Exploring Area/Delay Tradeoffs in an AES FPGA Implementation*

Joseph Zambreno, David Nguyen, and Alok Choudhary

Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208, USA
{zambro1, dnguyen, choudhar}@ece.northwestern.edu

Abstract. Field-Programmable Gate Arrays (FPGAs) have lately become a popular target for implementing cryptographic block ciphers, as a well-designed FPGA solution can combine some of the algorithmic flexibility and cost efficiency of an equivalent software implementation with throughputs that are comparable to custom ASIC designs. The recently selected Advanced Encryption Standard (AES) is slowly replacing older ciphers as the building block of choice for secure systems and is well suited to an FPGA implementation. In this paper we explore the design decisions that lead to area/delay tradeoffs in a single-core AES FPGA implementation. This work provides a more thorough description of the defining AES hardware characteristics than is currently available in the research literature, along with implementation results that are pareto optimal in terms of throughput, latency, and area efficiency.

1 Introduction and Motivation

Cryptography is one of the strongest tools for controlling against many kinds of security threats [1]. These algorithms and techniques form the basic building blocks of secure systems that serve a variety of purposes, including cryptographic hashing, secure key exchange, and digitally signing documents. Secure storage and transmission solutions are needed for all types of platforms, ranging from embedded devices where area is key to massively parallel machines that emphasize high performance. Such a diversity of requirements motivates the exploration of a wide range of cryptographic implementation characteristics.

Field-Programmable Gate Array (FPGA) technology is becoming a popular target for designing cryptographic ciphers, as witnessed by the wealth of recent research [2,3,4,5,6] and commercial [7] implementations. This increased interest in FPGAs from the cryptographic community has been driven by several factors:

- the individual operations required by this class of algorithms are generally simple in terms of required logic; as such any hardware implementation can increase efficiency by reducing the overhead introduced by software.

* This work was supported in part by the National Science Foundation (NSF) under grant CCR-0325207 and also by an NSF graduate research fellowship.

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 575–585, 2004.
© Springer-Verlag Berlin Heidelberg 2004

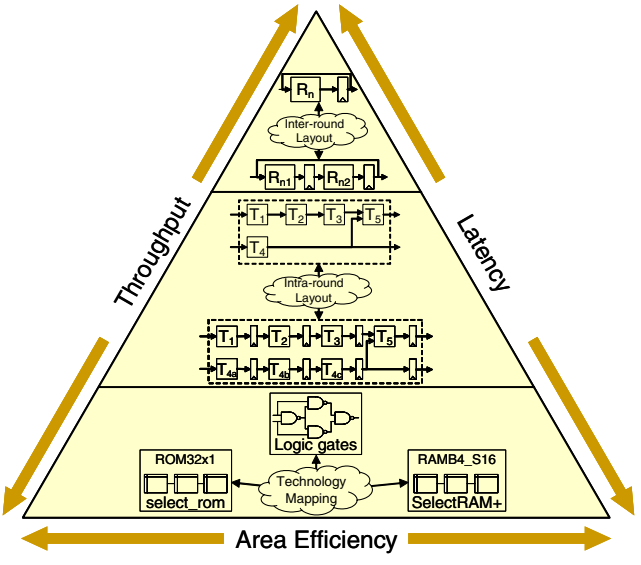

Fig. 1. Top-down block cipher design methodology

- the development process for FPGAs is extremely effective in terms of time-to-market and overall cost when compared to that of custom ASIC designs.
- the reconfigurable nature of FPGAs is especially attractive to cipher designers as it gives them the ability to apply modifications to the implemented algorithm after the initial time of programming [8]. This feature can be used to switch between a set of cryptographic algorithms at runtime [9], to address a freshly-discovered flaw in the cipher algorithm, or to optimize the architecture for a fixed range of inputs [5].

Much recent work in this field has focused on maximizing the theoretical throughput for these cryptographic block ciphers, including both the Data Encryption Standard (DES) [4] and the newly-introduced Advanced Encryption Standard (AES) [2,3,6]. The key distinction between the contributions of this paper and those implementations proposed previously stems from our top-down design methodology (Fig. 1) that allows for area and delay tradeoffs to be managed at several levels of the design hierarchy using a single parameterizable AES core.

As can be seen in Fig. 1, many current block ciphers can be described as a series of logic operations (rounds) that are repeated in an iterative fashion. At the *inter-round* level, decisions can be made as to how each round is laid out in terms of classical optimizations such as unrolling, tiling, and pipelining. Internal to each round structure there are *intra-round* decisions, which can include transformation partitioning and internal pipelining. Also, when considering FPGA

hardware, *technology mapping* is especially important, as these decisions can greatly influence designs by allocating specialized resources towards individual computations. Ultimately, making these decisions at each level of the design hierarchy provides much additional control over the performance and area characteristics of the resulting AES FPGA implementation. Our experimental results using Xilinx Virtex-II technology demonstrate that the careful application of this concept can lead to different designs with small area, high throughput, and low latency. One such implementation obtained a maximum throughput of 23.57 Gbps, which to the authors' knowledge is greater than any previously published value.

The remainder of this paper is organized as follows. In Sect. 2, an overview is provided of the AES encryption algorithm, with an introduction to the organization and functionality of the individual transformations from a hardware designer's perspective. Section 3 explores the key design decisions that are possible at various levels in the AES FPGA implementation process, discussing how these choices can result in significant area and delay tradeoffs. Experimental results are presented in Sect. 4, demonstrating how a single soft core can be fine-tuned to implement optimized designs in terms of performance, area, and efficiency metrics. Finally, the paper is concluded in Sect. 5 with a broad summary of some relevant ideas that require further exploration.

2 Overview of AES

In 1997, the U.S. National Institute of Standards and Technology (NIST) announced an open international competition for cipher designs to replace the aging DES as the federal information processing standard. The fifteen submissions to become the new AES standard were publicly evaluated based on algorithmic security, simplicity, and suitability to both hardware and software implementations. Among these submissions was the Rijndael algorithm, which was developed by Vincent Rijmen and Joan Daemen [10]. As it was well fitted to the above factors, Rijndael was selected as the AES competition winner in 2000.

AES is what is known as a symmetric key block cipher, *block cipher* meaning that it operates on fixed-length blocks of data at a time, *symmetric key* meaning that the same key is used during encryption and decryption [1]. Although the original Rijndael specification allowed for both blocks and keys of various lengths, AES is restricted to 128-bit blocks and keys of 128, 192, or 256 bits. In symmetric block ciphers, the algorithms for encryption and decryption using various key lengths often contain quite a large amount of similar features. Consequently, little context will be lost by restricting our focus for the remainder of this paper on AES encryption using a 128-bit key (AES-128E).

The structure of AES-128E is as follows (see Fig. 2). The initial 128-bit key is fed into the `KeyExpansion` function which produces separate keys for each of the 10 required rounds. These rounds combine their scheduled keys with a two

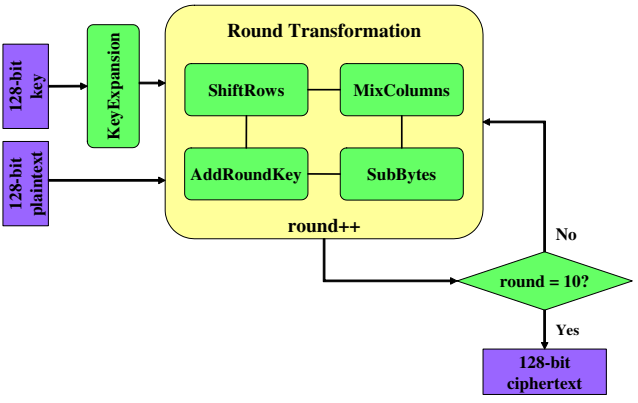

Fig. 2. Algorithmic view of AES-128E

dimensional representation of the input (the "state") using various transformations [11]:

- `SubBytes` calculates a non-linear function independently on each byte of the state. The substitution used by this transformation can be more simply represented as a lookup table which is referred to as an "S-box".
- `MixColumns` separately modifies each column of the state in what is essentially a matrix multiplication operation. Fortunately, in the 8-bit finite mathematical field relied on by this class of block ciphers, multipliers can be replaced with simpler fixed-length shifts and XOR operations.
- `ShiftRows` cyclically shifts the bytes in the last three rows of the state. As this function requires no computational hardware it can be implemented on an FPGA as simple wiring.
- `AddRoundKey` adds the round key to the state using a bitwise XOR operation.

For those interested in a more thorough description of the AES algorithm, both the official AES standardization documentation [11] and the developers' own writings [10] are informative reads.

3 Design Space Exploration

As the effects of FPGA design decisions on performance and area are often specific to individual architectures, it is necessary to further refine the FPGA target before proceeding in the analysis. For our experiments, we selected the Xilinx Virtex-II device family [12]. Like most Xilinx FPGAs, the Virtex-II devices can be best described as a two-dimensional array of Configurable Logic Blocks (CLBs) that are surrounded by I/O resources and routed together using a programmable interconnect mesh. These CLBs contain functional elements for implementing both combinatorial and synchronous logic, and also include some

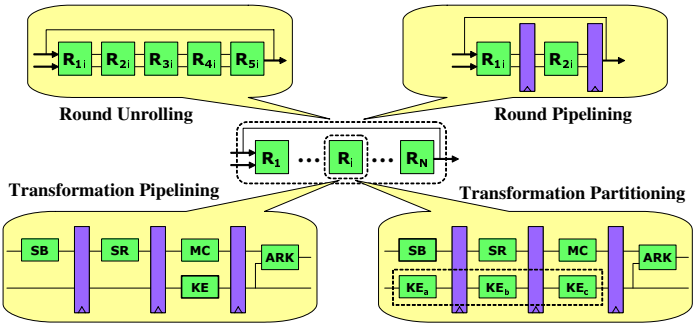

Fig. 3. AES-128E design decisions: *unrolling*, *pipelining*, and *partitioning*

sequential storage. Apart from the CLBs, Virtex-II FPGAs also contain a dedicated amount of dual-ported Block SelectRAM (BRAM) memory modules, each of which can hold up to 18 Kbits of data.

Given a target FPGA similar to the Virtex-II, the first design decision that needs to be made is in regards to the `KeyExpansion` routine. While the operations required to generate a key schedule from the original input key are not complicated, it makes intuitive sense to consider splitting this functionality into smaller `KeyExpansion` modules that would be placed alongside the actual round operations, in what is known as *online* key generation. This is due to the fact that the round key is not used until the final operation in each round (the `AddRoundKey` operation).

3.1 Inter-round Layout

Given that AES-128E is, at its highest level, essentially an iterative looping structure, it is interesting to look at the effect of some classical loop layout optimizations. *Unrolling* replaces a loop body with N copies of that loop body (Fig. 3). As the AES-128E algorithm is a single loop that iterates 10 times, any unrolling amount $1 \leq N \leq 10$ is valid, with $N = 1$ corresponding to the original looping case and $N = 10$ specifying a fully unrolled implementation.

Unrolling the rounds makes them highly amenable to *pipelining*, which is a technique that increases the number of blocks of data that can be processed concurrently. As can be seen in Fig. 3, pipelining in FPGA designs can be implemented by inserting registers between the modules that need to operate independently. Different implementations can be created that split the unrolled rounds into a certain number of pipeline stages, with a similar restriction as before that each stage should be of equal length.

The main advantage of unrolling and pipelining is that it increases the parallelism of the AES encryption algorithm, which should have a positive effect on throughput. It is also possible that a fully unrolled but not pipelined implementation will have a lower latency than in its iterative form. These performance

advantages do not come without a price, as unrolling will increase the required FPGA resources by approximately a factor of N; registers used for pipelining will also consume CLBs.

3.2 Intra-round Layout

The clock frequency that FPGAs can operate at is dependent on the critical logic path of the design. As such, the unrolling and pipelining of the rounds as discussed in the previous section will have little positive effect on the critical path when compared to an iterative round structure. In general, this maximum delay will be dependent on the individual transformations inside each round.

Fortunately, these sub-modules are also eligible for pipelining. Figure 3 shows an example of this *transformation pipelining*, where each of the AES transformations are represented by their initials (e.g. `SB` for `SubBytes`). This will reduce the critical path to that of the individual transformation with the greatest delay. We can improve upon this value even further by *partitioning* some of the transformations. Assuming that the `KeyExpansion` operation is performed online, it becomes a prime candidate for partitioning as it has the most slack between the time of its valid input and expected output. After that point it is likely that the maximum delay path will shift to another transformation, which can often also be partitioned. The level of partitioning can be tuned by initially creating highly-partitioned versions of the AES block transformations, and then connecting them with a variable number of pipeline registers.

When combined with round pipelining, transformation partitioning can lead to extremely large gains in throughput, with a relatively small increase in area due to the additional registers that would be needed. However, these heavily pipelined configurations will have extremely long latencies when compared to the base iterative version of AES-128E.

3.3 Technology Mapping

While the majority of the computations needed for the round transformations can be directly mapped to CLBs using the proper synthesis tools, the `SubBytes` operation, or more specifically the S-box tables found in `SubBytes`, can be implemented in one of several ways using Virtex-II technology:

- *Block SelectRAM* - the values in the lookup table for each S-box can be loaded onto these memories at configuration time. Since the memories are dual-ported, each RAM block can implement two separate S-boxes. Block SelectRAMs are dedicated resources on Virtex-II FPGAs, meaning that there is a hard upper limit on the number of them in any design.
- *Distributed SelectRAM* - distributed ROM primitives with the pre-loaded S-box values can also be synthesized directly using CLBs. These are often faster than the Block SelectRAMs, but will require additional glue logic.

– *Logic* - the lookup table code can be converted to a logical representation to be implemented on the CLBs. An advantage of this option is that it provides additional room for the synthesis tools to optimize for area and delay.

Because they are so numerous, the choice of lookup table technology can have a significant effect on the area/delay profile of the `SubBytes` operation and of AES-128E as a whole. Although less in number, these S-box operations are also needed in `KeyExpansion`, whether it is performed online or off.

4 Area and Performance Results

4.1 Experimental Setup

The AES-128E algorithm was implemented using a single VHDL core, with a configuration file to drive preset macros for controlling the round layouts and explicit synthesis directives to determine the mapping of S-boxes. For synthesis we used Synplify Pro 7.2.1 from Synplicity, which was configured to target a Xilinx XC2V4000 FPGA. The XC2V4000 is a medium-sized member of the Virtex-II device family, containing 5760 CLBs (equivalent to 23040 slices) and 120 Block SelectRAM modules. Xilinx ISE 5.2i was used for the place-and-route and timing analysis.

For each design we used these tools to measure the maximum possible clock rate (f_{clk}), the number of utilized slices (N_{slice}), and the number of Block SelectRAMs (N_{bram}) From these base statistics we calculated the resultant maximum throughput using the following equation for a block cipher in non-feedback mode:

$$Tput = \frac{128 \cdot f_{clk}}{blocks\_per\_cycle} \ , \tag{1}$$

where the number of blocks per cycle is 1 for a fully unrolled implementation, and greater than 1 for any design that re-uses the round structures to process a single input. Also, the latency required to encrypt a single block can be calculated as:

$$Lat = \frac{10 \cdot stages\_per\_round}{f_{clk}} \ , \tag{2}$$

where the number of clock cycles needed to process a single round is an average and may be a non-integer value. Finally, some idea about the efficiency of an implementation can be obtained by analyzing the following metric:

$$Eff = \frac{Tput}{N_{slice}} \ , \tag{3}$$

which is measured in throughput rate (bps) per utilized CLB slice.

Table 1. AES-128E implementation results for a Xilinx XC2V4000 FPGA

Design	f_{clk} (MHz)	N_{slice}	N_{bram}	$Tput$ (Gbps)	Lat (ns)	Eff ($\frac{\text{Mbps}}{\text{slice}}$)
UF1-PP0B	110.16	**387**	10	1.41	90.78	3.64
UF1-PP0D	77.91	1780	0	1.00	128.4	0.56
UF1-PP0L	59.00	2744	0	0.76	169.5	0.28
UF1-PP3D	178.09	1940	0	2.28	168.5	1.18
UF1-PP3L	147.75	2909	0	1.89	203.0	0.65
UF2-PP1B	118.57	753	20	3.04	84.34	4.04
UF2-PP2B	150.02	1011	20	3.84	133.3	3.80
UF2-PP2D	119.96	3445	0	3.07	166.7	0.89
UF2-PP3B	173.37	1254	20	4.44	173.0	3.54
UF2-PP3L	118.30	5570	0	3.03	253.6	0.54
UF5-PP0B	72.438	1532	50	4.64	**27.61**	3.03
UF5-PP1D	68.521	7995	0	4.39	145.9	0.55
UF5-PP2B	169.92	2206	50	10.88	117.7	4.93
UF5-PP2L	76.26	11974	0	4.88	262.3	0.41
UF5-PP3B	173.73	2810	50	11.12	172.7	3.96
UF10-PP1B	95.129	2518	100	12.18	105.1	4.84
UF10-PP1D	50.239	15365	0	6.43	199.0	0.418
UF10-PP2B	179.147	3766	100	22.93	111.6	**6.09**
UF10-PP3B	183.58	4901	100	23.50	163.4	4.79
UF10-PP3D	**184.16**	16938	0	**23.57**	162.9	1.39

4.2 Results

A selection of our experimental results can be found in Table 1. Each design is labeled UF*X*-PP*YZ*, where X corresponds to the round unrolling factor $X \in \{1, 2, 5, 10\}$. The Y value specifies the amount of transformation partitioning and pipelining; for $Y = 0$ the design has no pipelining, for $Y = 1$ each unrolled round is pipelined, for $Y = 2$ each round is split into two stages, and for $Y = 3$ each round is split into three stages, with transformations being partitioned across those stages. The Z value specifies the S-box technology mapping in the design, where for $Z =$ [B] Block SelectRAM is chosen, for $Z =$ [D] distributed ROM primitives are chosen, and for $Z =$ [L] logic gates are instantiated. For sake of brevity, unexceptional results from the set of possible designs were pruned when forming Table 1.

From these results several trends can be observed. As was expected, unrolling increased the number of slices by a significant amount. See UF10-PP3D which uses over 8.7× the amount of slices of its iterative counterpart UF1-PP3D. The gain in throughput often out-paced this increased area consumption, leading to an improved area efficiency with the larger unrolling factor. Also, for many of the designs using the distributed SelectRAM resulted in slightly higher clock rates when compared to the dedicated Block SelectRAM. The advantage to using the Block SelectRAM for the S-boxes can be seen in both the area consumption

and area efficiency – see the `UF2-PP2B` design which for the cost of 20 BRAMs saves 71% of the slices that `UF2-PP2D` requires. Since the performance of the two designs were fairly similar, this choice leads to a 4.26× increase in efficiency. It should also be noted that using logic gates to directly implement the S-box transformations was sub-optimal in all measured metrics when compared to using either SelectRAM type. Finally, aggressive transformation partitioning was quite effective in improving the clock rate and resultant throughput. This technique was successful even in the iterative case, illustrated by the `UF1-PP3L` design which obtained 2.5× the throughput of `UF1-PP0L` by partitioning the critical path.

The bolded values in Table 1 represent the designs which are optimal in terms of the selected area/performance characteristics. As was expected, the design that required the least amount of area (387 slices) was `UF1-PP0B`, an iterative implementation with no transformation partitioning that used Block SelectRAMs. Without using these SelectRAMs, the smallest design was `UF1-PP0D`, which required 1780 slices. Two of the most aggressively unrolled and pipelined designs (`UF10-PP3B` and `UF10-PP3D`) obtained clock rates of over 183 MHz, resulting in throughputs of over 23.5 Gbps. A design that demonstrates the usefulness of partial unrolling is `UF5-PP0B`, which has the lowest latency of all designs (27.61 ns). The `UF10-PP2B` design had the highest area efficiency (6.09 Mbps/slice).

4.3 Related Results

It is difficult to make direct comparisons between FPGA implementations of any algorithm since the specific hardware target is often different. However, many recent AES implementations have provided maximum throughput numbers for Xilinx FPGAs that can be used as a measuring stick. For example, Helion Technology reports throughputs of over 16 Gbps [7] for their high-performance commercial AES core. Also, several academic groups have reported high throughput values for designs that are similar to our most aggressively pipelined version. Järvinen et al. [6] created a cipher that operated at 17.8 Gbps, while Saggese et al. [3] reached just over 20 Gbps. Besides ours, the highest reported throughput for an AES implementation belongs to Hodjat and Verbauwhede [13], who designed a 21.54 Gbps core. Our superior throughput numbers can be explained by the fact that our top-down design flow motivated the discovery of additional ways of partitioning transformations, and that doubly packing S-boxes into Block SelectRAMs allowed for a completely unrolled design to fit into a relatively small device.

As a comparison to non-FPGA technologies, a hand-optimized assembly implementation of AES encryption in feedback mode achieved 1.538 Gbps on a 3.2 MHz Pentium IV processor [14]. An ASIC version of the design from [13] targeting $0.18\mu m$ technology was able to achieve greater than 30 Gbps [15]. While this ASIC implementation is far superior to any published FPGA implementation in

terms of throughput, this shortcoming is tolerable when considering the other advantages to FPGA technology as listed in Sect. 1.

5 Conclusions

In this paper a top-down methodology for implementing cryptographic block ciphers on FPGAs was proposed and evaluated. For AES-128E it was shown that these design decisions can be managed in a fashion that allows for fine tuning some of the area and delay characteristics. This methodology has been used to discover implementations that are competitive with others in terms of area, latency, and area efficiency. For the future, it would be interesting to see what technology-specific features of other FPGA device families could be exploited to further optimize AES. This same design methodology should also be extended to other cryptographic algorithms. Finally, it would be useful to further investigate how partial reconfiguration can optimize a block cipher given some knowledge of the input key pattern.

References

1. W. Stallings. *Cryptography and Network Security*, Prentice Hall, 2003.
2. A. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists. In *Proc. of the Third Advanced Encryption Standard (AES3) Candidate Conference*, pages 13–27, 2000.
3. G. P. Saggese, A. Mazzeo, N. Mazzoca, and A. G. M. Strollo. An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In *Proc. of the 13th Int'l Conference on Field-Programmable Logic and its Applications (FPL)*, pages 292–302, 2003.
4. J-P. Kaps and C. Paar. Fast DES implementation for FPGAs and its application to a universal key-search machine. In *Proc. of the 5th Annual Workshop on Selected Areas in Cryptography (SAC)*, pages 234–247, 1998.
5. I. Gonzalez, S. Lopez-Budeo, F. J. Gomez, and J. Martinez. Using partial reconfiguration in cryptographic applications: an implementation of the IDEA algorithm. In *Proc. of the 13th Int'l Conference on Field-Programmable Logic and its Applications (FPL)*, pages 194–203, 2003.
6. K. U. Järvinen, M. T. Tommiska, and J. O. Skyttä. A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In *Proc. of the Int'l Symposium on Field Programmable Gate Arrays (FPGA)*, pages 207–215, 2003.
7. Helion Technology, Inc. AES Xilinx FPGA core data sheet. available at http://www.heliontech.com, 2003.
8. T. Wollinger and C. Paar. How secure are FPGAs in cryptographic applications? In *Proc. of the 13th Int'l Conference on Field-Programmable Logic and its Applications (FPL)*, pages 91–100, 2003.
9. A. Dandalis, V. Prasanna, and J. Rolim. An adaptive cryptographic engine for IPSec architectures. In *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 132–144, 2000.

10. J. Daeman and V. Rijmen. The block cipher Rijndael. *Smart Card Research and Applications,LNCS 1820*, J-J. Quisquater and B. Schneier, Eds., Springer-Verlag, pages 288–296, 2000.
11. National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). *FIPS PUB 197*, available at http://csrc.nist.gov, 2001.
12. Xilinx, Inc. Virtex-II complete data sheet. available at http://www.xilinx.com, 2003.
13. A. Hodjat and I. Verbauwhede. A 21.54 Gbits/s fully pipelined AES processor on FPGA. In *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2004.
14. H. Lipmaa. AES implementation speed comparison. available at http://www.tcs.hut.fi/~aes/rijndael.html, 2003.
15. A. Hodjat and I. Verbauwhede. Minimum area cost for a 30 to 70 Gbits/s AES processor. In *Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 83–88, 2003.