512 slices: 190 Altera (Quartus II): device: EP1C3T144C7 clock: 61 MHz area: 645 LE t'put: 7.6 Mbps	throughput: 5.6 Mbps flip flop gates: 3360 other gates: 1662 total FPGA gates: 5022