AES Implementations Optimized for Mid-Range FPGAs

Bryan M. Sobczyk

Abstract— **The Rijndael Algorithm was chosen for the Advanced Encryption Standard (AES) in 2001 and formally published in FIPS Publication 197. Since Rijndael was released as a candidate a number of cores were created to test and benchmark the algorithm in both hardware and software. Rijndael was chosen partly based on its ability to be efficiently implemented in Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (AISCs). In AISC design, heavy use of combinational logic is advantageous. In FPGA designs each logic cell has local memory available and all free logic cells are equally valuable for design use. A survey of published AES architectures found they did not fully take advantage of ROM blocks to simplify and shorten critical paths in the algorithm's rounds. This paper will present a T-box design that will utilize FPGA memory in a core with a standard 32-bit bus width that will sustain a throughput of 20 Mbyte/sec.**

***Index Terms*—Advanced Encryption Standard, AES, Tbox, Cryptography, AES-128, AES-192, AES-256.**

I. AES Overview

THE Advanced Encryption Standard (AES) specification is documented in the National Institute of Standards and Technology's (NIST) FIPS 197 publication.[5] J. Daemen and V. Rijmen submitted Rijndael as part of NIST's AES contest. Candidates for the contest were tested based on strength of the algorithm against attacks, maximum throughput, and resources required for both software and hardware implementations. Rijndael was originally designed with a variety of key lengths and variable block lengths in mind. When the variable block length requirement was dropped, Rijndael was amended to a fixed 128-bit block length. Since the chosen core would be a US Federal Standard all teams participating had to openly publish their standard and must be free of Intellectual Property. The finalists were evaluated equally, but each submission differed in implementation costs, throughput, and versatility in implementation. Flexible algorithms that could run efficiently across Application Specific Integrated Circuits (ASICs) for smart cards, 32-bit microprocessors, and even 8 bit microcontrollers proved a challenge during final selection. [11] On October 2nd, 2000 NIST announced that Rijndael was the winner and new AES standard, based on the evaluation criteria, peer review, and excellent performance across a number of target platforms.

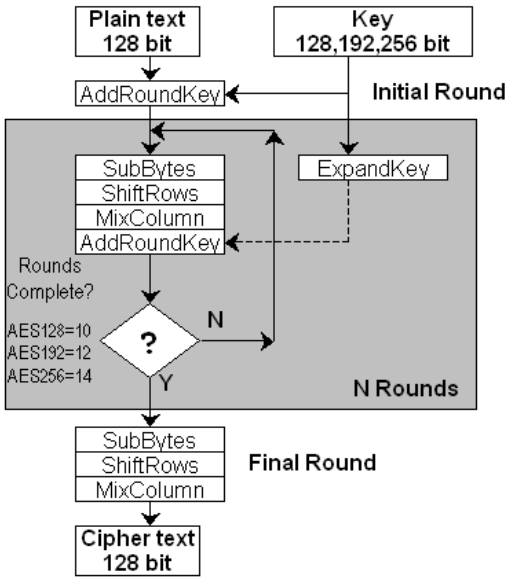

Figure 1: AES Encryption Round Structure

AES supports multiple key sizes (128,192, and 256 bits) and is used as a block cipher with a message size of 128 bits. The block cipher structure can be used in a variety of modes to create a secure stream cipher based on AES encryption and/or decryption. Using the basic Electronic Code Book (ECB) mode, a 128-bit message is encrypted with a key of 128, 192, or 256 bits to produce a 128-bit cipher text, as shown in Figure 1. A key expansion is first performed on the initial key values, based on a key schedule, to generate unique keys for all rounds of encryption. The key schedule was developed to use small amounts of memory, have no symmetries, have efficient diffusion of keys, and be non-linear. [11] Diffusion allows small changes in a previous key to cause significant changes in the next expanded key. Elimination of symmetries and linear functions allows generation of expanded keys that resist attacks and analysis on the cipher text. A perfect key expansion would generate seemingly random keys that are unique and easily computed from the initial and subsequent expanded keys. With no pattern to attack, the attacker would have to pick from all possible keys for every round of the algorithm. AES uses rotations, XOR operations for permutations, and table lookups from an Sbox table specified in FIPS 197 for direct substitutions on each byte of the key.

To further mix each key in the schedule a RCON value, a unique constant determined by the current key being generated, is added to break patterns in the key. On every new round of four, six, or eight key values an incrementing RCON value is XORed with the key to eliminate symmetries in the expanded keys.[5] An additional change is added for 256 bit keys; every fourth key undergoes a substitution to keep the same transform being applied over more than three consecutive expanded keys. The RCON values are listed in FIPS 197 along with example key expansions for 128, 192, and 256-bit key lengths.

Based on the round structure chosen, there are a number of options for implementing the key schedule. If each round is calculated iteratively, it is easy to calculate the expanded key on the fly. This can be accomplished by expanding only the current key and retaining the previous key values required to calculate the next key. To implement AES with a 128-bit key, only four previous values are required to calculate the keys on the fly. A full key expansion for all modes, some times referred to as a 3 in 1 design, requires up to eight previous values to be retained. If a full round is calculated for each cycle four keys must be simultaneously calculated and up to eight values need to be retained for all three key sizes. A potential drawback to on the fly calculation is the worst-case delay of looking up a value in a ROM table, two bit-wise XORs, and the gate delay of two muxes. The full round in a cycle approach can provide very high throughputs (128 bits computed every cycle) but requires fast key expansion or the entire key schedule to be pre-computed and stored. Storing the entire key schedule requires up to sixty 32-bit words to be stored for the 256-bit mode but allows the keys to be referenced in a single clock cycle if fast RAM blocks are used. Most modes having the pre-computed key values can speed the calculation of the rounds. If a mode with a changing key is used the benefits of pre-computation are diminished compared to the memory resources required. For memory limited applications, on the fly calculation provides fast key expansion without RAM blocks.

A survey of other papers has found both preprocessed and on the fly key expansion used in high throughput designs. McCloone and McCanney utilized preprocessing of the key with a LUT based Tbox implementation discussed later in this paper. [15] Qing et al., Wang and Ni, Wang et al., Lofty et al., Rizk et al., and Standaert et al. had all generated their keys on the fly to save on processing time and hardware resources.[29, 25, 26,14, 17, 23] Two on the fly key scheduling units are chained to provide Schaumont et al. Rijndael processor capable of 2.29 Gbit/sec throughput. [20] For a fully pipelined high-speed core the keys are preprocessed to allow the rounds to compute as soon as possible with no calculation delays, achieving a throughput of 30-70 Gbits/sec.[9] Lin and Huang pipelined the computation of the key to matched their pipelined round structure. [13] The timesavings in pre-expanding the key in the pipelined and single cycle architectures justified the use of more memory resources to store the keys.

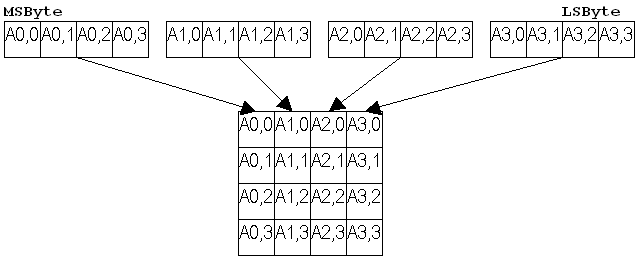

Figure 2: AES State Array

The AES standard is round based and operates on a thirty-two bit by thirty-two bit State array. The array is divided into sixteen bytes as shown in Figure 2. It should be noted that the indexes into the array are row then column. The indexes correspond to the byte sequence; every four bytes form a new row. The state is then taken through 10, 12, or 14 rounds for 128, 192, or 256-bit keys respectively. All operations are performed in the Galois Field GF(2^8). A full explanation of Galois Fields is presented in Stallings' textbook but the basic arithmetic operations can be explained briefly. [21] The equivalent of addition and subtraction in the field is XORing the two values. Multiplication and division involve a more complex algorithm but a simple table lookup can perform the operations required for AES. As suggested by Daemen and Rijmen in [11], a table of all 256 values multiplied by 0x02 is listed as the xtime table. Any multiplied value can be decomposed into a sum of powers of 0x02. For example 0x03*X can be expressed as X*(0x01 XOR 0x02), which can be calculated as (X XOR xtime(X)). This technique will be utilized later to pre-calculate the tBox values for our architecture.

The plaintext message is loaded into the State array of Figure 2 and the AddRoundKey operation is performed. During this initial state the message is XORed with the initial key. The new State array is taken through a series of identical rounds stated in the FIPS 197 standard including SubBytes, ShiftRows, MixColumns, and AddRoundKey. The rounds are designed to be invertible for decryption while being simple to insure delays are low since rounds are repeated at least ten times. SubBytes is a non-linear substitution of bytes directly substituted from the sBox array. Details of the sBox's construction are detailed in [11], where the authors provided the basis for choosing values. The sBox is mathematically proven to have both a transform for encryption and an inverse transform used for decryption. ShiftRows provides a diffusion of values within the State array through simple shifts. MixColumns provides a linear permutation of the State array on a byte-by-byte basis, while being easily reversed by an inverse permutation. Daemon and Rijmen have stated in [11] that the performance of this step on 8-bit processors was a driving factor in the choice of permutation. The final step, AddRoundKey, performs a bit-wise XOR with each 32-bit column of the State array with one 32-bit word of the expanded key. The number of rounds chosen by Daemen and

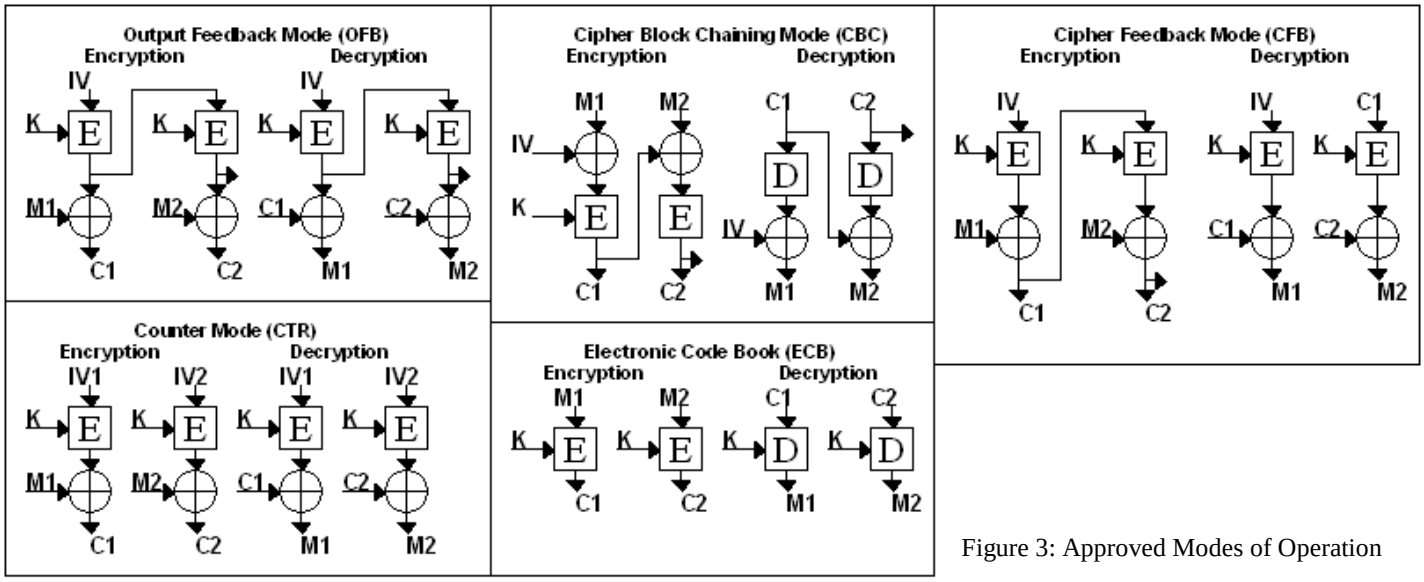

Figure 3: Approved Modes of Operation

Rijmen are based on known cryptanalysis attacks and diffusion criteria that require two rounds to fully diffuse single bit changes. The foreknowledge of how attackers could exploit the algorithm have lead to logical choices in each step of Rijndael, and hence AES, to make known attacks computationally infeasible, while preventing easily exploited patterns in the algorithm from developing. [11 & 21]

After the specified number of rounds described above is done, the final round only includes SubBytes, ShiftRow, and AddRoundKey steps. The resulting State array is the cipher text and is output as a 128-bit block. Decryption is performed using the same round structure and sequences of operations, but inverse tables are used for each step. The expanded keys are generated in the same way, but the groups of four sequential 32-bit expanded keys are used in reverse order.

II. Encryption Modes

Use of low area and high throughput encryption cores with an approved mode can match or exceed the throughput of ECB mode encryption and decryption. The use of approved modes can prevent identical plaintext blocks from having identical cipher text blocks. NIST has listed the following approved confidentiality modes in their publications:

- NIST SP 800-38A
  - Electronic Code Book (ECB)
  - Cipher Block Chaining (CBC)
  - Cipher Feed Back (CFB)
  - Output Feed Back (OFB)
  - Counter (CTR)
- NIST SP 800-38B
  - Cipher-based Message Authentication Code (CMAC)
- NIST SP 800-38C
  - Counter with Cipher Block Chaining-Message Authentication Code (CCM)
- NIST SP 800-38D
  - Galois/Counter Mode (GCM)
  - Galois Message Authentication Code (GMAC)

If only authentication is required, SP 800-38B specifies the use of Cipher-based Message Authentication Code Mode (CMAC). For both authentication and confidentiality, SP 800-38C specifies Counter with Cipher Block Chaining – Message Authentication Code (CCM). For high throughput authentication with confidentiality Galois/Counter Mode (GCM) is specified by SP 800-38D, but strict adherence to the recommended Initialization Vectors is required to satisfy the uniqueness requirement that provides the high degree of security.

The choice of the listed modes provides stronger security than the basic ECB mode but has consequences for the operation of the cipher. The simple Counter (CTR) mode provides a fast streaming cipher that only requires block encryption. Each message is XORed with the counter value to produce a block of cipher text. CTR mode encryption and decryption can be done in parallel, is easily pipelined, and allows changes to individual blocks without affecting other blocks. The Initial Vector (IV) can be provided by a simple counter or another unique but changing value like a memory address. It should be noted the counter width should produce enough unique values to prevent repeats or spread out repeats so 2^{256}-1 unique values get used before repeats occur between the 128-bit blocks used with AES. If a single bit error is encountered the message CTR will cause a single bit error in the cipher text. CTR provides a simple way to use an optimized encryption core for both stream encryption and decryption. A subset of CTR can utilize only a portion of the counter's bits, but it should be noted that only the same portion of the message could be encrypted with this method. Utilizing a larger AES key will not overcome this limitation since only a portion of the 128-bit State array can be used. This is regardless of the key size used.

OFB mode chains the resulting encryption from the previous block as input to the next block's encryption block. The resulting cipher text is XORed with the message to provide the cipher text. Since each block encryption is dependent on the previous result, it is not possible to pipeline or use parallelism to speed up the calculation. OFB does not depend on a sequence of Initial Vectors as CTR mode does. The first IV value provides the seed value used for all subsequent blocks. Again, the same encryption module can perform encryption and decryption. OFB can provide security for noisy channels, since small bit errors do not propagate through other blocks.

CFB mode differs from OFB in that the feedback is taken from the previous cipher text and not the encryption step. The drawback to this approach is that errors can propagate between each subsequent block. CBC mode is unique from previous modes since an IV is XORed with the message before encryption/decryption is performed. Unlike the other modes encryption and decryption modules are required but errors affect the current and next blocks of data.

A summary of modes is depicted in Figure 3. Of the available modes, this paper's architecture makes use of Counter mode for increased strength over ECB mode. A simple modification can allow OFB to be implemented due to its similar structure to CTR mode. These modes allow the encryption module to be utilized for both block encryption and decryption without requiring extra tables to be stored.

III. TBOX IMPLEMENTATION

Since the publication of Rijndael and the final AES specification a number of architectures to improve performance have been proposed and tested. One method to increase throughput without resorting to pipelining was to shorten the rounds to a single cycle. The tBox table approach was suggested and derived by Daemen and Rijmen and utilized by a number of researchers to improve performance. [11] An excellent explanation and datapath example is derived and explained by Gaj and Chodowiec in a chapter of Cryptographic Engineering. [7] The tBox structure combines the SubBytes, ShiftRows, and MixColumns steps utilizing linear algebra rules, in the GF(2^8), creating a 256 value table of thirty two bit words called a tBox. Every standard eight-bit sBox lookup in SubBytes can now be looked up in the tBox, which returns a 32-bit result.

As stated in [11] and [7], the transformations for one full round of AES are listed below:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \underbrace{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}}_{\text{MixColumns}} \underbrace{\begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j+1}] \\ S[a_{2,j+2}] \\ S[a_{3,j+3}] \end{bmatrix}}_{\text{ShiftRows (addition mod 4)}} \otimes \underbrace{\begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}}_{\text{AddRoundKey}}$$

SubBytes

Using the rules and properties of matrix math in GF(2^8):

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S[a_{0,j}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[a_{1,j+1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[a_{2,j+2}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[a_{3,j+3}] \begin{bmatrix} 01 \\ 01 \\ 02 \\ 03 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

The equation above can be implemented by tables that include the values of the Sbox multiplied by the 4x1 matrices, which are now represented as Tboxes below:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = T_0[a_{0,j}] \oplus T_1[a_{1,j+1}] \oplus T_2[a_{2,j+2}] \oplus T_3[a_{3,j+3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

Each Tbox table uses an eight-bit index into a 256-element array of 32-bit values. Storing these tables takes 8Kbits of ROM per table, for a total of 32Kbits of memory. A reduction in the number and size of the tables can take advantage of two FPGA strengths. First, the FPGA can use wires to copy or shift values at no cost in resources. This advantage is not possible with general-purpose processors that require separate processing cycles to shift values. Second, the four tables are built on the same eight-bit values that are shifted to form the other tables. It is possible to use a 256-item table of 24-bit words and shift it a total of three times to get al.l four Tbox table values. To save memory only one tBox could be used to give a four-time reduction in memory requirements. The down side is the single table would cause a four-fold increase in processing time. For this paper a trade off was made to use four tables from the same Tbox initial values, and shift them as necessary. This allows 32-bits of the State array to be computed every cycle and stored in the next State's registers.

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \underbrace{T_0[a_{0,j}]}_{\text{SameTbox0}} \oplus \underbrace{Shift_1(T_0[a_{1,j+1}])}_{\text{SameTbox0}} \oplus \underbrace{Shift_2(T_0[a_{2,j+2}])}_{\text{SameTbox0}} \oplus \underbrace{Shift_3(T_0[a_{3,j+3}])}_{\text{SameTbox0}} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

At the cost of memory and logic element resources, it is possible to build a high throughput core that computes the four instances of the above tBox equations in a single cycle. In this architecture a maximum of fourteen cycles would be needed to complete the 128-bit cipher text from a given plaintext already loaded at high speeds. Gaj and Chodowiec present an efficient version of this architecture that can be easily pipelined. [8] A similar decryption architecture, using inverse tables, can be used in the same way but must be supplied keys in the correct order to perform decryption. Wang et al., McLoone and McCanny, and Rouvroy et al. have published architectures similar or identical to the tBox architecture as presented by Gaj and Chodowiec.[26, 15, 22] Of specific interest to mid-range FPGA implementations is the approach the UCL Crypto Group took to use RAM blocks, shift registers, and short data paths to create a core that can encrypt and decrypt at 208 Mbit/sec on Spartan II devices. This was achieved with only 163 slices and 3 RAM blocks used. [22] The numbers compare favorably with a number of designs optimized for use on the larger and more capable Virtex FPGAs, while not using a majority of the device's resources. This paper's architecture will be compared to the UCL group's efficient core in the comparison section.

It should be further noted that tBox implementations do not always meet or exceed the throughput of more conventional AES implementations optimized for speed and/or area. The tBox method is not well suited to ASIC design due to the large amount of memory required. Memory can be laid out very efficiently in Very Large Scale Integration (VLSI) processes but consumes a large area of the die compared to a sea-of-gates implementation of combinational logic functions. The UCL Crypto Group Core on a Spartan 3 FPGA achieves 208 Mbit/sec with 163 slices used. McLoone et al. achieved 6956 Mbit/sec using 2222 slices but required a Virtex FPGA. Huang et al. bested the UCL Group by not using tBox architecture and achieved 647 Mbit/sec at a cost of 148 slices and 11 Block RAMs of a Spartan 3 device.

Software implantations would not benefit from the same techniques described above since software cannot make use of efficient shifts, cross wiring, and fast logic operations. The sequential nature of all modern general-purpose processors makes all operations equally costly in terms of delay. The basic algorithm can be optimized for specific processors in order to utilize matrix instructions, parallel processing via multiple cores, and/or the use of efficient compilers to minimize the assembly instructions required. Bernstein and Schwabe have outlined fast AES techniques that use each processor unique structure such as 64 bit operations, verses 32 bit calculations, masked tables, and efficient use of caches to shorten the time required to compute all rounds of AES. [30,31] In a method similar to the matrix math performed for Tboxes, another manipulation of the matrix is possible to effectively compute across the rows of the state to save extra corrective shifts. Bertoni et al. have saved rotations and extra memory lookups to speed up the standard software implementation used for embedded processors. [30] The effect

Sampling of the Altera Cyclone II Family

	EP2C20	EP2C35	EP2C50	EP2C70
Logic Elements (LE)	18,752	33,216	50,528	68,416
M4K RAM (4 Kbit RAM)	52	105	129	250
Minimum User I/O Pins	142	322	294	422
Smallest Leaded Package	240-PQFP	NA	NA	NA
Smallest Package	256-BGA	484-BGA	484-BGA	672-BGA

Sampling of the Xilinx Spartan 3 Family

	XC3S1500	XC3S2000	XC3S4000	XC3S5000
CLBs (1 CLB=4 Slices)	3,328	5,120	6,912	8,320
Distributed RAM Bits	208K	320K	432K	520K
Block RAM	576K	720K	1728K	1872K
Minimum User I/O Pins	221	333	489	489
Smallest Leaded Package	NA	NA	NA	NA
Smallest Package	FG320 BGA	FG456 BGA	FG676 BGA	FG676 BGA

Table 1: Sampling of Mid-Range FPGA Parts

of Graphics Processing Units (GPUs) and Single Instruction, Multiple Data (SIMD) instructions that are designed for array-based calculations may provide efficient matrix transformations that can use methods like the tBox to obtain faster software implementations. Further Discussion of software implantations are beyond the scope of this paper and provide an open field for further research.

IV. FPGA Cell Architecture

This paper aims for a Field Programmable Gate Array (FPGA) implementation of AES and warrants an investigation into the architecture of both the Xilinx and Altera families. The Cyclone II and Spartan 3 part families are equivalent low cost, mid-range small FPGAs (a sample of which is presented in Table 1). These devices do not have the resources or speed of a Virtex or Stratix device but they do provide greater performance than the older APEX/MAXII and XC/CoolRunner parts.

Altera's Cyclone II/III families are composed of Logic Array Blocks (LABs). Each LAB contains sixteen Logic Elements (LE). Dual ported RAM is also available via M4K RAM blocks with high clock rates (up to 260 Mhz). Each LE provides a four input Look Up Table (LUT), a register, carry chain logic, and connection muxes. Unlike the Xilinx equivalents of LE blocks, called slices, Altera elected not to add extra discrete logic elements within the cells. All functions are implemented via LUTs. This gap in implementation paths is offset by Altera's LE modes; a normal mode allows arbitrary functions while arithmetic mode configures the LE for fast and compact arithmetic functions. Using both the primary inputs and carry-in logic a six input function can be implemented in each LE. For comparison, Xilinx's slices can implement only four input functions in each slice.

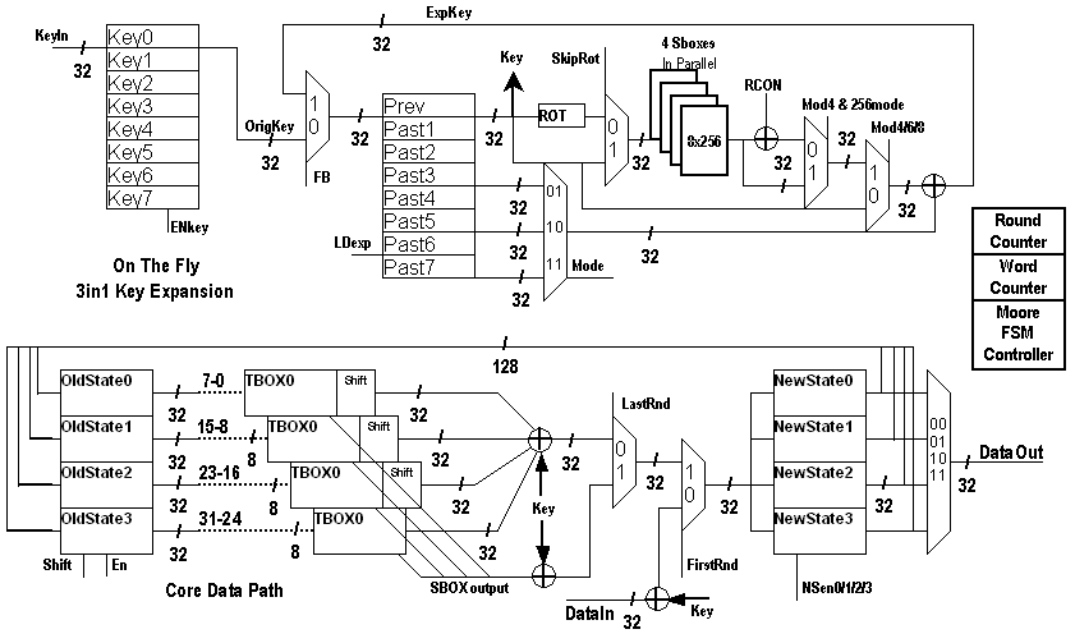

Figure 4: Our Architecture

Xilinx's Spartan 3 family is built on Configurable Logic Blocks (CLBs) that contain four slices each. Every slice contains two logic function tables, two registers, carry logic, and individual logic gates. A number of slices provide shift registers and distributed RAM as special functions. Better use of the cells provides high resource utilization, and allows large and complex designs to fit in smaller devices. The function tables can be used as Look Up Tables (LUTs) that can implement any four-input function as stored values. Two four-input tables can be stored in each slice, and the output can be registered within the slice. Many adders and multipliers can be implemented to fit within the cell, and make use of the carry logic and spare gates for very high slice utilization. Spartan's block RAMs offer 18Kbits of fast, dual-ported, synchronous storage. For 32-bit numbers, 512 elements can be stored in one 18K block. As outlined in the Xilinx Spartan 3 User Guide, each dual ported RAM can also be used for 128 states with up to 36 outputs in a single block. Use of the SRL16 shift register blocks can greatly lower device utilization if a reset is not required. SRL16 blocks configure the LUTs as a shift register, but do not use the slice's flip-flops. The built-in Mux can select either of the slice's LUTs to configure longer shift registers at low hardware cost.

Each manufacturer has gone to great lengths to point out the similarities between underlying hardware and promote any difference in performance. Altera claims their faster speed grades produce faster customer implementations, and Xilinx promotes the special configuration modes like SRL16 registers and the fast carry logic as leading to better, high performance designs. Xilinx has acknowledged that newer versions of the Quartus design suite from Altera have higher performance than earlier implementations. Due to Altera's table centric cells, the software's optimization and fitting of logic is very important to high throughput designs. Xilinx has a similar push for their ISE tools to recognize logical parts of designs that can fit into each cell and still recognize special cases like the SRL16 register. Ultimately, only real life testing of designs across devices, vendors, and software tools can prove the advantages of each manufacturer's product line.

V. Our Architecture

The architecture described in this paper is based on the tBox implementation but targets smaller CPLD families like Altera's Cyclone II/III and Xilinx's Spartan III. The specification for the core shown in Figure 4 included a number of items that were included for interfacing and re-utilizing costly resources and not for performance. The specification included:

- Throughput of at least 20Mbyte/second (160 Mbit/second) through the core
- 32-bit Input/Output registers utilizing interrupt-like triggering
- tBox-based round architecture to achieve single cycle calculations inside the rounds
- Reuse of most registers, minimal use of RAM
- Use of only standard IEEE compatible VHDL code to insure code can be ported across vendor tools
- Register-Transfer-Level (RTL) style coding to avoid excess hardware creation
- No vendor specific libraries, macros, or Intellectual Property (IP) cores
- Modular bottom-up construction to aid in testing and reuse
- Separation of data path and controllers to keep control hardware apart from the data path.

Preliminary Tbox Resources

	Xilinx ISE 9.1 Webpack			Altera Quartus II Web Ed.	
Table Sizes	Slices	BRAMs	Max CLK freq.	Logic Elements	Max CLK freq.
24x256	18	1	134 MHz	547	205 MHz
3x8x256	18	3	139 MHz	547	205 MHz
4x24x256	74	2	90.6 MHz	2156	166 MHz
4x3x8x256	74	6	97.2 MHz	2156	166 MHz

Table 2: Tbox Resource Tests

- Strict use of only STD_LOGIC and STD_LOGIC_VECTORS to insure no type conversions are necessary
- Simple external control of the core for I/O, mode selection (AES128/192/256), and status

The tBoxes used in this paper were computed manually utilizing the xtime tables provided by Daemen and Rijmen. The results were copied to a VHDL description of a ROM. [11] It should be noted that McLoone and McCanny published three tables used to create the tBoxes. An Sbox, Sbox * 02, and Sbox * 03 table were presented to form all four tBoxes as necessary. A comparison of this paper's tables with the tables published by McLoone and McCanny turned up five discrepancies:

- sBox * 02 table for B3 & B4 are reversed from the order of this project's tables (CA CF instead of CF CA)

Table 2: Sampling of Mid-Range FPGA Parts

25 instead of 25 AF)
- sBox * 03 value for 56 contains a typo (E0 instead of E8)

Examination of the specific terms suggest the swapped terms in the Sbox * 02 table were continued to the Sbox * 03 table, since the correct values were listed. The E0/E8 type looks like a simple typing mistake. The remainder of the tables successfully confirmed the table data in this paper's implementation when NIST test vectors were run and verified.

Original planning for the tables included only one tBox selected by multiple XORs, but it was realized that having only one tBox allowed only a single look up per clock cycle. This would require multiple clock cycles per round. During key expansion and final rounds the need for the sBox values would also take a performance hit since only one eight bit value can be looked up at a time. It was decided to use the same 24-bit wide table four times to allow single cycle rounds and simultaneous 32-bit sBox lookups.

In order to better utilize each Slice or Logic Element an experiment was conducted. It was suggested that the synthesis tools would use memory resources better if the tables were not 256 items x 24-bit words but left as 256 items x 8 bit tables. Both the Xilinx and Altera synthesis tools were given a two-

All devices are listed as USB 2.0 Compatible

		Drive Under Test			
	Test	Thumb Drive 1	Thumb Drive 2	Laptop Drive	Desktop Drive
FlashMemToolkit	Read MIN MB/s	0.8 MB/s	6.3 MB/s	cannot test	cannot test
	Read MAX MB/s	0.9 MB/s	7.5 MB/s	cannot test	cannot test
	Read AVG MB/s	0.9 MB/s	7.4 MB/s	cannot test	cannot test
HDTUNE 2.55	Min MB/sec	cannot test	cannot test	19.4	13.3
	Max MB/sec	cannot test	cannot test	21.7	30
	AVG Mb/s	cannot test	cannot test	20.3	27.8
	Burst Rate	cannot test	cannot test	17.5 MB/s	23.8 MB/s
CrystalDiskMark 2.2	Seq Read	cannot test	7.84 MB/s	23.86 MB/s	28.44 MB/s
	Seq Write	cannot test	2.44 MB/s	20.06 MB/s	24.38 MB/s
	512K Read	cannot test	7.763 MB/s	17.15 MB/s	16.54 MB/s
	512K Write	cannot test	1.35 MB/s	19.06 MB/s	16.06 MB/s
	4k Read	cannot test	3.728 MB/s	0.529 MB/s	0.346 MB/s
	4k Write	cannot test	0.024 MB/s	1.082 MB/s	0.864 MB/s

Table 3: Disk Drive Throughput Tests

register bank and one table design. One used a 24 bit x 256 word ROM and another using three separate 8 bit x 256 word ROM tables. No code was added to specify the ROM should be implemented as chained Lookup Tables (LUTs) or static RAM blocks. Xilinx ISE Webpack utilized BRAM blocks, achieving a final clock rate of 134 MHz. Splitting up the tables utilized 3 BRAM blocks with a slightly higher clock rate of 139 MHz. Altera's Quartus II Web Edition was set to use RAM and ROM for any size table, but when optimizing for speed implemented Logic Element LUTs were used instead of RAM blocks. Quartus reported a final clock rate of 205 MHz. A second test with four instances of the first test was run and summarized in Table 2. Based on the findings the individual tables were used to provide a small increase in speed for the Xilinx devices.

Key Expansion was originally going to be pre-computed and stored in a RAM block until required. This design requires both a counter and a large 32-bit x 60 item RAM block. The requirement for a design that can work for all three modes of AES (128/192/256 bit), without reloading the device required a simple but flexible design. A straightforward design with 32-bit x 4 word register for the key and a maximum of 32-bit x 8 word temporary register bank allows enough storage for the largest 256 bit key size. A small table and a counter that advances on every count that equals zero modulo four, six, or eight, respectively, supply the RCON constant. As stated before, four separate sBox lookups are performed simultaneously to insure the next key is calculated in one cycle.

It should be noted that if all four lookups of this design are done in one cycle, using a network of 16 Tbox tables described previously, they could be stored in Look Up Tables (LUTs) in each individual Logic Element (Altera's LE) or Configurable Logic Block (Xilinx's CLB). This approach allows 128 bits to be calculated every cycle but comes at a high logic element cost. This made it possible to implement the core on smaller FPGA devices, but remains better suited for the larger logic cells of the Stratix and Virtex families.

Aside from memory issues, the ports on the core had to be planned to not exceed the I/O pins available on smaller devices. The 32-bit I/O requirement was based on standard bus

Author(s)	Part Family	Device Listed	CLB Slices/LE	BRAMs	Clock Frequency	Throughput Reported	Throughput per Slice/LE
Rouvroy/Standaert	Xilinx Virtex	XC3S40-6	163	3 RAMs	123 Mhz	358 Mbit/sec	2.45 Mbit/sec/slice
Mourad et al.	Xilinx Virtex	XC4LX160	437	10 RAMs	Not listed	1716 Mbit/sec	3.93 Mbit/sec/slice
Rizk/Morsy	Xilinx Virtex 4	4V1X60	1468	0 RAMs	Not listed	1.664 Gbit/sec	3.93 Mbit/sec/slice
Wang/Ni	Xilinx Virtex-E	1000-BG860	1857	0 RAMs	125 Mhz	1.54 Gbit/sec	829 Kbit/sec/slice
McLoone/McCanny	Xilinx Virtex	XCV812E	2000	224 RAMs	Not listed	12020 Mbit/sec	4.7 Mbit/sec/slice
McLoone/McCanny	Xilinx Virtex	XCV812E	2222	100 RAMs	Not listed	6956 Mbit/sec	3.1 Mbit/sec/slice
Wang/Chang/Lin	Xilinx Virtex	XCV82E	3046	280 RAMs	61 Mhz	1.952 Gbit/sec	641 Mbit/sec/slice
Panato/Barcelo/Reis	Altera Cyclone	EP1C20F400C6	4057	0 RAMs	100 Mhz	256 Mbit/sec	63 Kbit/sec/LE
Panato/Barcelos/Reis	Altera Flex10K	EPF10KA250	4805	40960 bits	12.5 Mhz	256 Mbit/sec	53 Kbit/sec/LE
Rizk/Morsy	Xilinx Virtex 4	4V1X60	18855	200 RAMs	Not listed	28.51 Gbit/sec	2 Mbit/sec/slice
Rizk/Morsy	Xilinx Virtex 4	4V1X60	20155	200 RAMs	Not listed	23.09 Gbit/sec	1 Mbit/sec/slice
Our Architecture-CTR	Xilinx Spartan 3	XC3S1000	572	7 RAMs	93 Mhz	29.5 Mbit/sec	30 Kbit/sec/slice
Our Architecture-Encrypt	Xilinx Spartan 3	XC3S1000	1337	6 RAMs	97.6 Mhz	27.1 Mbit/sec	20 Kbit/sec/slice
Our Architecture-CTR	Altera Cyclone II	EP2C70	3941	0 RAMs	84.71 Mhz	23.5 Mbit/sec	6 Kbit/sec/LE
Our Architecture-Encrypt	Altera Cyclone II	EP2C70	4099	0 RAMs	106 Mhz	29.1 Mbit/sec	7 Kbit/sec/LE

Table 4: Core Comparison Chart

widths for interfacing to other logic on the FPGA or to an external processor. A survey of open AES cores found most were based on 128-bit I/O to the core, which would not fit in many smaller FPGA families. The I/O bottleneck was also important to consider if the core was partnered with a transceiver chip for use with Universal Serial Bus (USB), Ethernet, or serial port. A fast core throughput would be wasted if it could not exchange data with these interfaces effectively. Therefore a throughput of 20Mbyte/second was chosen based on the actual throughput seen on a number of USB transceivers and microcontrollers after the USB stack and processor overhead is taken into account. (See Table 3) This speed is not the raw 480 Mbyte/second data rates advertised in the USB Version 2.0 specification but the average observed data rates once the delays of the USB stack and the 8/16/32 bit microcontrollers that contained a USB physical layer.

VI. Use of Athena scripts

To aid in comparing results of this paper with other cores George Mason University's Cryptographic Engineering Research Group (CERG) scripts were utilized to batch synthesize the core with a variety of parts and optimization options. ATHENa, Automated Tool for Hardware EvaluatioN consists of a number of Perl scripts run with the free versions of Xilinx and Altera's synthesis tools. Version 0.2 was used for the preliminary results in Table 4 that uses Xilinx Webpack 9.1i. The early version of the tool provided an easy way to synthesize and summarize results from the tool. The results for Spartan 3, Virtex 4, and Virtex 5 parts in Table 5 were all reported by ATHENa for the encryption only and CTR mode cores. It should be noted ATHENa does not fill in the units for the metrics listed, all frequencies are in Mhz, clocks are reported in nanoseconds, latency in clock cycles, and the throughput is measured in megabytes per second. The script used is listed as an Appendix at the end of this paper.

VII. Results

After our architecture for AES 128-bit key encryption was synthesized, implemented, and placed, both the Xilinx and Altera tools reported a clock rate that would sustain the required 20 Mbyte/sec data rate. A maximum throughput of 29.4 Mbyte/sec was possible with a Cyclone II. Addition of the counter and extra register for CTR mode operation had opposite effects for the vendors. With Xilinx Spartan 3 family adding the extra data path components decreased the number of slices required. With Altera Cyclone II family an increase in Logic Elements was found. In both cases a reduction of at least 10 Mhz was observed but both product families still met the throughput requirement. Post-implementation testing showed the test data was still valid. Comparisons with other published results are provided in Table 3.

Both vendors' synthesis and implementation tools took different paths in synthesizing the design. ISE Webpack 9.1i optimized the design further than expected. When BRAMs were inferred they provided registered inputs that made separate "Old State" register banks unnecessary. The ROMs were packed into single Block RAMs to save resources. ISE 9.1i found a total of 17 ROMs, 16 of which were the 256x8 bit ROMs. One-Hot Finite State Machine (FSM) coding was used for the twenty states to optimize for speed. 96 SRL16E shift registers were created for the 64 3-bit shift registers and 32 x 4-bit shift registers, compacting the design considerably. Quartus did not use RAM resources; instead it elected to use LE blocks that allowed for a faster design in both the encryption and CTR mode variants. Options in Quartus were set to use memory for any size table but the tool found the LE

Xilinx : Spartan3	(Mhz)	(ns)	(Mhz)	(ns)	(ns)	(Mbyte/sec)	ATHENa
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3c100fg676-5	105.734	9.458	79.371	12.599	692.945	23.321	6
xc3c100fg676-5	105.734	9.458	88.347	11.319	622.545	26.342	4
xc3c100fg676-5	105.734	9.458	98.707	10.131	557.205	23.578	2
Xilinx : Spartan3a							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3s200afg320-5	149.504	6.689	98.795	10.122	556.71	28.23	5
xc3s200afg320-5	149.504	6.689	101.874	9.816	539.88	29.11	4
xc3s200afg320-5	149.504	6.689	119.119	8.395	461.725	34.03	6
Xilinx : Spartan3aDSP							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3sd1800afg676-5	149.279	6.699	94.589	10.572	581.46	27.03	9
xc3sd1800afg676-5	149.279	6.699	104.69	9.552	525.36	29.91	1
xc3sd1800afg676-5	149.279	6.699	113.404	8.818	484.99	32.40	7
Xilinx : Spartan3e							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3s250eft256-5	149.182	6.703	97.982	10.206	561.33	27.99	1
xc3s250eft256-5	149.182	6.703	102.03	9.801	539.055	29.15	5
xc3s250eft256-5	149.182	6.703	112.867	8.86	487.3	32.25	7
Xilinx : Virtex4-LX							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc4vlx15ff668-12	194.619	5.138	168.095	5.949	327.195	48.03	4
xc4vlx25ff668-12	194.619	5.138	183.318	5.455	300.025	52.38	1
xc4vlx15ff668-12	194.619	5.138	198.847	5.029	276.595	56.81	9
Xilinx : Virtex5_LX							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc5vlx30ff676-3	282.158	3.544	171.762	5.822	320.21	49.07	1

ATHENa Script Results with Xilinx Webpack 9.1i : Encryption Core Only

Xilinx : Spartan3	(Mhz)	(ns)	(Mhz)	(ns)	(ns)	(Mbyte/sec)	ATHENa
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3c100fg676-5	95.42	10.48	75.16	13.305	745.08	30.411	5
xc3c100fg676-5	95.42	10.48	80.36	12.444	696.864	28.443	1
xc3c100fg676-5	95.42	10.48	96.327	10.317	577.752	23.582	3
Xilinx : Spartan3a							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3s200afg320-5	90.556	11.043	78.499	12.739	713.384	29.12	8
xc3s200afg320-5	90.556	11.043	91.567	10.921	611.576	24.96	5
xc3s200afg320-5	90.556	11.043	92.421	10.82	605.92	24.73	6
Xilinx : Spartan3aDSP							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3sd1800afg676-5	90.56	11.043	90.4	11.062	619.472	25.29	8
xc3sd1800afg676-5	90.56	11.043	91.391	10.942	612.752	25.01	1
xc3sd1800afg676-5	90.56	11.043	92.885	10.766	602.896	24.61	9
Xilinx : Spartan3e							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc3s250eft256-5	105.636	9.466	99.275	10.073	564.088	23.02	4
xc3s250eft256-5	105.636	9.466	103.125	9.697	543.032	22.17	2
xc3s250eft256-5	105.636	9.466	107.368	9.262	518.672	21.17	5
Xilinx : Virtex4-LX							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc4vlx15ff668-12	176.311	5.672	161.76	6.182	340.01	14.39	3
xc4vlx25ff668-12	176.311	5.672	170.648	5.86	322.3	13.64	6
xc4vlx15ff668-12	176.311	5.672	179.372	5.575	306.625	12.98	5
Xilinx : Virtex5_LX							
	SYN FREQ	SYN TCLK	IMP FREQ	IMP TCLK	LATENCY	THROUGHPUT	RUN
xc5vlx30ff676-3	231.578	4.318	215.424	12.5	255.31	10.803	1

ATHENa Script Results with Xilinx Webpack 9.1i : CTR Mode

Table 5: Sample of ATHENa v0.2 Results

blocks faster and the overall utilization of the device was still low. For the Xilinx families the manual synthesis was compared with ATHENa's results and yielded similar after running the tool in batch mode and trying various optimization techniques. ATHENa showed the throughput required was possible even with various software switches in use and batch mode tested various Xilinx part families with one script and the same source files.

VIII. Performance and Resource Comparisons

Our design was built on trade-offs but aimed for mid-range FPGAs. A survey of published results found the ECB AES128 encryption-only core and CTR mode AES128 core both were slower than the other cores researched but used a smaller mix of resources than most of the other high throughput cores. The highest throughput cores utilized many BRAM modules and as few slices as possible. The downside was heavy use of fast BRAM resources and the core still required a large number of Slices/LE. The Tbox based designs of McLoone and McCanny, Wang et al., and Rourvoy and Standaert all had higher throughput but required the use of Xilinx Vertex family parts which have larger slices than the Spartan 3 family. [15,26,22]

The closest design to this paper's architecture would be the compact design of Rouvroy et al.. [22] Their design team targeted the Virtex XC3S50 with 163 Slices, 3 RAM blocks, and a 71.5 Mhz clock. While less slices and LUTs were used they were still only able to output a 128-bit block every 44 cycles. Moving to the XC2V40 part, the clock rose to 123 Mhz but still had a 44-cycle latency before the data was ready. They reported a throughput of 26 Mbyte/sec (208 Mbit/sec) using the XC3S50 and 44 Mbyte/sec (358 Mbit/sec) using the XC2V40. [22] The difference in resources required could be attributed to the full utilization of SRL16 registers, dual port RAM, and short data paths. Their Spartan 3 throughput result was similar to this paper's, but our architecture can be directly synthesized on Altera's Cyclone family. The other design was optimized for Xilinx parts. We believe the use of the lower cost Spartan and Cyclone devices both met the design goals and made good use of the available resources to achieve 20 Mbyte/sec throughput.

IX. Conclusion

This paper set out to create a core capable of sustaining a 20 Mbytes/sec data rate while being portable to both Xilinx and Altera's mid-range FPGA families. The use of tBoxes allowed memory available on the devices to simplify and speed up the basic AES/Rijndael algorithm. Our architecture also set out to present an area efficient design capable of using a 3-in-1 key schedule since many surveyed papers only offered AES with 128-bit keys. Reuse of data paths and foreknowledge of the intended application for the core drove design trade-offs. Design choices were made in favor of saving resources while not sacrificing throughput. The application of encryption modes was introduced to simplify the design of the core and prevent duplicate blocks of data from being encrypted to the same cipher block. The final result was portable, had a small footprint, and was ideal for becoming a coprocessor or peripheral core in an embedded design.

X. APPENDIX A: ATHENA V0.2 SCRIPT LISTING

```
# work directory, used as a root for all result directories
WORK_DIR = <C:\single_run\tboxAES128\>
# directory containing synthesizable source files for the project
SOURCE_DIR = <C:\single_run\tboxAES128\src>
# synthesizable source files listed in the order suitable for synthesis and implementation
# low level modules first, top level entity last
SOURCE_FILES = sbox.vhd, regN.vhd, tboxes.vhd, Rcon.vhd, upcounter.vhd, keyReg8x32.vhd,
shftReg8x32.vhd, datapath.vhd, AES128controller.vhd, AES128.vhd
# directory containing synthesizable source files for the project
TESTBENCH_DIR = <../sources>
# testbench files listed in the order suitable for simulation
# low level modules first, top level entity last
TESTBENCH_FILES =
# project name
# it will be used in the names of result directories
PROJECT_NAME = tboxAES128
# name of top level entity
TOP_LEVEL_ENTITY = tboxAES128
# name of top level architecture
TOP_LEVEL_ARCH = core
# name of clock net
CLOCK_NET = clk
#formulas for latency
LATENCY = TCLK*55
#formulas for THROUGHPUT
THROUGHPUT = TCLK*128/55
# OPTIMIZATION_TARGET = speed | area
OPTIMIZATION_TARGET = speed
# OPTIONS = default | user
OPTIONS = default

# APPLICATION = single_run | placement_search
# single_run: single run through synthesis and implementation with options
#           defined in the file options.<OPTIONS>_<OPTIMIZATION_TARGET>
# placement_search: runs through impementation with different values of cost table
#     with constant options defined in options.<OPTIONS>_<OPTIMIZATION_TARGET>
#  APPLICATION = placement_search the list of all FPGA devices targeted by a given
#    application
# FPGA_VENDOR = Xilinx
# END VENDOR is used to denote the end of list of devices for a given vendor
# for FPGA_VENDOR = Xilinx
# FPGA_FAMILY = SpartanXL | Spartan-II | Spartan3 | Spartan3A | Spartan3ADSP |
# Spartan3AN | Spartan3E       VIRTEX | VIRTEX-E | VIRTEX-E EM | VIRTEX-II |
# VIRTEX-II_PRO | VIRTEX-II_PRO_X |   VIRTEX-4_LX | VIRTEX-4_SX | VIRTEX-4_FX |
#            VIRTEX-5_LX | VIRTEX-5_LXT |  VIRTEX-5_SXT | VIRTEX-5_FXT
# END FAMILY is used to denote the end of list of devices for a given family
# FPGA_DEVICES = <list of device names from vendor.device.lib separated by commas>
#| best_match | all
# For best match, parameters of the best match must be provided
FPGA_VENDOR = Xilinx
FPGA_FAMILY = Spartan3
FPGA_DEVICES = best_match
MAX_SLICE_UTILIZATION  = 0.95
MAX_BRAM_UTILIZATION   = 1.0
MAX_DSP_UTILIZATION    = 1.0
MAX_MUL_UTILIZATION    = 1.0
MAX_PIN_UTILIZATION    = 1.0
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_SYN_FREQ = 50
REQ_IMP_FREQ = 25
END FAMILY
FPGA_FAMILY = Spartan
FPGA_DEVICES = best_match
MAX_SLICE_UTILIZATION  = 0.95
MAX_BRAM_UTILIZATION   = 1.0
MAX_DSP_UTILIZATION    = 1.0
MAX_MUL_UTILIZATION    = 1.0
MAX_PIN_UTILIZATION    = 1.0
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_SYN_FREQ = 50
REQ_IMP_FREQ = 25
END FAMILY
FPGA_FAMILY = Spartan3ADSP
FPGA_DEVICES = best_match
MAX_SLICE_UTILIZATION  = 0.95
MAX_BRAM_UTILIZATION   = 1.0
MAX_DSP_UTILIZATION    = 1.0
MAX_MUL_UTILIZATION    = 1.0
MAX_PIN_UTILIZATION    = 1.0
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_SYN_FREQ = 50
REQ_IMP_FREQ = 25
END FAMILY
FPGA_FAMILY = VIRTEX-4_LX
FPGA_DEVICES = all
MAX_SLICE_UTILIZATION  = 0.8
MAX_BRAM_UTILIZATION   = 1.0
MAX_DSP_UTILIZATION    = 1.0
MAX_MUL_UTILIZATION    = 1.0
MAX_PIN_UTILIZATION    = 0.9
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_SYN_FREQ = 450
REQ_IMP_FREQ = 400
END FAMILY
FPGA_FAMILY = VIRTEX-5_LX
FPGA_DEVICES = best_match
MAX_SLICE_UTILIZATION  = 0.95
MAX_BRAM_UTILIZATION   = 1.0
MAX_DSP_UTILIZATION    = 1.0
MAX_MUL_UTILIZATION    = 1.0
MAX_PIN_UTILIZATION    = 1.0
SYN_CONSTRAINT_FILE = default
IMP_CONSTRAINT_FILE = default
REQ_SYN_FREQ = 50
REQ_IMP_FREQ = 25
END FAMILY
END VENDOR
```

REFERENCES

[1] Altera Corp., *Advanced Synthesis Cookbook: A Design Guide for Stratix II/III/IV Devices*, available at http://www.altera.com/literature/manual/stx_cookbook.pdf.

[2] Altera Corp., *Cyclone II Device Handbook, Vol. 1*, available at http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf.

[3] Altera Corp., *Stratix III FPGAs vs. Xilinx Virtex-5 Devices: Architecture and Performance Comparison*, available at http://www.altera.com/literature/wp/wp-01007.pdf.

[4] Chodowiec, P., Khuon, P., Gaj, K., "Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining", Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays, p.94-102, February 2001.

[5] FIPS 197: Advanced Encryption Standard. National Institute of Standards and Technology, 2001, available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[6] Fischer, V. and Drutarovsky, M., "Two Methods of Rijndael Implementation in Reconfigurable Hardware," Lecture Notes in Computer Science, Volume 2162/2001, pp 77-92, 2001.

[7] Gaj, K. and Chodowiec P. "FPGA and ASIC Implementations of AES" in *Cryptographic Engineering*, Koc, Cenin Kaya (ed.), ISBN 978-0-387-71816-3, Springer Science+Business Media LLC, 2009, pp 235-294.

[8] Gaj, K. and Chodowiec, P., "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," *Proc. Third Advanced Encryption Standard Candidate Conf.*, pp. 40-56, 2000.

[9] Hodjat, A., Verbauwhede, I., "Area-Throughput Trade-offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 366-372, April, 2006.

[10] Huang, C., Chang, C., Lin, M., Tai, H., "The FPGA Implementation of 128-bits AES Algorithm Based on Four 32-bits Parallel Operation," *Data, Privacy, and E-Commerce, International Symposium on*, pp. 462-464, The First International Symposium on Data, Privacy, and E-Commerce (ISDPE 2007), 2007.

[11] J. Daemen and V. Rijmen. *The Design of Rigndael: AES – The Advanced Encryption Standard*, ISBN 3-540-42580-2, Springer-Verlag, 2002.

[12] Li, C., Chien, C., Hong, J., Chang, T., "An Efficient Area-Delay Product Design for MixColumns/InvMixColumns in AES," *VLSI, IEEE Computer Society Annual Symposium on*, pp. 503-506, 2008 IEEE Computer Society Annual Symposium on VLSI, 2008.

[13] Lin, S., and Huang, C., "A High-Throughput Low-Power AES Cipher for Network Applications," Proceedings of the 2007 Asia and South Pacific Design Automation Conference, pp 595-600, 2007.
[14] Lotfy, O., Ahmed, M., Camel, T., "AES Embedded Hardware Implementation," *Adaptive Hardware and Systems, NASA/ESA Conference on*, pp. 103-109, Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), 2007.
[15] McLoone, M., McCanny, J., "Rijndael FPGA Implementation Utilizing Look-up Tables",IEEE Workshop on Signal Processing Systems, pp. 349-360, 2001.
[16] Panato, A., Barcelo, M., Reis, R., "A Low Device Occupation IP to Implement Rijndael Algorithm", Design, Automation and Test in Europe Conference and Exhibition, pp 20 – 25, 2003.
[17] Rizk, M.R.M., Morsy, M., "Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA," 2nd International Design and Test Workshop, pp. 207-217, 2007.
[18] Rodriguez-Heriquez, F., Saqib, N. A., Diaz-Perez, A., "4.2Gbit/s single-chip FPGA Implementation of AES Algorithm", Electronics Letters Volume 39, Issue 15, pp. 1115 – 1116, 2003.
[19] Rodriguez-Henriquez, F, Saqib, N. A., Perez, A. D., Koc, C. K., Cryptographic Algorithms on Reconfigurable Hardware, ISBN 0-387-33883-7, Springer Science+Business Media LLC, 2006.
[20] Schaumont, P., Kuo, H., Verbauwhede, I., "Unlocking the Design Secrets of a 2.29 Gb/s Rijndael Processor," Annual ACM IEEE Design Automation Conference, Proceedings of the 39th annual Design Automation Conference, 2002.
[21] Stallings, W., Cryptography and Network Security, ISBN 0-13-1873162, Pearson Prentice Hall, 2006, Upper Saddle River, NJ, 2006.
[22] Standdaert, F., Rouvroy, G., Quisquater, J., Legat, J., "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications," Information Technology: Coding and Computing, International Conference on, vol. 2, pp. 583, International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2, 2004.
[23] Standdaert, F., Rouvroy, G., Quisquater, J., Legat, J., "A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL," International Symposium on Field Programmable Gate Arrays, Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays, pp 216 - 224 , 2003.
[24] Standaert, F., Rouvroy G., Quisquater, J., Legat, J., "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," Lecture Notes in Computer Science, Volume 2779/2003, pp 334-350, 2003.
[25] Wang, S. and Ni, W., "An Efficient FPGA Implementation of Advanced Encryption Standard," Computational Intelligence and Multimedia Applications, International Conference on, vol. 2, pp. 179-187, International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007), 2007.
[26] Wang, J., Chang, S., Lin, P., "A Novel Round Function Architecture for AES Encryption/Decryption Utilizing Look-up Table," Application-Specific Systems, Architectures and Processors, IEEE International Conference on, pp. 296, 13th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'02), 2002.
[27] Xilinx Corp., Spartan-3 FPGA Family Datasheet, available at http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf.
[28] Xilinx Corp., Advantages of the Virtex-5 FPGA 6-Input LUT Architecture, available at http://www.xilinx.com/support/documentation/white_papers/wp284.pdf.
[29] Zhong, Y., Wang J., Zhao, Z., Yu, D., Li, L., "A Low –Cost and High Efficiency Architecture of AES Crypto-Engine," Communication and Networking in China 2007, *pp. 308-312, Second International Conference on Communications and Networking in China (CHINACOM 2007), 2007*.

SOFTWARE REFERENCES

[30] Bertoni,G., Breveglieri, L., Fragneto, P., Macchetti, M., Marchesin, S., "Efficient Software Implementation of AES on 32-Bit Platforms," Revised papers from the 4th International Workshop of Cryptographic Hardware and Embedded Systems, pp. 159-171, 2002.
[31] Bernstein, D., Schwabe, P., "New AES Software Speed Records," *Lecture Notes in Computer Science Vol. 5365*, pp. 322-336, Proceedings of the 9th International Conference on Cryptology in India, 2008.