SANDIA REPORT

SAND2005-6137
Unlimited Release
Printed October 2005

Small Circuits for Cryptography

W. Erik Anderson, Timothy J. Draelos, Richard C. Schroeppel,
Mark D. Torgerson, and Russell D. Miller

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2005-6137
Unlimited Release
Printed October 2005

Small Circuits for Cryptography

Erik Anderson, Timothy Draelos, Richard Schroeppel, and Mark Torgerson
Cryptography and Information Systems Surety Department

Russell D. Miller
Physical Design and Product Realization Department

Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-0785

Abstract

This report examines a number of hardware circuit design issues associated with implementing certain functions in FPGA and ASIC technologies. Here we show circuit designs for AES and SHA-1 that have an extremely small hardware footprint, yet show reasonably good performance characteristics as compared to the state of the art designs found in the literature. Our AES performance numbers are fueled by an optimized composite field S-box design for the Stratix chipset. Our SHA-1 designs use register packing and feedback functionalities of the Stratix LE, which reduce the logic element usage by as much as 72% as compared to other SHA-1 designs.

Contents

1 Small Circuits for Cryptography **9**

1.1 Introduction 9

1.2 Stratix Architecture 10

2 Advanced Encryption Standard (AES) **12**

2.1 AES Architecture 12

2.2 Round Component Optimizations 14

2.3 Performance and Analysis 16

3 Secure Hash Algorithm (SHA-1) **18**

3.1 SHA-1 Architecture 20

3.2 Implementation of the Logic Functions 21

3.3 Performance and Analysis 23

4 Concluding Remarks **24**

List of Figures

1.1 Stratix LE in normal mode 11

2.1 AES encryption 13

2.2 Architecture of AES 14

2.3 State table for 128-bit AES 15

2.4 AES 32-bit Data Unit 16

3.1 SHA-1 Top Level Description 18

3.2 Round Mix operation 19

3.3 Architecture of SHA-1 21

3.4 Implementation of f_t using Stratix logic elements 22

List of Tables

2.1 AES performance figures for various cores 17

3.1 SHA-1 performance figures for various cores 23

Chapter 1

Small Circuits for Cryptography

1.1 Introduction

In the world of network communications there seems to be an ever increasing appetite for bandwidth and throughput. New applications require more bandwidth and users simply require more data to satisfy their needs. This means that there is a good deal of pressure placed on the networking community to build faster networks. The faster networks are built with new hardware and software products designed specifically to handle the latest and greatest speeds.

The networking community realizes the need for security in digital data communications. Cryptographic processes that are applied to networking technologies must be able to support the ever increasing network speeds. Thus the cryptographic community follows the typical trend to build extremely fast cryptographic primitives and to develop creative optimizations of existing primitives.

In software, there are a number of design principles that can allow software implementations to be slower or faster. One such principle is memory usage. Typically one may trade computations for a series of look ups. Look up tables tend to be fast provided that they are not too big, but with each cryptographic design there are trade-offs as to when a table look up is more efficient than just doing the calculation. In software, memory usage is not an overly large concern. Typical computing platforms have plenty of memory available.

With today's small feature sizes, ASIC and FPGA designers have a great deal of freedom to build very large, very fast circuits that will keep up with the network speed requirements. Some of the design trade-offs found in software are also found in hardware. One may trade calculations for memory look-ups. Depending on the trades one will increase speed by using more silicon and power.

In typical applications the drive for speedier cryptographic applications makes sense and the fact that more memory or power is used to accomplish that goal is acceptable because memory and power are typically cheap and plentiful. It is interesting to note that the typical commercial problem of pushing for ever increasing speeds has left somewhat of a void on the other end of the spectrum. Little research has been conducted in the cryptographic

community to optimize modern cryptographic primitives in a resource constrained environment. Thus non-standard applications that have highly limiting constraints are not as well supported as one may like.

The focus of this report is to examine hardware implementations of modern cryptographic primitives that may be more suited to hardware devices with little available memory and limited power usage. Indeed, we focus on the Advanced Encryption Standard (AES) and the Secure Hashing Algorithm (SHA-1). Both algorithms are standards supported by the National Institute of Science and Technology (NIST) and are suitable for, and in some cases required for, sensitive but unclassified government data transactions.

Our hardware designs have two flavors. The first is Field Programmable Gate Arrays (FPGA) and the second is Application Specific Integrated Circuits (ASIC). The reasoning behind focusing our research effort in two different directions is that we feel that most custom applications with extreme resource constraints will be implemented in an ASIC. However, designing with FPGAs is cheaper, and more accessible than going straight to ASIC technologies. Provided one looks toward the eventual ASIC design, FPGA technologies seem to be a wonderful environment for research, development, and conducting experiments in. Indeed, they provide a logical stepping stone along the path to ASIC development and optimization. With ASIC development in mind, our designs attempt to minimize overall circuit size and power consumption, while at the same time sacrificing as little of the speed as is possible.

The report proceeds as follows: Section 1.2 continues with a description of the Stratix FPGA device family. Chapter 2 discusses implementation strategies and details of AES. Similarly, Chapter 3 discusses SHA-1.

1.2 Stratix Architecture

The Stratix FPGA device family incorporates a number of different technologies in its architecture. It contains embedded DSP blocks, phase lock loops, configurable logic, and three different types of memory. The DSP blocks can implement shift registers and multipliers up to 36×36 bits, which make them ideal for signal processing applications. The memory sizes come in 512, 4K, and 512K bits, each supporting a variety of configurations. The Stratix devices are set up in a row and column based layout. The different components on the chipset are connected through row and column interconnects. These links establish communication pathways between the configurable logic in the logic array blocks (LABs), DSP blocks, and memories.

The most basic component in the Stratix chipset is a logic element (LE). An LE can implement both synchronous and asynchronous types of logic. Each LE contains a reconfigurable 4-1 lookup table (LUT) and 1-bit programmable register. There are 10 LEs per logic array block which are interconnected through local connection lines. Fig. 1.1 illustrates a Stratix LE operating in normal mode. Each LE has three outputs which are independently driven by either the LUT or register. This feature is called register packing and is designed to improve logic utilization. The register has clock, clock enable, and clear inputs and can be driven by either internal logic or I/O pins. To speed up register feedback operations

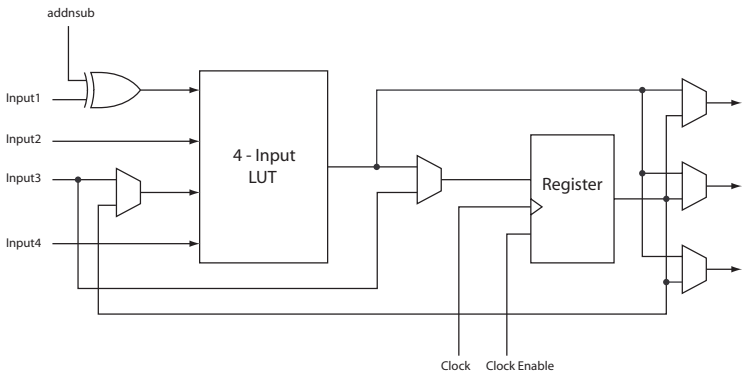

Figure 1.1. Stratix LE in normal mode

each LE has a register feedback line driving the 4-1 LUT as a muxed input. The register may also be synchronously driven by either the LUT or directly by the LE inputs. Besides normal mode an LE can also be reconfigured to operate in dynamic arithmetic mode. Dynamic arithmetic mode offers dynamic addition and subtraction, and supports clock/counter enable, and clear/load inputs. For a more detailed description of these types of modes see [18].

In our implementation we used a Stratix EP1S40F780C5 FPGA chip. It contains 41250 logic elements, 384 512, 183 4K, and 4 512K RAM blocks. We synthesized and compiled our architectures using Altera's Quartus II 5.0 software. Our circuit descriptions were written in Verilog HDL code.

Chapter 2

Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) was selected in 2000 by the US National Institute of Standards and Technologies as a replacement to the DES block cipher. AES has grown in popularity over the past few years and has been accepted as the de facto standard in the cryptographic community. AES is based on Rijndael [6], a multi-round iterated block cipher with key and block lengths ranging from 128, 192, and 256 bits in size. There have been numerous research proposals on various hardware implementations of the AES block cipher. Most hardware implementations have focused on developing architectures that maximize data throughput. These architectures usually utilize fast unrolled pipelines with algorithmic optimizations made to the AES S-boxes. Recently, attention has grown to developing compact implementations of AES with the goal of reducing logic area [5, 8, 15, 13, 17]. These implementations are ideal for low power embedded applications that require cryptographic security. We explore various techniques in compacting the AES algorithm and develop a AES architecture for both FPGA and ASIC technologies.

Our analysis begins by examining the structure of the AES block cipher. We will focus on the 128-bit version of AES due to its popularity. We follow our discussion with a careful examination of various algorithmic enhancements. We incorporate these enhancements into our architecture and conduct a performance comparison to other designs contained in the literature. Our architecture was developed around the Stratix FPGA chipset, therefore all of our techniques and optimizations are based on 4-1 LUTs. Since gates are much smaller than their LUT counterparts this approach may not be optimal for ASIC technologies. The difference between any area discrepancies should be rather small, therefore the proposed architecture should provide near optimal results.

2.1 AES Architecture

The encryption round transformation is comprised of four basic components *ShiftRow*, *ByteSub*, *MixColumn*, and *AddRoundKey*. The individual order of each component is important with the exception of *ByteSub* and *ShiftRow* which can be reversed. The 128-bit version

$\mathbf{AES}_{128}(M, K)$

Key Scheduler(K)
Let $W_0.W_1.W_2.W_3 \leftarrow K$
for $i \leftarrow 4$ **to** 43 **do**
  $temp \leftarrow W_{i-1}$
  If $i = 0 \bmod 4$
    $temp \leftarrow ByteSub(RotByte(temp)) \oplus Rcon(i/4)$
  $W_i \leftarrow W_{i-4} \oplus temp$
for $t \leftarrow 0$ **to** 10 **do**
  $RoundKey_t \leftarrow (W_{4t}, W_{4t+1}, W_{4t+2}, W_{4t+3})$

Encrypt$(M, RoundKey_0, \ldots, RoundKey_10)$
Let $State \leftarrow M \oplus RoundKey_0$
for $t \leftarrow 1$ **to** 9 **do**
  $State \leftarrow RoundTransform(State, RoundKey_t)$
$State \leftarrow FinalRound(State, RoundKey_10)$
return $Ciphertext \leftarrow State$

Figure 2.1. AES encryption

of AES consists of 10 rounds of round transformation. The first 9 rounds apply *ShiftRow*, *ByteSub*, *MixColumn*, and *AddRoundKey* while the final round applies all operations except *MixColumn*. For distinction these are labeled as *RoundTransform* and *FinalRound* respectively, see Fig. 2.1 for an illustration.

The round structure implementation of AES has several basic architectures to choose from. The fastest of these architectures unrolls the AES algorithm and applies pipeline stages to maximize the clock rate. This architecture achieves high data throughput at the expense of more resource utilization. The size of the architecture can be reduced by rolling the AES algorithm into a single round. This allows processing over a smaller number of logic resources. The AES round structure allows for even further size reduction. It is possible to break up the *ByteSub* and *MixColumn* components into four equal pieces and apply a single piece iteratively over 4 clock cycles. This can effectively reduce the circuit size by approximately 75% with an order of 4x cost in speed. The authors in [8] go one step further and break the *ByteSub* component into sixteen pieces. This configuration requires more control logic and does not reduce the circuit to 1/16 its size. A careful balance between the extra control logic and *ByteSub* reduction must be applied.

For our design we implemented a 4 clock per round rolled architecture. The 128-bit AES state is stored in a 128-bit shift register and shifted every clock by 32 bits. The round keys are precomputed by the circuit and stored in memory. It does not make sense from an efficiency standpoint to compute the round keys on the fly, since most encryption schemes do not update their symmetric keys after each block cipher instantiation. For additional

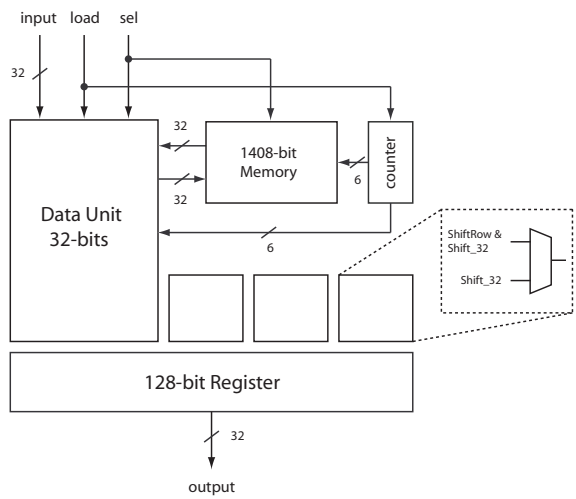

Figure 2.2. Architecture of AES

logic savings the key scheduler is integrated with the round computation by sharing the *ByteSub* component. The top level design is shown in Fig. 2.2. The Data Unit contains the integrated key scheduler and round computation.

2.2 Round Component Optimizations

The state table is comprised of 16 bytes of state table elements $a_{i,j}$, see Fig. 2.3. Each AES component transforms the state table elements using byte operations which are based on arithmetic in the finite field,

$$\mathbb{F}_{2^8} \cong \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1) \tag{2.1}$$

Depending on the AES component these operations may act on the state table rows, columns, or individual table elements.

The *ByteSub* component is comprised of sixteen 8-bit non-linear transformations operating on the state of the cipher. Each table element $a_{i,j}$ is transformed using the map $a_{i,j} \mapsto A(a_{i,j}^{-1}) + b$, where A is a fixed linear transformation, b a fixed 8-bit vector, and $a_{i,j}^{-1}$ denotes multiplicative inversion modulo $x^8 + x^4 + x^3 + x + 1$. The individual 8-bit operations are referred to as AES S-boxes and are the most computationally expensive part of the cipher. For that reason the *ByteSub* component is more expensive than any other AES round component. The size of the *ByteSub* component is dependent on the optimization technique used. The fastest technique uses a large look up table to compute the entire operation. This particular method is quite expensive because it requires a large number of gates to implement. Furthermore it lacks in versatility since it is nearly impossible to pipeline. For a Stratix chipset an 8-bit S-box using look up tables requires 208 logic elements. Given that there are 16 8-bit S-boxes for one round of AES, the table lookup method is not a particularly attractive option. Several authors have implemented the 8-bit S-boxes using

low-order bytes ⇑	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	⇓
	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	
	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	*high-order bytes*
	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	

Figure 2.3. State table for 128-bit AES

memory [5, 10, 11, 13] for FPGA devices, but loses its appeal when transferred to ASICs, since memories utilize large areas. Therefore we do not apply this technique in our design.

There has been considerable improvement in the design of AES S-boxes using composite fields [9]. Composite field techniques provide an alternative representation of the 8-bit field elements shown in equation 2.1. Using a subfield $\mathbb{F}_{2^4}$ we can represent any 8-bit field element by two 4-bit strings. This is equivalent to having an isomorphism ϕ mapping $\mathbb{F}_2[x]/(x^8+x^4+x^3+x+1) \rightarrow \mathbb{F}_{2^4}[y]/p(y)$ for some irreducible polynomial $p(y) = y^2+\alpha y+\lambda$. The benefit of working with this new representation is that multiplicative inversions are much more efficient. The multiplicative inversion of a field element $a_1y + a_0$, $a_1, a_0 \in \mathbb{F}_{2^4}$ is given by,

$$(a_1y + a_0)^{-1} = \frac{a_1}{(a_0^2\lambda + a_0a_1\alpha + a_1^2)} \cdot y + \frac{a_0 + a_1\alpha}{(a_0^2\lambda + a_0a_1\alpha + a_1^2)}. \tag{2.2}$$

The addition and multiplication operations given in equation 2.2 are 4-bit operations over the field $\mathbb{F}_{2^4} \cong \mathbb{F}_2[x]/q(x)$. Using the isomorphism ϕ we can transform an AES S-box to the equivalent computation $(A\phi^{-1})(\phi(a_{i,j})^{-1}) + b$. Choosing appropriate polynomials $p(y)$ and $q(x)$, and an isomorphism ϕ the size of a AES S-box can be minimized using,

$$(\phi^*, p^*(y), q^*(x)) := \arg\min_{\phi,p(y),q(x)} \big(\mathbf{cost}(\phi) + \mathbf{cost}(A\phi^{-1}) + \mathbf{cost}(MultInverse)\big). \tag{2.3}$$

where cost is the number of logic elements needed to implement this operation for a Stratix FPGA. This minimization was performed over the subclass of isomorphisms ϕ, that are also linear transformations. This reduces the composition $A\phi^{-1}$ to a linear transformation T whence the $\mathbf{cost}(A\phi^{-1}) := \mathbf{cost}(T)$. Based on these considerations it can be shown that the irreducible polynomials $p^*(y) = y^2 + y + 9$, $q^*(x) = x^4 + x + 1$ and isomorphism,

$$\phi^* = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.4}$$

minimize equation 2.3. The total number of logic elements needed to perform an AES S-box using composite fields is 62. This is a considerably less than the table look up method which required 208 LEs. In our architecture we implement 4 AES S-boxes in parallel over the columns of the state table. This requires a total of 4 clocks per round.

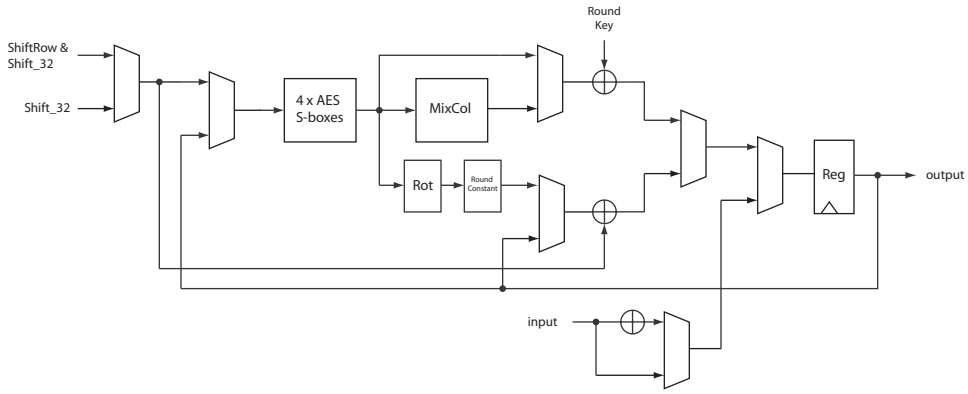

Figure 2.4. AES 32-bit Data Unit

The *ShiftRow* operation shifts the rows of the state table cyclically to the left by either 0, 1, 2, or 3 bytes. The number of bytes a row is shifted is determined by the rows label. The first row is shifted 0 bytes while the last row is shifted 3 bytes. In our implementation the 128-bit register storing the cipher state is multiplexed with *ShiftRow* & *Shift_32* and *Shift_32*. This allows for dynamic selection of the *ShiftRow* operation so that the key schedule can be computed.

The *MixColumn* operation applies a fixed linear transformation to each column of the state table. Each column is independently processed using the invertible linear transformation,

$$\begin{bmatrix} a'_{0,i} \\ a'_{1,i} \\ a'_{2,i} \\ a'_{3,i} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{bmatrix} \tag{2.5}$$

For additional logic savings we replace the multiplication of 3 in the *MixColumn* transformation to $2 + 1$ so that the output byte $a'_{0,i} = 2a_{0,i} + 3a_{1,i} + a_{2,i} + a_{3,i}$ becomes $2a_{0,i} + 2a_{1,i} + a_{1,i} + a_{2,i} + a_{3,i}$. In our architecture a single *MixColumn* transformation is applied every clock directly after the 4 AES S-boxes.

The final component *AddRoundKey* is the bit wise exclusive or of the round key. The round keys are derived using the key scheduler which is driven by the AES symmetric key. The key scheduler algorithm shares the AES S-boxes with the round computation. This allows a further reduction in logic resources. The Data Unit containing the round transformation and integrated key scheduler is shown in Fig. 2.4.

2.3 Performance and Analysis

Our compact implementation of AES encryption is based on a 4 clock per round looped architecture. To help reduce logic area we developed composite field optimizations to the

Table 2.1. AES performance figures for various cores

Design	Device	Area		Max Clock (MHz)	Throughput (Mbps)
		LE	Block RAMs		
Our	Stratix-5	569	1 M4K	71	207
Helion Tech	Stratix-5	630	3 M4K	138	360
Helion Tech	Stratix-5	1023	10 M4K	122	1400
Iguchi et al. [11]	Stratix-5	5142	40 M4K	39	5080
Iguchi et al. [11]	Stratix-5	12560	80 M4K	160	20480

AES S-boxes. All of our optimizations were based on the Stratix FPGA chipset. We compiled our design using Altera's Quartus II 5.0 software with a speed grade set to 5. Table 2.1 provides a performance comparison to other AES architectures contained in the literature. We restricted our comparisons to other Altera devices since they shared a similar logic structure.

Our implementation was the smallest of the AES cores that contained both encryption and key scheduling. As a comparison we listed several high throughput implementations. These provide an idea of the various performance tradeoffs of speed vs. area.

The size of our architecture could have been further reduced by replacing the 4 AES composite field S-boxes with 4 lookup tables. These lookup tables could have been implemented using 2 M4K dual-port RAMs. The size of our design would be reduced to $321 = 569 - 4 * 62$ logic elements and 3 M4K RAM blocks. This design would have required a much larger logic area for an ASIC implementation. The size of an 8-bit AES S-box using table lookups in CMOS7 is 909 equivalent gates. Using our composite field design we are able to reduce the area by more than 40% to 371 gates. This provides considerable savings if multiple AES S-boxes are used. For additional performance the composite field AES S-boxes can be pipelined. This is not true when using table lookups.

Chapter 3

Secure Hash Algorithm (SHA-1)

In 1995, NIST introduced SHA-1 as the secure hash standard. The SHA-1 design is based on the Merkle-Damgård iterative hash structure [7]. The SHA-1 hash transformations were designed for 32-bit platforms and can accept any message input up to 2^{64} bits in length. The output produces a 160-bit output called a message digest.

The architecture for SHA-1 can be broken into two separate stages, one for preprocessing and the other for hash computations. The preprocessing stage prepares the message into a block format by appending padding at the end. Padding bits are added until the new message can be parsed into 512-bit message blocks M_i, $i = 1, \ldots, N$. The second stage takes the message blocks and iteratively applies round mixing operations. Internally these mix operations use five hash values, each 32-bits a piece. At the start of the computation the five hash values are initialized with five 32-bit word constants $H_j^{(0)}$, $j = 0, \ldots, 4$. The first message block and initialized hash values are processed through 80 rounds of mixing and produce a 160-bit output. The output of each mix operation is used to reinitialize the hash values with five new constants. This process continues for each of the N successive message blocks. An illustration of the second stage of SHA-1 is shown in Fig. 3.1.

The SHA-1 mixing operations Round Mix uses two main processes in its computation. The first process called the Message Scheduler expands a message block M_i into 80 32-

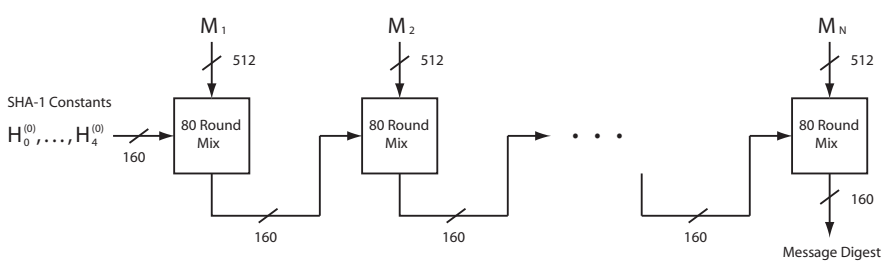

Figure 3.1. SHA-1 Top Level Description

Round Mix$(M_i, H_0^{(i-1)}, \ldots, H_4^{(i-1)})$

Message Scheduler(M_i)
Let $M_i^{(0)} \cdots M_i^{(15)} \leftarrow M_i$
for $t \leftarrow 0$ **to** 15 **do**
  $W_t \leftarrow M_i^{(t)}$
for $t \leftarrow 16$ **to** 79 **do**
  $W_t \leftarrow ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$

Mix$(W_t, H_0^{(i-1)}, \ldots, H_4^{(i-1)})$
Let $(a, b, c, d, e) \leftarrow (H_0^{(i-1)}, H_1^{(i-1)}, H_2^{(i-1)}, H_3^{(i-1)}, H_4^{(i-1)})$
for $t \leftarrow 0$ **to** 79 **do**
  $T \leftarrow ROTL^5(a) + f_t(b, c, d) + e + K_t + W_t$
  $e \leftarrow d$
  $d \leftarrow c$
  $c \leftarrow ROTL^{30}(b)$
  $b \leftarrow a$
  $a \leftarrow T$
return $(H_0^i, H_1^i, H_2^i, H_3^i, H_4^i) \leftarrow (a + H_0^{(i-1)}, b + H_1^{(i-1)}, c + H_2^{(i-1)},$
$d + H_3^{(i-1)}, e + H_4^{(i-1)})$

Figure 3.2. Round Mix operation

bit words W_t, $t = 0, \ldots, 79$. The initialization hash constants $H_j^{(i)}$ and words W_t supply the inputs for the second process, called Mix. The Mix operation uses five 32-bit working variables a, b, c, d, and e. The hash value inputs initialize the working variables at the start of the Mix computation. During each round the working variables are mixed using logic functions, shift rotations, and constant additions. At the end of the eightieth round the working variables are added to the initial hash values to form the output. For more details see Round Mix algorithm in Fig. 3.2.

There are a number of different strategies to consider when developing a SHA-1 implementation. The most commonly used architectures are designed to maximize data throughput. These designs usually implement the five working variables in a parallel fashion [14]. This offers a considerable amount of speed and is well suited for ASIC hardware.

The round structure of the Round Mix operation reveals a great deal of parallelism. The Message Scheduler and Mix processes may be parallelized to shorten the critical path and increase the clock rate. The addition of the constant K_t may also be pushed over into the Message Scheduler process. This balances the work load between the Message Scheduler and Mix processes since they now both operate on five variables. By moving the constant K_t over to the Message Scheduler process a straightforward serial implementation of SHA-1 can be developed. Instead of processing the hash variables in parallel, the variables can now be called from memory and computed serially. This increases the number of clocks per

round from one to five. This feature is employed in our design.

The benefit of going from a parallel to serial design in an ASIC is minimal at best, since the registers are swapped for a memory based configuration. The benefits of a serial implementation for an FPGA on the other hand, are quite noticeable. As we will show a significant reduction in resources can be achieved employing this particular method. Using the configurable logic of an FPGA for storing register variables, is an inefficient use of the logic resources.

3.1 SHA-1 Architecture

Our approach in developing a compact version of SHA-1 for FPGAs remained simple. The goal of our design was to reduce the number of logic resources by effectively using the FPGA technology. We did this by incorporating lookup tables in our development process. This allowed us to manage the size and speed of our design at each stage of our development. When appropriate we also take advantage of the Stratix LE architecture by employing register packing and feedback techniques. Since most FPGAs utilize 4-1 LUTs in their construction our design will provide the same resource benefits as that achieved for the Stratix chipset. We begin our analysis by reviewing the SHA-1 algorithm in more depth. We will focus our attention on the size requirements and implementation features of SHA-1 for FPGA devices.

A large fraction of SHA-1's size can be attributed to the storage of 32-bit variables. The Message Scheduler and Mix operations alone contribute to at least $35 = 19 + 16$ 32-bit variables for computation. If Stratix LEs were the only available storage resource, it would require at least 1120 logic elements to store all these variables, since each LE contains a single 1-bit programmable register. This is not the most efficient use of the LE logic resources since the majority of the combinatorial functionality will remain unused. To utilize the logic more efficiently we propose to store these variables in two M4K 128×32 bit memory blocks, one assigned to both the Message Scheduler and Mix operations. The Mix operation memory stores the initialization constants $H_0^{(0)}, \ldots, H_4^{(0)}, H_0^{(i-1)}, \ldots, H_4^{(i-1)}$ and the working variables a, b, c, d, and e. While the Message Scheduler memory stores the four constants K_t and sixteen of the W_t variables. To efficiently use the memory bandwidth and increase circuit speed the constant variable K_t is be integrated into the Message Scheduler algorithm. The Message Scheduler returns $W_t + K_t$ instead of W_t. This saves us one clock of computation.

For rounds $t \geq 16$ the Message Scheduler takes five clocks to load the 32-bit variables $W_{t-3}, W_{t-8}, W_{t-14}, W_{t-16}$, and K_t. The value $W_t + K_t$ is computed at the end of the fifth clock cycle. In a similar fashion it takes the Mix operation five clocks to load the working variables a, b, c, d, and e to compute $\mathrm{ROTL}^5(a) + f_t(b, c, d) + e$. For increased throughput the Message Scheduler is pipelined to delay the arrival of $W_t + K_t$ to the beginning of the 5th clock of the Mix operation. This allows the new value of a to be computed without delay by combining the Message Scheduler and Mix operation calculations. This gives fives clocks per round for a total of 401 clocks over 80 rounds, with one extra clock accounting for the pipelined delay. When setup and load times are accounted for the Mix Round operation takes a total of 415 clocks. The proposed architecture for our design is shown in Fig. 3.3.

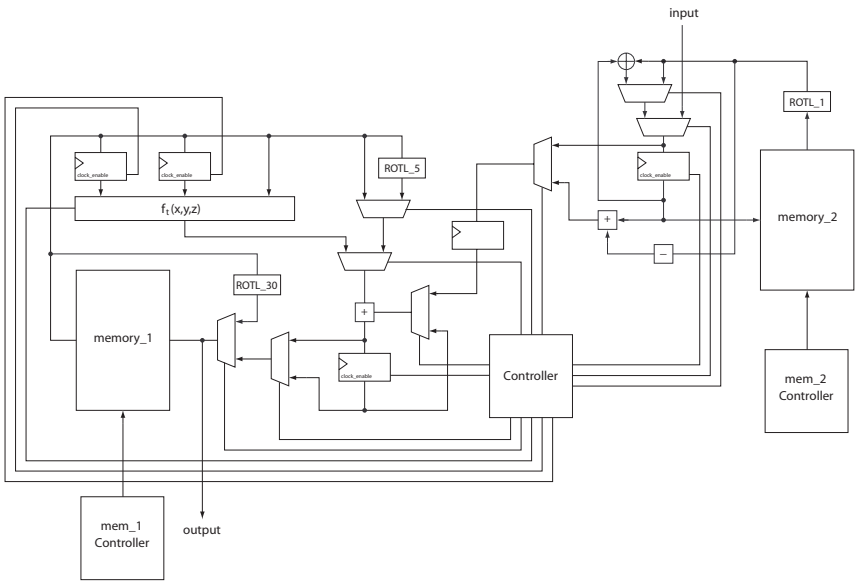

Figure 3.3. Architecture of SHA-1

The memory controller for the mix operation memory uses a cyclic addressing scheme. In SHA-1 the working variables b, d, and e are updated using $e_{new} \leftarrow d_{old}$, $d_{new} \leftarrow c_{old}$, and $b_{new} \leftarrow a_{old}$. The memory controller shifts the labeling of the addresses cyclically by one after each round of computation. Since only a few registers are updated every clock, this conserves power consumption. The main controller in the circuit controls the clock enables for all the registers and mux channel selections. The size of the controllers size were minimized by assigning don't cares to transitions that had no effect on the main computation. This allowed us to find the smallest combinatorial implementation using 4-1 lookup tables.

3.2 Implementation of the Logic Functions

SHA-1 uses three combinatorial logic functions to compute the Round Mix operations. In this section we develop a compact implementation of these logic functions optimized for the Stratix LE architecture. Our design uses register feedback and packing functionalities built into the logic element structure. This helps improve on both speed and logic resource utilization.

The function f_t is a family of three combinatorial functions. The Mix process computation uses the three functions over 20 consecutive rounds a piece, with one function being used twice. The function f_t is defined as follows:

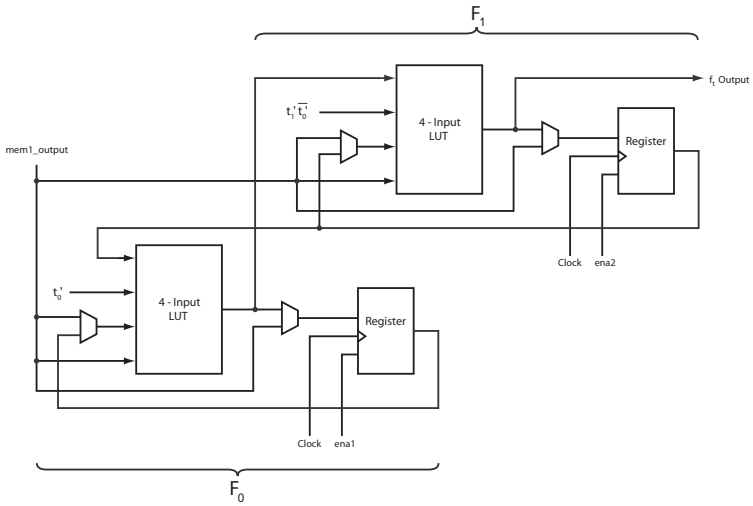

Figure 3.4. Implementation of f_t using Stratix logic elements

$$f_t(x,y,z) := \begin{cases} (xy) \oplus (\overline{x}z) & , \quad 0 \le t \le 19 \\ x \oplus y \oplus z & , \quad 20 \le t \le 39,\ 60 \le t \le 79 \\ (xy) \oplus (xz) \oplus (yz), & \quad 40 \le t \le 59 \end{cases}$$

Using a modification we can decompose the logic function f_t into two smaller functions. In order to best utilize the 4 inputs of a 4-1 LUT, we break the variable t down from 7 to 2 bits. This can be accomplished be defining $t' := (t'_1, t'_0)$ to be $(0,0)$ when $0 \le t \le 19$, $(0,1)$ when $20 \le t \le 39$, $(1,0)$ when $40 \le t \le 59$, and $(1,1)$ when $60 \le t \le 79$. If we take,

$$F_0(x,y,z,t) := \begin{cases} x(y \oplus z), & t = 0 \\ x \oplus y \quad , & t = 1 \end{cases}$$

and

$$F_1(x,y,z,t) := \begin{cases} x \oplus z \quad , & t = 0 \\ x \oplus (yz), & t = 1 \end{cases}$$

we can reconstruct f_t using the composition $F_1(F_0(x,y,z,t'_0), y, z, t'_1\overline{t'_0})$. This construction is shown in Fig. 3.4 using Stratix logic elements. The registers in F_0 are used to store the working variable d when it is loaded from the `mem1_output`. Similarly the registers in F_1 are used to store c. When working variable b arrives, the registers in F_0 and F_1 feed their values back through the 4-1 lookup tables to compute f_t. This construction uses only 64 logic elements, which is the minimum for a Stratix chipset.

Table 3.1. SHA-1 performance figures for various cores

Design	Device	Area		Max Clock (MHz)	Throughput (Mbps)
		LE	Block RAMs		
Our	Stratix-5	401	2 M4K	159	197
Aldec Inc.	APEX20K-1	1413	-	64	404
Alma Tech.	APEX20KE	1329	-	64	404
Cast	Stratix-5	1621	-	107	676
Ning et al. [14]	ACEX-1	1622	-	43	269

3.3 Performance and Analysis

There is surprisingly little information contained in the literature on compact implementations of SHA-1 in FPGAs. The designs that we encountered were optimized for maximum throughput, and take only one clock cycle per round . While fast, this design strategy is not the best for utilizing the FPGA resources efficiently. This significantly wastes combinatorial logic, since most of the logic elements are used in storing registered variables only. In our approach we applied an iterative loop architecture to both the Message Scheduler and Mix processes. This saves a considerable amount of logic, since most of the registered variables could be pushed into memory. Furthermore we optimized our algorithms around the logic element structure which helped reduce size and increase throughput.

In our analysis we conducted a comparison of several SHA-1 cores. We restricted ourselves to the Stratix, APEX, and ACEX device families, since they all shared a similar logic element structure. The APEX and ACEX logic elements do not incorporate register feedback and use an older manufacturing process, therefore an implementation on these devices will be slightly larger and slower. We compiled our design using Altera's Quartus II 5.0 software with a speed grade set to 5. To ensure design accuracy the Quartus chip editor was used as a comparison.

Among the compared architectures, our design had the smallest resource utilization. We were able to reduce LE resources by as much as 72%. Our data throughput was slower than the other designs, since our Round Mix operation takes five clock cycles rather than one. The performance figures for all the implementations are shown in Table 3.1.

In addition to the above comparisons we also compared the *Throughput/DesignCost* ratio for each of the above designs. Assigning cost metrics C_{LE} and C_{M4K} to LEs and M4K RAMs, we can compute the resource cost of a design using the formula,

$$DesignCost := (\#LEs) \cdot C_{LE} + (\#M4K) \cdot C_{M4K}.$$

Out of all the designs our SHA-1 architecture showed the best *Throughput/DesignCost* ratio whenever $C_{M4K} < 35 \cdot C_{LE}$. Considering that a Stratix EP1S40 device contains 183 M4K RAM blocks, this condition should hold true for many designs.

Chapter 4

Concluding Remarks

We have shown new design strategies applicable to AES and SHA-1. The resulting circuits have an extremely small hardware footprint yet do not overly sacrifice in terms of speed. The developed, optimized S-box composite field technologies have led to the smallest Stratix FPGA implementation for AES encryption with integrated key scheduler. The same technology applied to ASIC technology reduces an AES S-box from 909 gates using table lookups to 371 gates in CMOS7.

Our SHA-1 architecture is optimized for the Stratix FPGA chipset. We employed register packing and feedback functionalities for optimal performance and size. This reduces logic element usage by as much as 72% as compared to other highly optimized designs found in the literature. Our architecture provides the best Throughput/DesignCost ratio over any design found in the literature.

References

[1] M. Aigner, S. Dominikus, S. Mangard, "A Highly Regular and Scalable AES Hardware Architecture", *IEEE Trans. on Computers*, 52(4), 483-491, April 2003.

[2] Aldec Inc., `http://www.aldec.com/products/ipcores/Configurations/`.

[3] Alma Tech., "SHA-1 Cores", `http://www.alma-tech.com`.

[4] Cast, "Secure Hash Algorithm Cryptoprocessor Megafunction", `www.cast-inc.com/cores/sha-1/cast_sha-1-a.pdf`.

[5] P. Chodowiec, K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm", *CHES 2003*, Vol. 2779, 319-333, Sept. 2003.

[6] J. Daemen, V. Rijmen, "AES Proposal: Rijndael", NIST, `http://www.nist.gov/aes`.

[7] I. Damgård, "A Design Principle for Hash Functions", *CRYPTO 1989* LNCS 435, 416-427, 1989.

[8] S. Dominikus, M. Feldhofer, J. Wolkerstorfer, "Strong Authentication for RFID Systems using the AES Algorithm", *CHES 2004*, Vol. 3156, 357-370, Aug. 2004.

[9] P. K. Dubey, C. S. Jutla, V. Kumar, J. Rao, P. Rohatgi, A. Rudra, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic", *CHES 2001*, Vol. 2162, 171-184, 2001.

[10] Helion Technology, "High Performance AES (Rijndael) Cores for Altera FPGA", `http://www.heliontech.com/core2.htm`.

[11] Y. Iguchi, T. Sasao, H. Qin, "An FPGA Design of AES Encryption Circuit with 128-bit Keys", *GLSVLSI 2005*, 147-151, April, 2005.

[12] H. Kuo, P. Schaumont, I. Verbauwhede, "Design and Performance Testing of a 2.29 Gb/s Rijndael Processor", *IEEE Journal of Solid-State Circuits*, 569-572, March 2003.

[13] J. D. Legat, J. J. Quisquater, G. Rouvroy, F. X. Standaert, "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of AES Rijndael Very Well Suited for Small Embedded Applications", *ITCC 2004*, Vol. 2, 583-587, April 2004.

[14] Z. Ning, D. Zibin, "FPGA Implementation of SHA-1 Algorithm", *5th International Conference on ASIC*, Vol. 2, 1321-1324, Oct. 2003.

[15] N. Pramstaller, J. Wolkerstorfer, "An Efficient AES Implementation for Reconfigurable Devices", *Austrochip 2003*, 5-8, Oct. 2003.

[16] R. Rivest, "The MD4 Message Digest Algorithm", RFC 1320, April 1992.

[17] A. Satoh, S. Morioka, S. Munetoh, K. Takano, "A Compact Rijndael Hardware Architecture with S-Box Optimization", *ASIACRYPT 2001*, Vol. 2248, 239-254, 2001.

[18] Altera Corporation. Stratix Device Handbook, Volume 1. Altera Corporation, San Jose, California, 2005.

DISTRIBUTION:

2 MS 0672
W. E. Anderson, 5614

2 MS 0672
T. J. Draelos, 5614

2 MS 1072
R. D. Miller, 017351

2 MS 0672
R. Schroeppel, 5614

2 MS 0672
M. Torgerson, 5614

1 MS 0672
T. McDonald, 5614

1 MS 0671
R. E. Trellue, 5610

1 MS 0455
S. G. Varnado, 5600

1 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 9616

1 MS 0123
LDRD Program Office, Attn:
Donna Chavez