Decipher:
Architecture Development of Reconfigurable Encryption Hardware

Kenneth Eguro and Scott Hauck
{eguro, hauck}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500

UWEE Technical Report
Number UWEETR-2002-0012
10/15/02

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: http://www.ee.washington.edu

Decipher: Architecture Development of Reconfigurable Encryption Hardware

Kenneth Eguro and Scott Hauck
{eguro, hauck}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500

University of Washington, Dept. of EE, UWEETR-2002-0012
October 2002

Abstract

Domain-specific FPGAs attempt to improve performance over general-purpose reconfigurable logic by providing only the necessary flexibility needed for a range of applications. One typical optimization is the replacement of more universal fine-grain logic elements with a specialized set of coarse-grain functional units. While this improves computation speed and reduces routing complexity, this also introduces a unique problem. It is not clear how to simultaneously consider all applications in a domain and determine the most appropriate overall number and ratio of different functional units. In this paper we use the candidate algorithms of the Advanced Encryption Standard competition to explore this problem. We introduce three algorithms that attempt to balance the hardware needs of the domain and optimize the overall performance and area requirements for an encryption-specialized reconfigurable array.

1 Introduction

The Advanced Encryption Standard competition offered a compelling opportunity for designers to exploit the benefits of domain-specific FPGAs to produce a versatile and early-to-market encryption device. Figure 1 provides a brief timeline for the competition. The competition requirements and the candidate algorithms have several unique characteristics that make designing specialized, coarse-grain reconfigurable devices particularly attractive. First, high performance is very important due to today's large volume of sensitive electronic traffic. Second, flexibility, in addition to being helpful to allow for future updates, was a necessity for pioneering designers since the contest allowed submissions to be modified in order to address any security or performance concerns that might be raised during public review. Furthermore, the control logic and routing structure for the system does not need to be complex since the iterative dataflow for most of the algorithms conforms to one of three simple styles. Figure 2 shows the common dataflow types. Lastly, very few different types of functional units are needed because all of the required operations can be implemented with a combination of simple ALUs, multipliers, Galois Field multipliers, bit permutations, memories, and multiplexors.

<u>Advanced Encryption Standard Competition Timeline</u>

January 1997	The National Institute for Standards and Technology issues a public call for symmetric-key block cipher algorithms that are both faster and more secure than the aging Data Encryption Standard
August 1998	From around the world, twenty-six submissions are received. Fifteen algorithms are accepted to compete in an eight-month review period
August 1999	Based upon brief but careful public analysis and comment about security and efficiency, five algorithms are selected for further scrutiny.
October 2000	After a nine-month second review period and several public forums, Rijndael is announced as the new encryption standard
December 2001	The Secretary of Commerce makes the AES a Federal Information Processing Standard. This makes AES support compulsory for all federal government organizations as of May 2002

Figure 1 - *A brief timeline for the Advanced Encryption Standard competition sponsored by the National Institute for Standards and Technology. Note that modifications to four algorithms were submitted between August 1998 and August 1999.*

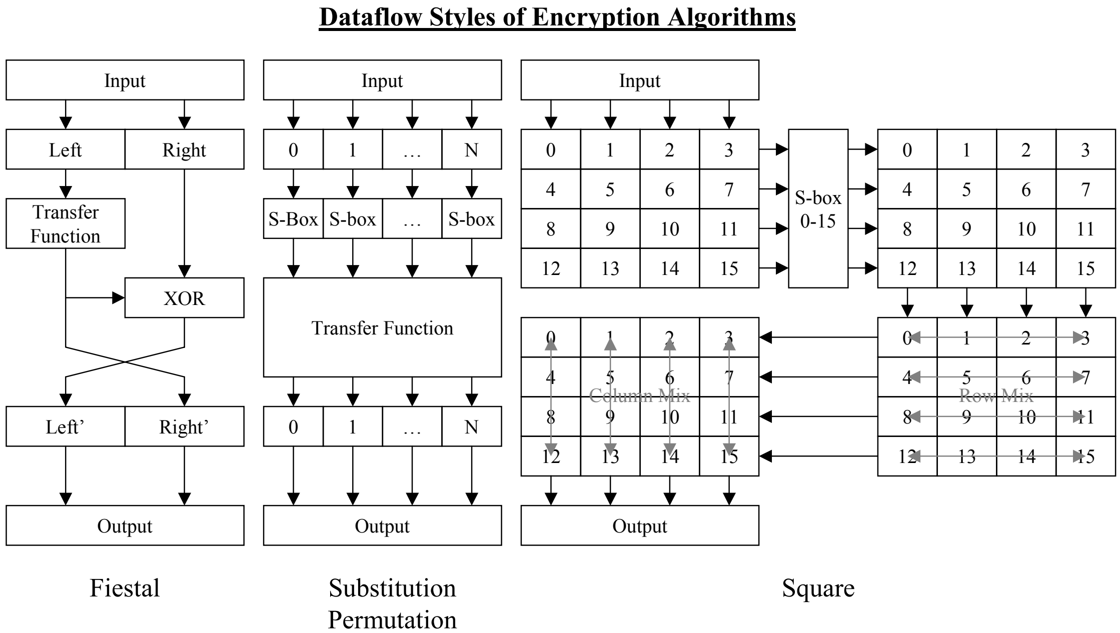

Figure 2 - *Block diagrams of common encryption styles. Thirteen of the fifteen AES candidate algorithms conform to one of these dataflow types.*

2 Implications of Domain-Specific Devices

Although domain-specific FPGAs can offer great speed improvements over general-purpose reconfigurable devices, they also present some unique challenges. One issue is that while design choices that affect the performance and flexibility of classical FPGAs are clearly defined and well understood, the effects that fundamental architecture decisions have on specialized reconfigurable devices are largely unknown and difficult to quantify. This problem is primarily due to the migration to coarse-grain logic resources. While the basic logic elements of general-purpose reconfigurable devices are generic and universally flexible, the limiting portions of many applications are complex functions that are difficult to efficiently implement using the fine-grain resources provided. These functions require many logic blocks and lose much of their performance in intra-function communication. By mapping these applications onto architectures that include more sophisticated and specialized coarse-grain functional units, they can be implemented in a smaller area with better performance. While the device may lose much of its generality, there are often common or related operations that reoccur across similar applications in a domain. These advantages lead to the integration of coarse-grain functional elements into specialized reconfigurable devices, as is done in architectures such as RaPiD[1]. However, the migration from a sea of fine-grained logical units to a clearly defined set of coarse-grained function units introduces a host of unexplored issues. Merely given a domain of applications, it is not obvious what the best set of functional units would be, much less what routing architecture would be appropriate, what implications this might have on necessary CAD tools, or how any of these factors might affect each other.

The first challenge, the selection of functional units, can be subdivided into three steps. First, all applications in a domain must be analyzed to determine what functions they require. Crucial parts such as wide multipliers or fast adders should be identified. Next, this preliminary set of functional units can be distilled to a smaller set by capitalizing on potential overlap or partial reuse of other types of units. Different sizes of memories, for example, can be combined through the use of multi-mode addressing schemes. Lastly, based upon design constraints, the exact number of each type of unit in the array should be determined. For example, if the applications are memory intensive rather than computationally intensive, the relative number of memory units versus ALUs should reflect this.

In this paper we will use the 15 candidate algorithms of the Advanced Encryption Standard competition to examine the difficulties of the functional unit selection process. Primarily, we will focus on the problem of determining the most appropriate quantity and ratio of functional units. While operator identification and optimization are also complex problems and unique to coarse-grain architectures, the algorithms themselves, at least in the encryption and DSP domains, provide an obvious starting point. The algorithms in these domains have a relatively small number of strongly typed functional units, so it is fairly simple to perform the logical optimization and technology mapping by hand.

Although this may overlook subtle optimizations, such as the incorporation of more sophisticated operators, this does provide an acceptable working set.

3 Functional Unit Design

We chose to build a platform similar to the RaPiD reconfigurable architecture [1]. While the necessary functional units are different, the architecture is particularly well suited for linear, iterative dataflow. However, since it is a coarse-grain architecture, there are particular differences that separate it from general-purpose reconfigurable devices. One major design decision is the bit-width of the architecture. Since the operations needed by the AES competition algorithms range from single-bit specific manipulations to wide 128-bit operations, we determined that a 32-bit word size provided a reasonable compromise between the awkwardness of wide operators and the loss of performance due to excessive intra-function routing. In addition, while the algorithms did not necessary preclude the use of 64, 16 or 8-bit processors, the natural operator width for many of the algorithms was specifically designed to take advantage of the more common 32-bit microprocessors. With the bit-width of the architecture defined, the next problem was to find a comprehensive set of operators. Analysis identified six primary operations required for the AES candidate algorithm. These operations lead to the development of seven distinct types of functional units. See Figure 3 for a list of operations and Figure 4 for a description of the functional unit types.

Required Operators of the AES Candidate Algorithms

Class	Operations
Multiplexor	Dynamic dataflow control
Rotation	Dynamic left rotation, static rotation, static logical left/right shift, dynamic left shift
Permutation	Static 32-bit permutation, static 64-bit permutation, static 128-bit permutation
RAM	4-bit lookup table, 6-bit lookup table, 8-bit lookup table
Multiplication	8-bit Galois Field multiplication, 8-bit integer multiplication, 32–bit integer multiplication
ALU	Addition, subtraction, XOR, AND, OR, NOT

Figure 3 – *Table of six operator classes used in the AES competition algorithms.*

One peculiarity of a RaPiD-like architecture is the distinct separation between control and datapath logic. Like the RaPiD architecture, we needed to explicitly include multiplexors in the datapath to provide support for dynamic dataflow control. In addition, due to the bus-based routing structure, we needed to include rotator/shifters and bit-wise crossbars to provide support for static rotations/shifts and bit permutations. Although these static operations would be essentially free to implement using the routing resources of a general-purpose FPGA, there is the benefit that the necessary dynamic rotations and shifts can be supported with minimal additional hardware. For future flexibility, we also chose to add in currently unused operations such as arithmetic shifting. We chose to implement a dynamically/statically controlled rotation/shift unit separately from a statically controlled crossbar for two reasons. First, static random bit permutations are needed far less than rotation or shift operations and we expect the crossbar to be larger than its rotation/shift counterpart. Second, the additional hardware required to make a crossbar emulate a dynamically controlled rotator/shifter is too large to be efficient.

Next we considered the logical and arithmetic needs of the algorithms. First, since all of the algorithms contain addition and subtraction or bit-wise logical operations, we chose to roll all of these functions into one ALU type. For simplicity and future flexibility, we chose to simply extend the 16-bit RaPiD ALU [1] to a 32-bit version. Second, many of the algorithms require either an 8 or 32-bit integer multiplication or a related function, an 8-bit Galois Field multiplication. See Appendix A for an explanation of Galois Field multiplication. Although these operations can be performed using the other functional units that we have included, the frequency and complex nature of these operations make them ideal candidates for dedicated functions units both in terms of area efficiency and speed. We chose to implement the 32-bit integer multiplier and the 4-way 8-bit integer/Galois field multiplier as two separate units for three main reasons. First, the AES algorithms do include multiplications up to 64 bits. To quickly calculate these multiplications, it was necessary to implement a wide multiplier. Second, as can be seen from the diagram in Appendix A, it is difficult to make an efficient multi-mode 32-bit integer/8-bit Galois field multiplier. Most likely, this unit would only be able to handle one or possibly two Galois multiplications at a time. This is not efficient in terms of resource utilization or speed. Lastly, if we do implement a four-way 8-bit Galois field multiplier, it is able to handle four 8-bit integer multiplications with minimal modification and little additional hardware.

Finally, we considered the memory resources that our architecture should provide. While one of the AES candidate algorithms requires a larger lookup table, most of the algorithms use either a 4 to 4, a 6 to 4 or an 8 to 8 lookup table.

Instead of separating these out into three distinct types of memory units, we chose to combine them into one memory that could support all three addressing modes. From this, we developed a 256-byte memory that either contained eight 4 to 4 lookup tables (each with 4 pages of memory), eight 6 to 4 lookup tables, or one 8 to 8 lookup table. See Figure 5 for an illustrated description of these addressing modes.

Functional Unit Description

Unit	Description
Multiplexor	32 x 2:1 muxes
Rotate/shift Unit	32-bit dynamic/static, left/right, rotate/(logical/arithmetic) shift
Permutation Unit	32 x 32:1 statically controlled muxes
RAM	256 byte memory with multi-mode read function • Mode 0: Single 256 byte memory (8-bit input, 8-bit output) • Mode 1: 8 x 64 nibble memories (8 x 6-bit inputs, 8 x 4-bit outputs)
32-bit Multiplier	32–bit integer multiplication (32 –bit input, 64-bit output)
8-bit Multiplier	4 x 8-bit modulus 256 integer multiplications or 4 x 8-bit Galois Field multiplications
ALU	Addition, subtraction, XOR, AND, OR, NOT

Figure 4 – *Table of the seven types of functional unit resources supported by our system.*

Multi-mode RAM unit

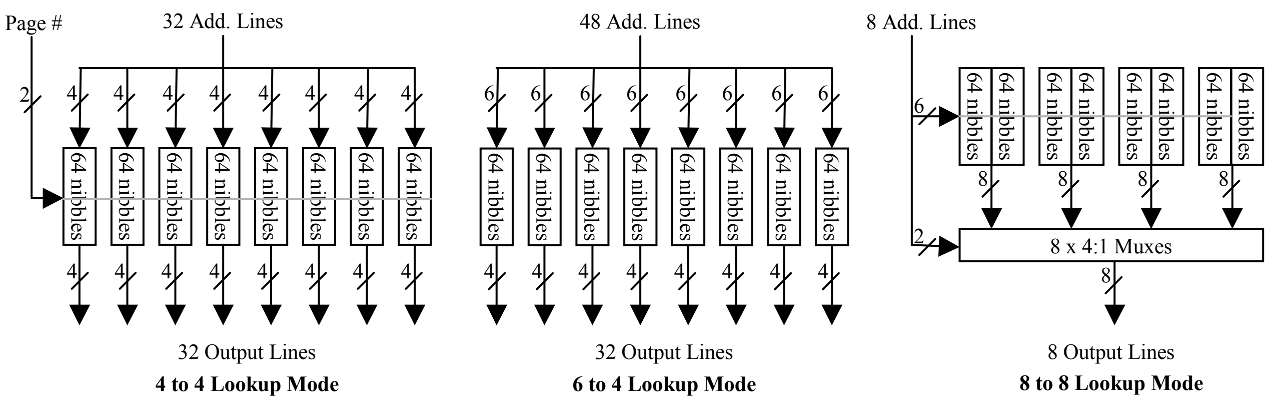

Figure 5 - *The three most common lookup table configurations for our RAM unit*

4 Functional Unit Selection

Although it is relatively straightforward to establish the absolute minimum area required to support a domain, determining the best way to allocate additional resources is more difficult. During the functional unit selection process for the AES candidate algorithms, we determined the necessary hardware to implement each of the algorithms in a range of performance levels. We identified the resource requirements for systems ranging from a completely unrolled algorithm to a single iteration of the main encryption loop or, in some cases, sub-loops as appropriate, attempting to target natural arrangements in between. From this data we discovered four factors that obscure the relationship between hardware resources and performance. First, although the algorithms in our domain share common operations, the ratio of the different functional units varies considerably between algorithms. Without any prioritization, it is unclear how to distribute resources. For example, if we consider the fully rolled implementations for six encryption algorithms, as in Figure 6, we can see the wide variation in RAM, crossbar, and runtime requirements among the different algorithms. To complicate matters, if we attempt to equalize any one requirement over the entire set, the variation among the other requirements becomes more extreme. The second factor that complicates correlation between hardware availability and performance is that the algorithms have vastly different complexities. This means that the hardware requirement for each algorithm to support a given throughput differs considerably. It is difficult to fairly quantify the performance-versus-hardware tradeoff of any domain that has a wide complexity gap. In Figure 7 we show an example of five different encryption algorithms that all have similar throughput, but have a wide variation in hardware requirements. The third problem of allocating hardware resources is that the requirements of the algorithms do not necessarily scale linearly or monotonically when loops are unrolled. This phenomenon makes it difficult to foresee the effect of decreasing the population of one type of functional unit and increasing another. See Figure 8 for an example of this non-uniform behavior. The last problem of estimating performance from available resources is that overextended functional units often can be supplemented by using combinations of other, underutilized units. For example, a regular bit permutation could be accomplished with a mixture of shifting and masking. Although this flexibility may improve resource utilization, it also dramatically increases the number of configurations to be evaluated.

Ratio Complications

Algorithm (Baseline)	RAM Blocks	XBar	Runtime
CAST-256 (1x)	16	0	48
DEAL (1x)	1	7	96
HPC (1x)	24	52	8
Loki97 (1x)	40	7	128
Serpent (1x)	8	32	32
Twofish (1x)	8	0	16
Average	**16.2**	**16.3**	**54.7**
Std. Dev.	**14.1**	**21.1**	**47.6**

Algorithm (Unrolling Factor)	RAM Blocks	XBar	Runtime
CAST-256 (2x)	32	0	24
DEAL (32x)	32	104	3
HPC (1x)	24	52	8
Loki97 (1x)	40	7	128
Serpent (8x)	32	32	4
Twofish (4x)	32	0	4
Average	**32**	**32.5**	**28.5**
Std. Dev.	**5.6**	**40.6**	**49.4**

Figure 6 – *Two examples of the complications caused by varying hardware demands. The table on the left compares the RAM, crossbar and runtime requirements for the baseline implementations of six encryption algorithms. The table on the right displays the compounded problems that occur when attempting to normalize the RAM requirements across algorithms.*

An effective solution must have the flexibility needed to simultaneously address the multi-dimensional hardware requirements of the entire domain while maximizing usability and maintaining hard or soft area and performance constraints. In the following sections we propose three solutions to the functional unit selection problem. The first addresses hard performance constraints. The second and third algorithms attempt to maximize the overall performance given a hard or soft area constraint.

Complexity Disparity

Algorithm (Unrolling Factor)	RAM Blocks	XBar	Runtime
CAST-256 (2x)	32	0	24
DEAL (4x)	4	16	24
Loki97 (4x)	160	7	32
Magenta (4x)	64	0	18
Twofish(1x)	8	0	16
Average	**53.6**	**4.6**	**22.8**
Std. Dev.	**64.1**	**7.1**	**6.3**

Figure 7 – *An illustration of the imbalance that occurs when attempting to equalize throughput across algorithms.*

Scaling Behavior

Algorithm (Unrolling Factor)	RAM Blocks	Mux	Runtime
FROG (1x)	8	23	512
FROG (4x)	8	72	128
FROG (16x)	8	256	32
FROG (64x)	16	120	8
FROG (256x)	64	30	2

Figure 8 - *An example of the unpredictable nature of hardware demands when unrolling algorithms.*

5 Performance-Constrained Algorithm

The first algorithm we developed uses a hard minimum throughput constraint to guide the functional unit selection. As described earlier, we began the exploration of the AES domain by establishing the hardware requirements of all of the algorithms for a variety of performance levels. These results are shown in Appendix B. First, we determined the hardware requirements for the most reasonably compact versions of each algorithm. For all algorithms except for Loki97, these fully rolled implementations require very modest hardware resources. Loki97 is unique because the algorithm requires a minimum of 10KB of memory. After this, we determined the hardware requirements for various unrolled versions of each algorithm at logical intervals. We use this table of results to determine the minimum hardware requirements for all algorithms to support a given throughput constraint.

We first determine the hardware requirements to run each algorithm at a given minimum throughput. We then examine these requirements to establish the maximum required number of each type of functional unit. To calculate the overall performance for this superset of resources, we re-examine each algorithm to determine if there are sufficient resources to allow for greater throughput, then sum the overall clock cycles required to run all of the algorithms. See Figure 9 for an example of this process. Note that this is a greedy algorithm and, due to the non-monotonic behavior of hardware requirements, does not necessarily find the minimum area or maximum performance for the system.

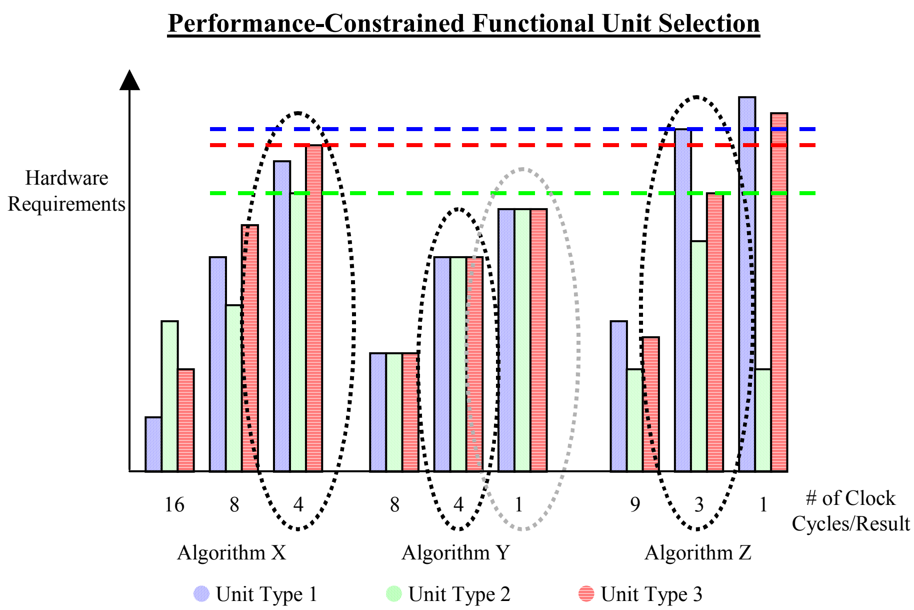

Figure 9 - *Illustration of our performance-constrained selection algorithm for a performance threshold of 4 clock cycles. The three horizontal dotted lines represent the minimum hardware requirements. Note that during the second Algorithm Y is able to unroll further.*

6 Area-Constrained Algorithm

The next two algorithms we developed use simulated annealing to provide more sophisticated solutions that attempt to capitalize on softer performance constraints to improve average throughput. The second algorithm begins by randomly adding components until limited by a given area constraint. Next we evaluate the quality of the configuration by determining the average maximum throughput across the domain given the existing resources. If an algorithm cannot be implemented on the available hardware, a penalty is incurred. Then we perturb the configuration by randomly picking two types of components, removing enough of the first type to replace it with at least one of the second, then adding enough of the second type to fill up the available area. Finally, the quality of the new configuration is evaluated in the same manner as before. If the new configuration provides the same or better average throughput, it is accepted. If it does not provide better performance, based on the current temperature and relative performance degradation, it may or may not be accepted. This process continues based on a simple cooling schedule. See Figure 10 for an illustration of this procedure. Note that, while for simplicity we did not directly deal with the possibility of functional unit emulation in this paper, there is no inherent limitation in either of the area-constrained solutions that would prevent this from being addressed with a larger hardware/throughput matrix

7 Improved Area-Constrained Algorithm

Our last algorithm attempts to balance performance and area constraints. First, we eliminate implementations from the hardware/throughput matrix that do not provide enough throughput to meet a specified minimum performance requirement. Then, we randomly select one of the remaining implementations of each algorithm for our current arrangement. We then determine the minimum hardware and area requirements necessary to fit all of the algorithms at their current settings. Afterwards, we establish if any algorithms can be expanded to a higher performance level given the calculated hardware resources. We calculate the new performance over all of the algorithms, then penalize for any excessive area requirements. We then perturb the configuration by randomly choosing one algorithm and changing the setting to a different performance level. Finally, the quality is re-evaluated and compared to the original arrangement in a similar manner as described earlier. See Figure 11 for an illustration of this process.

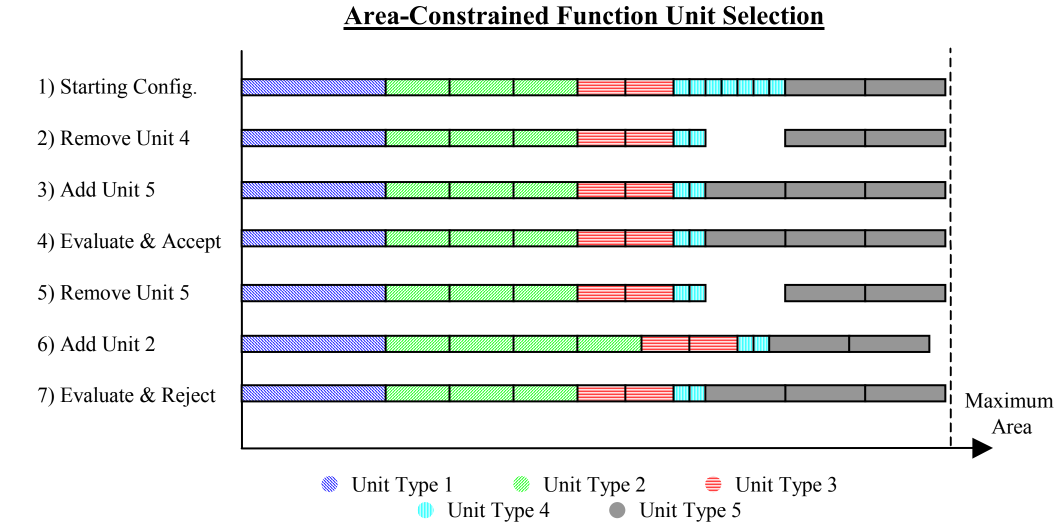

Figure 10 - *Illustration of our area-constrained selection algorithm.*

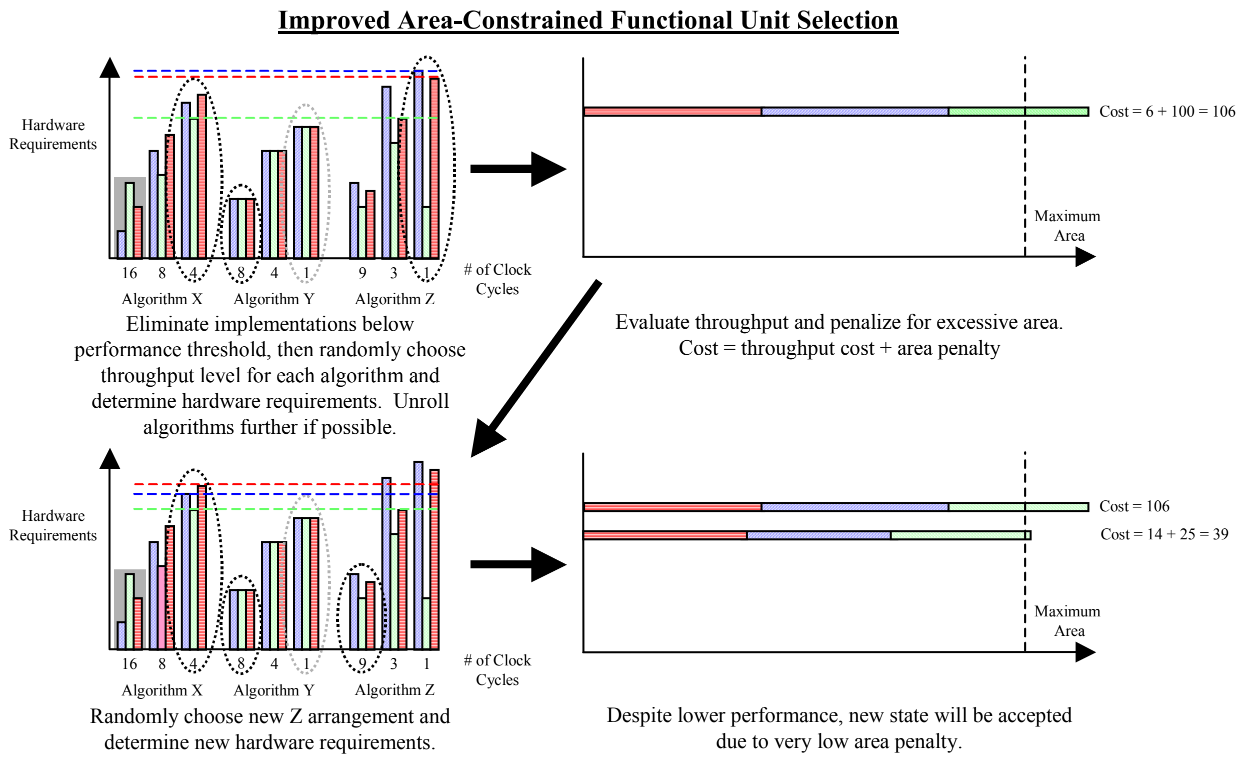

Figure 11 - *Illustration of our improved area-constrained selection algorithm assuming the throughput threshold is 10 cycles/block.*

8 Testing and Results

The testing of the functional unit selection techniques began by using the performance-constrained algorithm as a baseline for comparison. We first identified all of the distinct throughput levels between the AES candidate algorithms. As seen in Appendix B, these ranged between 1 and 512 cycles per data block. Then, each of these distinct throughput constraints was fed into the performance-constrained functional unit selection algorithm. The area requirements for each were recorded and then used as inputs to the two area-constrained techniques.

These three techniques produce very different results when applied to the set of AES candidate algorithms. The results of our testing can be seen in Appendix C. As expected, the hard constraints of the performance driven approach has limitations. As seen in Figures 12, the maximum time required by any algorithm is the lowest out of the three methods for most of the implementations. However, as seen in Figure 13, the overall performance of the system suffers by as much as almost 50% as compared to either of the area-driven techniques. If the design constraints allow for some flexibility in term of the minimum acceptable performance, a better compromise would be either of the area driven approaches. Although the results are very similar between these two techniques, the improved area-constrained method consistently produces better overall performance and, as seen in Figure 14, smaller area requirements.

<u>Minimum Throughput Results of Functional Unit Selection</u>

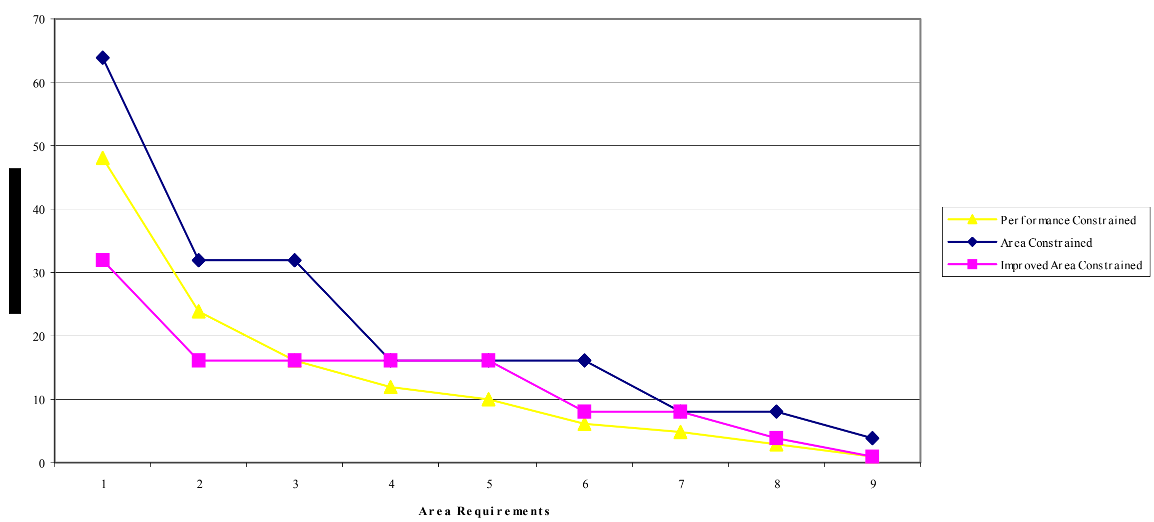

Figure 12 - *Graph of worst-case throughput. The area results are shown in Figure 14.*

<u>Overall Performance Results of Functional Unit Selection</u>

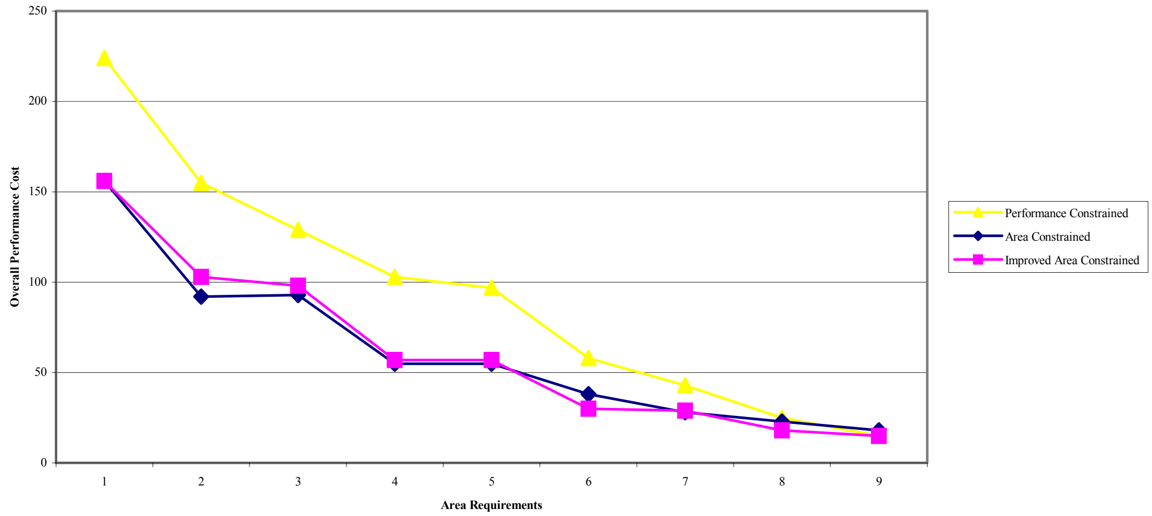

Figure 13 - *Graph of overall throughput cost. The area results are shown in Figure 14.*

<u>Area Results of Functional Unit Selection</u>

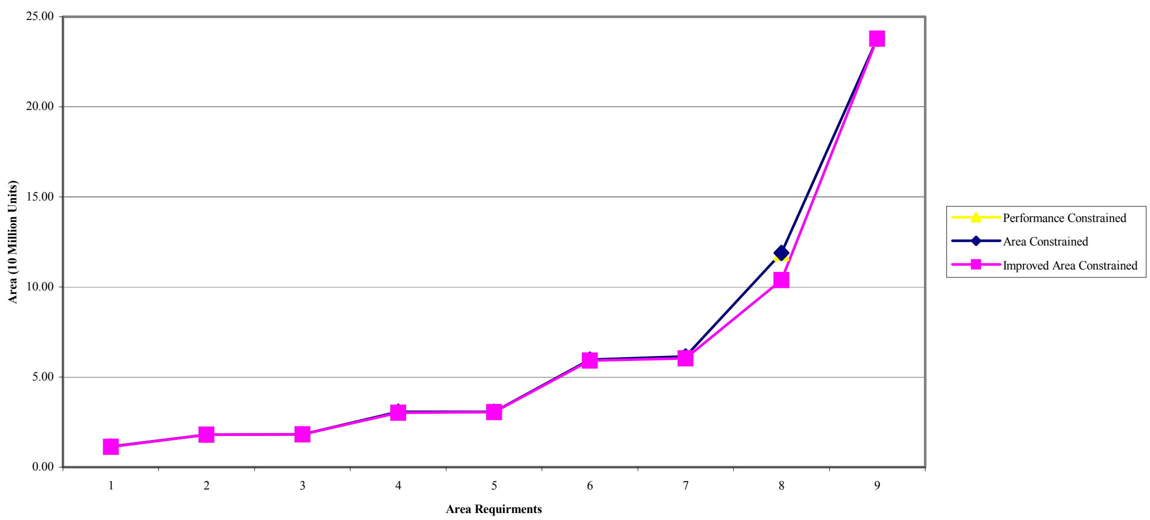

Figure 14 - *Graph of area requirements used to obtain the throughput results in Figures 12 and 13*

The three functional unit selection techniques also recommended very different hardware resources. As seen in Figure 15, the hard constraints of the performance driven method lead to a very memory dominated architecture. This is primarily caused by the quickly growing memory requirements of Loki97 and, eventually, MAGENTA. See Appendix B for the details of the hardware requirements for all of the encryption algorithms. While this additional memory may be necessary to allow for these algorithms to run a high speed, it does not adequately reflect the requirements of the other encryption algorithms. As seen in Figure 16, the original area driven technique has a fairly even response to varying area limitations. Since only three algorithms benefit from having more than 64KB of memory and only one or two benefit from large numbers of multiplexors, by devoting more resources to the other components the average throughput can be improved. As seen in Figure 17, the improved area-constrained technique combines these recommendations. Like the original area-constrained technique, it recognizes the limited usage of multiplexors. However, it also considers the moderate RAM requirements of many of the high performance implementations of the AES algorithms. This is reflected in the mild emphasis of RAM units in the medium to large area tests.

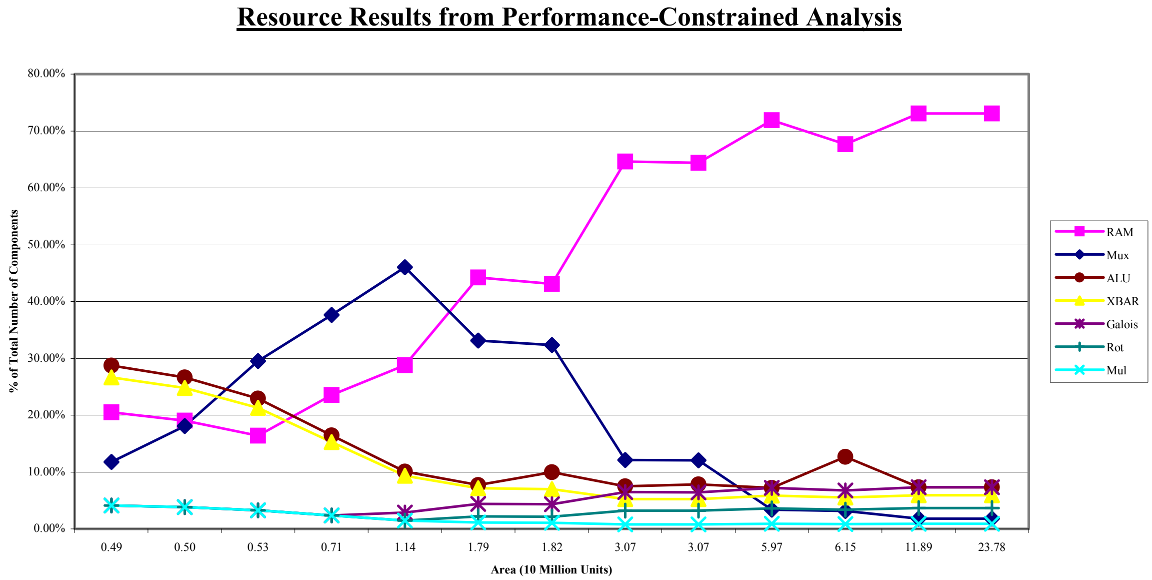

Figure 15 - *The functional unit ratio recommended by the performance-constrained selection technique.*

Resource Results from Area-Constrained Analysis

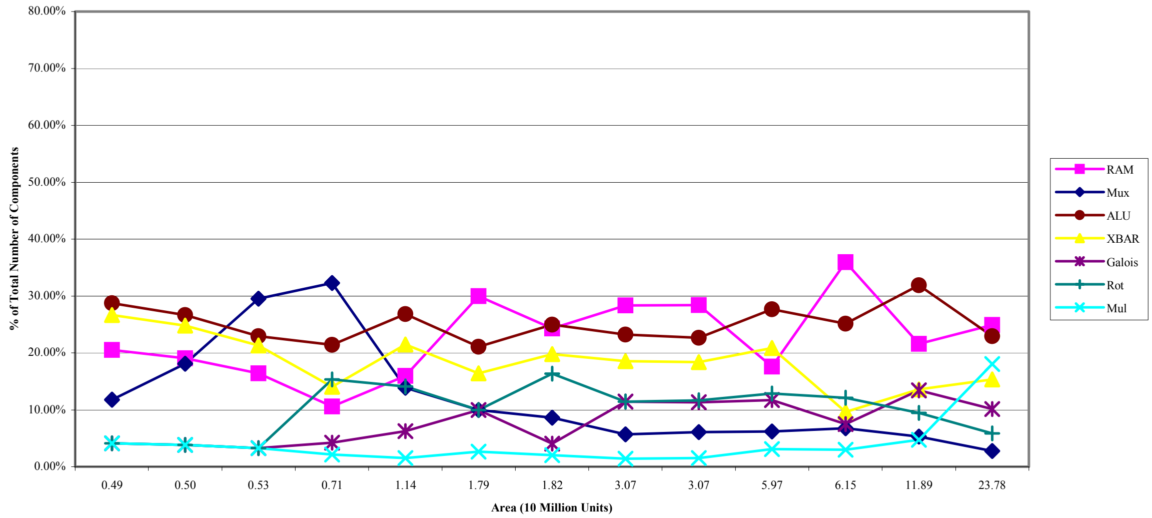

Figure 16 - *The functional unit ratio recommended by the more flexible area-constrained technique.*

Resource Results from Improved-Area Constrained Analysis

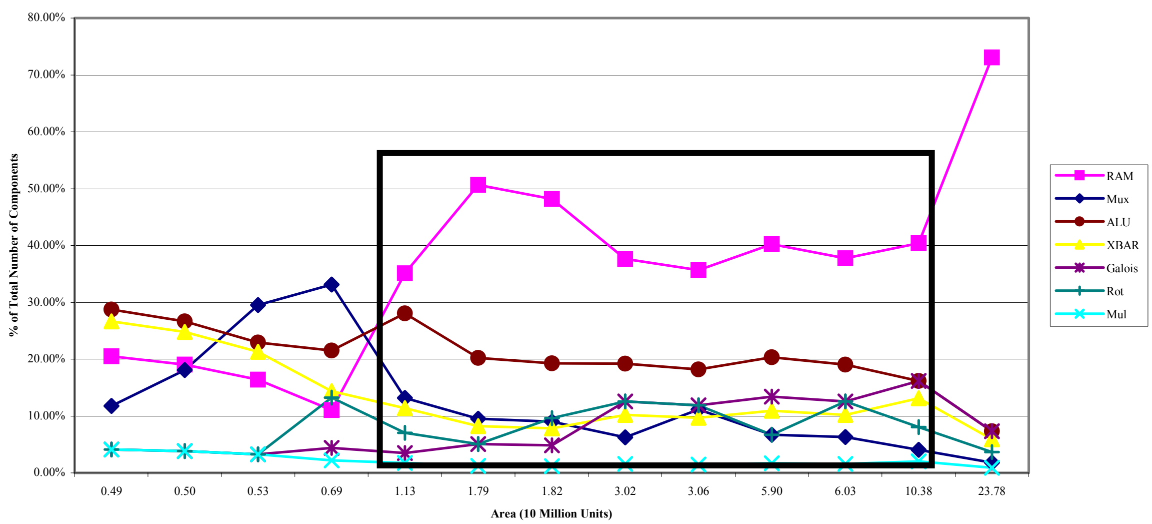

Figure 17 - *The functional unit ratio recommended by the improved area-constrained technique. The selected area is of most interest because it represents high performance implementations and the relative ratios of the various components are mostly stable*

We believe that scaleable, flexible moderate to high performance encryption architectures can be based on a tile-able cell structure. The results from our tests show that the improved area-constrained method best combined area and performance constraints. While taking special consideration for stable, high performance implementations and the possibility for future flexibility, we arrived at the component mixture shown in Figure 18. Although this is a large cell and would produce a very coarse-grained architecture, perhaps consisting of only 16 or 32 cells, it allows the target encryption algorithms to map with a minimum of resource wastage and a maximum of performance and flexibility. For example, an architecture consisting of 16 such cells would have an average throughput of one result every 6.7 clock cycles with an average component utilization of 79%.

Component Mixture

Unit Type	Num / Cell	% of Num	% of Area
MUX	9	18.00%	5.65%
RAM	16	32.00%	45.99%
Xbar	6	12.00%	7.34%
Mul	1	2.00%	11.04%
Galois	2	4.00%	10.45%
ALU	12	24.00%	16.20%
Rot	4	8.00%	3.33%

Figure 18 - Recommended component mixture extrapolated from functional unit analysis

9 Conclusions

In this paper we introduced a design problem unique to coarse-grained FPGA devices. Although we encountered the difficulties of functional unit selection while exploring an encryption-specific domain, we believe that the causes of the problem are not exclusive to this domain and can be expected to be common in any complex group of applications. We presented three algorithms that attempt to balance vastly different hardware requirements with performance and area constraints. The first algorithm produces a configuration with the absolution minimum area to guarantee a hard performance requirement. The second algorithm maximizes average throughput given a hard area limitation. The third algorithm combines these two strategies to offer good overall performance given less area.

The functional unit selection problem will become more difficult as reconfigurable devices are expected to offer better and better performance over large domain spaces. Increased specialization of function units and growing domain size combined with the need for resource utilization optimization techniques such as functional unit emulation will soon complicate architecture exploration beyond that which can be analyzed by hand. In the future, designers will need CAD tools that are aware of these issues in order to create devices that retain the flexibility required for customization over a domain of applications while maintaining good throughput and area characteristics.

10 References

[1] Ebeling, Carl; Cronquist, Darren C. and Paul Franklin; "RaPiD - Reconfigurable Pipelined Datapath"; *The 6th International Workshop on Field-Programmable Logic and Applications, 1996*

[2] Adams, C. and J Gilchrist; "The CAST-256 Encryption Algorithm"'; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[3] Lim, C.H. "CRYPTON : A New 128-bit Block Cipher"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[4] Knudsen, Lars R.; "DEAL - A 128-bit Block Cipher"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[5] Gilbert, H.; Girault, M.; Hoogvorst, P.; Noilhan, F.; Pornin, T.; Poupard, G.; Stern, J. and S. Vaudenay; "Decorrelated Fast Cipher : An AES Candidate" *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[6] Nippon Telegraph and Telephone Corporation; "Specification of E2 – A 128-bit Block Cipher"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[7] Georgoudis, Dianelos; Leroux, Damian; Chaves, Billy Simón and TecApro International S.A.; "The 'FROG' Encryption Algorithm"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[8] Schroeppel, Rich; "Hasty Pudding Cipher Specification"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[9] Brown, Lawrie and Josef Pieprzyk; "Introducing the New LOKI97 Block Cipher"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[10] Jacobson, M.J. Jr. and K. Huber; "The MAGENTA Block Cipher Algorithm"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[11] Burwick, Carolynn; Coppersmith, Don; D'Avignon, Edward; Gennaro, Rosario; Halevi, Shai; Jutla, Charanjit; Matyas, Stephen M. Jr.; O'Connor, Luke; Peyravian, Mohammad; Safford, David and Nevenko Zunic; "Mars – A Candidate Cipher for AES" *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998

[12] Rivest, Ron; Robshaw, M.J.B.; Sidney, R. and Y.L. Yin; "The RC6 Block Cipher"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998
[13] Daemen, Joan and Vincent Rijmen; "AES Proposal : Rijndael"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998
[14] Cylink Corporation; "Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES)"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998
[15] Anderson, Ross; Biham, Eli and Lars Knudsen; "Serpent : A Proposal for the Advanced Encryption Standard"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998
[16] Schneier, B.; Kelsey, J.; Whiting, D.; Wagner, D.; Hall, C. and N. Ferguson; "Twofish : A 128-bit Block Cipher"; *Proc. of 1st AES Candidate Conference*, Aug.20-22, 1998
[17] Schneier, Bruce; *Applied Cryptography*; John Wiley & Sons Inc.; pp 270 - 278

Appendix A – Galois Field Multiplication

Manipulating Galois Field variables is unique in that all operations - addition, subtraction, etc., begin with two variables in a field and result in an answer that also lies in the field. One difference between conventional multiplication and Galois Field multiplication is that an N-bit conventional multiplication results in a (2N)-bit product while a Galois Field multiplication, as mentioned earlier, must result in an N-bit product in order to stay in the field.

Galois Field multiplication begins in a similar manner to conventional multiplication in that all partial products are calculated in an identical manner. From that point though, there are two key differences. First, partial sums are calculated using bit-wise modulo 2 addition instead of conventional N-bit carry addition. Second, an iterative reduction may be performed to adjust the output to stay in the field. If the preliminary sum is greater or equal to 2^N, the result lies outside the N-bit field and must be XOR-ed with a left justified (N+1)-bit reduction constant. The most significant bit of the reduction constant is always a 1, so as to eliminate the most significant bit in the preliminary sum. This process is repeated until the result lies within the N-bit field. For all of the Galois field multiplications performed in the AES candidate algorithms, N is 8 and the reduction constant is “100011011”.

							A	1	0	1	0	1	0	1	0	
						X	B	0	0	1	0	0	0	1	1	
								1	0	1	0	1	0	1	0	Partial Products
							1	0	1	0	1	0	1	0		
						0	0	0	0	0	0	0	0			
					0	0	0	0	0	0	0	0				
				0	0	0	0	0	0	0	0					
			1	0	1	0	1	0	1	0						
		0	0	0	0	0	0	0	0							
⊕	0	0	0	0	0	0	0	0								
	0	0	1	0	1	0	0	1	0	1	1	1	1	1	0	Sum Reduction
Left justified to leading 1 →		⊕	1	0	0	0	1	1	0	1	1					
			0	0	1	0	1	0	0	0	0	1	1	1	0	
Left justified to leading 1 →				⊕	1	0	0	0	1	1	0	1	1			
					0	0	1	0	1	1	0	0	0	1	0	
Left justified to leading 1 →						⊕	1	0	0	0	1	1	0	1	1	
							0	0	1	1	1	1	0	0	1	

Appendix B – Hardware Requirements

Algorithm	Cycles *	32-bit 2:1 Mux	256B RAM	XBAR	32- bit Mul	Galois	ALU	ROT/shift	Area
CAST-256	4 X 12	10	16	0	0	0	5	1	726266
	4 X 6	20	32	0	0	0	10	2	1452532
	2 X 6	16	64	0	0	0	20	4	2722280
	1 X 6	8	128	0	0	0	40	8	5261776
	1 X 1	0	768	0	0	0	240	48	31205088
Crypton	12	8	16	16	0	0	36		1446400
	6	4	32	32	0	0	68	0	2735872
	4		48	48	0	0	100		4025344
	3	4	64	64	0	0	132	0	5375744
	2	4	96	96	0	0	196	0	8015616
	1	0	192	192	0	0	388	0	15904768
Deal	6 X 16	6	1	7	0	0	7	0	299200
	6 X 8	6	2	10	0	0	10	0	427776
	6 X 4	6	4	16	0	0	16	0	684928
	6 X 2	6	8	28	0	0	28	0	1199232
	6 X 1	4	16	52	0	0	52	0	2212608
	3 X 1	4	32	104	0	0	104	0	4394752
	2 X 1	4	48	156	0	0	156	0	6576896
	1 X 1	0	96	312	0	0	312	0	13092864
DFC	8	5	0	0	8	0	26	1	1546298
	4	5	0	0	16	0	52	2	3054516
	2	5	0	0	32	0	104	4	6070952
	1	1	0	0	64	0	208	8	12073360
E2	12	4	16	4	4	0	43	3	1918830
	6	4	32	8	8	0	78	3	3645806
	3	4	64	16	16	0	148	3	7099758
	2	4	128	24	24	0	218	3	11669870
	1	0	256	48	48	0	428	3	23117422
Frog	4 X 16 X 8	23	8	0	0	0	1	0	470592
	4 X 16 X 4	38	8	0	0	0	2	0	601216
	4 X 16 X 2	72	8	0	0	0	4	0	892928
	4 X 16 X 1	128	8	0	0	0	8	0	1384960
	2 X 16 X 1	256	8	0	0	0	16	0	2490880
	1 X 16 X 1	240	8	0	0	0	32	0	2631168
	1 X 8 X 1	120	16	0	0	0	64	0	2520576
	1 X 4 X 1	60	32	0	0	0	128	0	3670272
	1 X 2 X 1	30	64	0	0	0	256	0	6655104
	1 X 1 X 1	15	128	0	0	0	512	0	12967488
HPC	8	4	24	52	0	0	56	4	2597608
	4	4	48	104	0	0	112	8	5164752
	2	4	96	208	0	0	224	16	10299040
	1	0	192	416	0	0	448	32	20537152
Loki97	16 X 8	13	40	7	0	0	14	0	1827520
	16 X 4	11	80	7	0	0	16	0	3240256
	16 X 2	7	160	7	0	0	20	0	6065728
	16 X 1	4	320	7	0	0	28	0	11754752
	8 X 1	4	640	14	0	0	56	0	23479040
	4 X 1	4	1280	28	0	0	112	0	46927616
	2 X 1	4	2560	56	0	0	224	0	93824768
	1 X 1	0	5120	112	0	0	448	0	1.88E+08

*A x B notation indicates nested looping

Algorithm	Cycles *	32-bit 2:1 Mux	256B RAM	XBAR	32- bit Mul	Galois	ALU	ROT/shift	Area
Magenta	6 X 3 X 4	12	16	0	0	0	20	4	1017576
	6 X 3 X 2	12	32	0	0	0	22	4	1608424
	6 X 3 X 1	8	64	0	0	0	26	4	2759656
	6 X 1 X 1	8	192	0	0	0	74	12	8091576
	3 X 1 X 1	4	384	0	0	0	148	24	16091760
	2 X 1 X 1	4	576	0	0	0	222	36	24122408
	1 X 1 X 1	0	1152	0	0	0	444	72	48183888
Mars	16	12	24	0	1	0	36	8	1733200
	8	16	32	0	2	0	48	14	2433964
	4	32	64	0	4	0	96	28	4867928
	2	64	128	0	8	0	192	56	9735856
	1	128	256	0	16	0	384	112	19471712
RC6	8	4	0	0	2	0	10	6	522972
	4	4	0	0	4	0	16	12	949944
	2	4	0	0	6	0	28	24	1535856
	1	0	0	0	8	0	52	48	2409184
Rijndael	10 X 16	8	4	0	0	1	12	3	490790
	10 X 8	8	4	0	0	2	12	3	554206
	10 X 4	8	4	0	0	4	12	3	681038
	10 X 2	8	8	0	0	8	12	3	1074222
	10 X 1	4	16	0	0	16	12	3	1830126
	5 X 1	4	32	0	0	32	16	6	3498716
	2 X 1	4	80	0	0	80	28	15	8504486
	1 X 1	0	160	0	0	160	48	30	16816972
Safer+	8 X 16	8	16	0	0	4	11	0	1052896
	8 X 8	8	16	0	0	8	14	0	1355712
	8 X 4	8	16	0	0	16	20	0	1961344
	8 X 2	8	16	0	0	32	32	0	3172608
	8 X 1	4	16	0	0	64	56	0	5564672
	4 X 1	4	32	0	0	128	104	0	10967808
	2 X 1	4	64	0	0	256	200	0	21774080
	1 X 1	0	128	0	0	512	160	0	39555072
Serpent	32	8	8	32	0	0	16	8	1158096
	16	4	8	32	0	0	28	16	1405088
	8	4	16	32	0	0	52	32	2239040
	4	4	32	32	0	0	100	64	3906944
	2	4	64	32	0	0	196	128	7242752
	1	0	128	32	0	0	388	256	13883904
Twofish	16	4	8	0	0	8	17	3	1125678
	8	4	16	0	0	16	26	6	2089820
	4	4	32	0	0	32	44	12	4018104
	2	4	64	0	0	64	80	24	7874672
	1	0	128	0	0	128	152	48	15557344

*A x B notation indicates nested looping

Appendix C – Results

Results of Performance-Constrained Analysis

Max # of Cycles	Aggregate Throughput	# Mux	# RAM	# Xbars	# Mul	# Galois	# ALU	# Rot	Est. Area
512	902	23	40	52	8	8	56	8	4.92027E+06
256	646	38	40	52	8	8	56	8	5.03451E+06
160	518	72	40	52	8	8	56	8	5.29346E+06
96	360	128	80	52	8	8	56	8	7.11515E+06
48	224	256	160	52	8	16	56	8	1.13877E+07
24	155	240	320	52	8	32	56	16	1.79422E+07
16	129	240	320	52	8	32	74	16	1.82371E+07
12	103	120	640	52	8	64	74	32	3.06758E+07
10	97	120	640	52	8	64	78	32	3.07413E+07
6	58	60	1280	104	16	128	128	64	5.96530E+07
5	43	60	1280	104	16	128	240	64	6.14880E+07
3	25	64	2560	208	32	256	256	128	1.18879E+08
1	15	128	5120	416	64	512	512	256	2.37759E+08

	% of Total Number of Components							% of Total Area						
Max # of Cycles	Mux %	RAM %	XBAR %	Mul %	Galois %	ALU %	Rot %	Mux %	RAM %	XBAR %	Mul %	Galois %	ALU %	Rot %
512	11.79%	20.51%	26.67%	4.10%	4.10%	28.72%	4.10%	3.56%	28.36%	15.69%	21.79%	10.31%	18.65%	1.64%
256	18.10%	19.05%	24.76%	3.81%	3.81%	26.67%	3.81%	5.75%	27.71%	15.34%	21.30%	10.08%	18.22%	1.61%
160	29.51%	16.39%	21.31%	3.28%	3.28%	22.95%	3.28%	10.36%	26.36%	14.59%	20.25%	9.58%	17.33%	1.53%
96	37.65%	23.53%	15.29%	2.35%	2.35%	16.47%	2.35%	13.70%	39.22%	10.85%	15.07%	7.13%	12.90%	1.14%
48	46.04%	28.78%	9.35%	1.44%	2.88%	10.07%	1.44%	17.12%	49.01%	6.78%	9.41%	8.91%	8.06%	0.71%
24	33.15%	44.20%	7.18%	1.10%	4.42%	7.73%	2.21%	10.19%	62.21%	4.30%	5.98%	11.31%	5.11%	0.90%
16	32.35%	43.13%	7.01%	1.08%	4.31%	9.97%	2.16%	10.02%	61.20%	4.23%	5.88%	11.13%	6.65%	0.89%
12	12.12%	64.65%	5.25%	0.81%	6.46%	7.47%	3.23%	2.98%	72.77%	2.52%	3.50%	13.23%	3.95%	1.05%
10	12.07%	64.39%	5.23%	0.80%	6.44%	7.85%	3.22%	2.97%	72.62%	2.51%	3.49%	13.20%	4.16%	1.05%
6	3.37%	71.91%	5.84%	0.90%	7.19%	7.19%	3.60%	0.77%	74.84%	2.59%	3.59%	13.61%	3.52%	1.08%
5	3.17%	67.65%	5.50%	0.85%	6.77%	12.68%	3.38%	0.74%	72.61%	2.51%	3.49%	13.20%	6.40%	1.05%
3	1.83%	73.06%	5.94%	0.91%	7.31%	7.31%	3.65%	0.41%	75.11%	2.60%	3.61%	13.66%	3.53%	1.09%
1	1.83%	73.06%	5.94%	0.91%	7.31%	7.31%	3.65%	0.41%	75.11%	2.60%	3.61%	13.66%	3.53%	1.09%

Results of Area-Constrained Analysis

Max # of Cycles	Aggregate Throughput	# Mux	# RAM	# Xbars	# Mul	# Galois	# ALU	# Rot	Est. Area
512	902	23	40	52	8	8	56	8	4.92027E+06
256	646	38	40	52	8	8	56	8	5.03451E+06
128	518	72	40	52	8	8	56	8	5.29346E+06
128	297	122	40	53	8	16	81	58	7.11515E+06
64	156	73	84	113	8	33	141	74	1.13877E+07
32	92	64	192	105	17	64	135	64	1.79422E+07
32	93	68	192	156	16	32	197	129	1.82371E+07
16	55	64	320	209	16	129	262	129	3.06758E+07
16	55	69	321	208	17	128	256	132	3.07413E+07
16	38	136	387	459	68	258	608	283	5.96530E+07
8	28	146	774	208	64	161	541	261	6.14880E+07
8	23	204	828	521	182	514	1223	364	1.18879E+08
4	18	143	1282	789	930	523	1182	303	2.37759E+08

Max # of Cycles	% of Total Number of Components							% of Total Area						
	Mux %	RAM %	XBAR %	Mul %	Galois %	ALU %	Rot %	Mux %	RAM %	XBAR %	Mul %	Galois %	ALU %	Rot %
512	3.56%	28.36%	15.69%	21.79%	10.31%	18.65%	1.64%	11.79%	20.51%	26.67%	4.10%	4.10%	28.72%	4.10%
256	5.75%	27.71%	15.34%	21.30%	10.08%	18.22%	1.61%	18.10%	19.05%	24.76%	3.81%	3.81%	26.67%	3.81%
128	10.36%	26.36%	14.59%	20.25%	9.58%	17.33%	1.53%	29.51%	16.39%	21.31%	3.28%	3.28%	22.95%	3.28%
128	13.06%	19.61%	11.06%	15.07%	14.26%	18.65%	8.24%	32.28%	10.58%	14.02%	2.12%	4.23%	21.43%	15.34%
64	4.88%	25.73%	14.73%	9.41%	18.38%	20.29%	6.57%	13.88%	15.97%	21.48%	1.52%	6.27%	26.81%	14.07%
32	2.72%	37.33%	8.69%	12.70%	22.62%	12.33%	3.60%	9.98%	29.95%	16.38%	2.65%	9.98%	21.06%	9.98%
32	2.84%	36.72%	12.70%	11.76%	11.13%	17.70%	7.15%	8.61%	24.30%	19.75%	2.03%	4.05%	24.94%	16.33%
16	1.59%	36.39%	10.12%	6.99%	26.67%	13.99%	4.25%	5.67%	28.34%	18.51%	1.42%	11.43%	23.21%	11.43%
16	1.71%	36.42%	10.05%	7.41%	26.41%	13.64%	4.34%	6.10%	28.38%	18.39%	1.50%	11.32%	22.63%	11.67%
16	1.74%	22.63%	11.42%	15.28%	27.43%	16.70%	4.79%	6.18%	17.60%	20.87%	3.09%	11.73%	27.65%	12.87%
8	1.81%	43.91%	5.02%	13.95%	16.60%	14.42%	4.29%	6.77%	35.92%	9.65%	2.97%	7.47%	25.10%	12.11%
8	1.31%	24.29%	6.51%	20.52%	27.42%	16.86%	3.09%	5.32%	21.58%	13.58%	4.74%	13.40%	31.88%	9.49%
4	0.46%	18.81%	4.93%	52.42%	13.95%	8.15%	1.29%	2.78%	24.88%	15.31%	18.05%	10.15%	22.94%	5.88%

Results of Improved Area-Constrained Analysis

Max # of Cycles	Aggregate Throughput	# Mux	# RAM	# Xbars	# Mul	# Galois	# ALU	# Rot	Est. Area
512	902	23	40	52	8	8	56	8	4.92027E+06
256	646	38	40	52	8	8	56	8	5.03451E+06
128	518	72	40	52	8	8	56	8	5.29346E+06
128	297	120	40	52	8	16	78	48	6.93000E+06
32	156	60	160	52	8	16	128	32	1.13170E+07
16	103	60	320	52	8	32	128	32	1.79130E+07
16	98	60	320	52	8	32	128	64	1.82360E+07
16	57	64	384	104	16	128	196	128	3.01920E+07
16	57	120	384	104	16	128	196	128	3.06180E+07
8	30	128	768	208	32	256	388	128	5.90250E+07
8	29	128	768	208	32	256	388	256	6.03180E+07
4	18	128	1280	416	64	512	512	256	1.03820E+08
1	15	128	5120	416	64	512	512	256	2.37759E+08

Max # of Cycles	% of Total Number of Components							% of Total Area						
	Mux %	RAM %	XBAR %	Mul %	Galois %	ALU %	Rot %	Mux %	RAM %	XBAR %	Mul %	Galois %	ALU %	Rot %
512	3.56%	28.36%	15.69%	21.79%	10.31%	18.65%	1.64%	11.79%	20.51%	26.67%	4.10%	4.10%	28.72%	4.10%
256	5.75%	27.71%	15.34%	21.30%	10.08%	18.22%	1.61%	18.10%	19.05%	24.76%	3.81%	3.81%	26.67%	3.81%
128	10.36%	26.36%	14.59%	20.25%	9.58%	17.33%	1.53%	29.51%	16.39%	21.31%	3.28%	3.28%	22.95%	3.28%
128	13.19%	20.13%	11.14%	15.47%	14.64%	18.44%	7.00%	33.15%	11.05%	14.36%	2.21%	4.42%	21.55%	13.26%
32	4.04%	49.31%	6.82%	9.47%	8.97%	18.53%	2.86%	13.16%	35.09%	11.40%	1.75%	3.51%	28.07%	7.02%
16	2.55%	62.31%	4.31%	5.99%	11.33%	11.71%	1.81%	9.49%	50.63%	8.23%	1.27%	5.06%	20.25%	5.06%
16	2.51%	61.21%	4.23%	5.88%	11.13%	11.50%	3.55%	9.04%	48.19%	7.83%	1.20%	4.82%	19.28%	9.64%
16	1.61%	44.36%	5.11%	7.10%	26.89%	10.64%	4.28%	6.27%	37.65%	10.20%	1.57%	12.55%	19.22%	12.55%
16	2.98%	43.75%	5.04%	7.00%	26.51%	10.49%	4.22%	11.15%	35.69%	9.67%	1.49%	11.90%	18.22%	11.90%
8	1.65%	45.38%	5.23%	7.27%	27.50%	10.77%	2.19%	6.71%	40.25%	10.90%	1.68%	13.42%	20.34%	6.71%
8	1.62%	44.41%	5.12%	7.11%	26.91%	10.54%	4.29%	6.29%	37.72%	10.22%	1.57%	12.57%	19.06%	12.57%
4	0.94%	43.00%	5.95%	8.26%	31.27%	8.08%	2.49%	4.04%	40.40%	13.13%	2.02%	16.16%	16.16%	8.08%
1	0.41%	75.11%	2.60%	3.61%	13.66%	3.53%	1.09%	1.83%	73.06%	5.94%	0.91%	7.31%	7.31%	3.65%