Fast and Efficient Implementation of AES Via Instruction Set Extensions

AJ Elbirt , *Member, IEEE*

Abstract—
Efficient implementation of block ciphers is critical towards achieving both high security and high-speed processing. Numerous block ciphers, including the Advanced Encryption Standard (AES), have been proposed and implemented, using a wide and varied range of functional operations. Existing microprocessor architectures do not provide this broad range of support. However, the advent of intellectual property (IP) processor cores presents the opportunity to augment existing datapaths with instruction set extensions to add acceleration modules in the form of new instructions. We will present a general purpose instruction set extension to a 32-bit SPARC V8 compatible processor core that accelerates the performance of Galois Field fixed field constant multiplication, a core element of the AES algorithm. This extension will be shown to accelerate AES encryption versus pure software implementations at a small hardware cost. This matches the improvement demonstrated in previously proposed AES-specific instruction set extensions while maintaining a generalized implementation format capable of supporting other algorithms that use Galois Field fixed field constant multiplication.

Keywords: cryptography, AES, block cipher, Galois Field

I. Introduction

With more than 188 million Americans connected to the Internet [1], information security has become a top priority. Many applications — electronic mail, electronic banking, medical databases, and electronic commerce — require the exchange of private information. For example, when engaging in electronic commerce, customers provide credit card numbers when purchasing products. If the connection is not secure, an attacker can easily obtain this sensitive data. In order to implement a comprehensive security plan for a given network to guarantee the security of a connection, *Confidentiality*, *Data Integrity*, *Authentication*, and *Non-repudiation* must be provided [2], [3], [4].

Cryptographic algorithms used to ensure confidentiality fall within one of two categories: private-key (also known as symmetric-key) and public-key. Symmetric-key algorithms use the same key for both encryption and decryption. Conversely, public-key algorithms use a public key for encryption and a private key for decryption. In a typical session, a public-key algorithm will be used for the exchange of a session key and to provide authenticity through digital signatures. The session key is then used in conjunction with a symmetric-key algorithm. Symmetric-key algorithms tend to be significantly faster than public-key algorithms and as a result are typically used in bulk data encryption [3]. The two types of symmetric-key algorithms are block ciphers and stream ciphers. Block ciphers operate on a block of data while stream ciphers encrypt individual bits. Block ciphers are typically used when performing bulk data encryption and the data transfer rate of the connection directly follows the throughput of the implemented algorithm.

High throughput encryption and decryption are becoming increasingly important in the area of high-speed networking. Many applications demand the creation of networks that are both private and secure while using public data-transmission links. These systems, known as Virtual Private Networks (VPNs), can demand encryption throughputs at speeds exceeding Asynchronous Transfer Mode (ATM) rates of 622 million bits per second (Mbps). Increasingly, security standards and applications are defined to be algorithm independent. Although context switching between algorithms can be easily realized via software implementations, the task is significantly more difficult when using hardware implementations. The advantages of a software implementation include ease of use, ease of upgrade, ease of design, portability, and flexibility. However, a software implementation offers only limited physical security, especially with respect to key storage [3], [5]. Conversely, cryptographic algorithms that are implemented in hardware are by nature more physically secure as they cannot easily be read or modified by an outside attacker when the key is stored in special memory internal to the device [5]. As a result, the attacker does not have easy access to the key storage area and cannot discover or alter its value in a straightforward manner [3].

When using a general-purpose processor, even the fastest software implementations of block ciphers cannot satisfy the required bulk data encryption data rates for high-end applications [6], [7], [8], [9], [10]. As a result, hardware implementations are necessary for block ciphers to achieve this required performance level. Although traditional hardware implementations lack flexibility with respect to algorithm and parameter switching, configurable hardware devices offer a promising alternative for the implementation of processors via the use of IP cores in Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) technology. To illustrate, Altera Corporation offers IP core implementations of the Intel 8051 microcontroller and the Motorola 68000 processor in addition to their own Nios®-II embedded processor [11]. Similarly, Xilinx Inc. offers IP core implementations of the PowerPC processor in addition to their own MicroBlaze^{TM} and PicoBlaze^{TM} embedded processors [12]. ASIC and FPGA technologies provide the opportunity to augment the existing datapath of a processor implemented via an IP

AJ Elbirt is an Assistant Professor with the Department of Computer Science, University of Massachusetts Lowell, Lowell, MA 01854, USA. Phone: 978-934-3328, Fax: 978-934-3551, Email: Adam_Elbirt@uml.edu

core to add acceleration modules supported through newly defined instruction set extensions targeting performance-critical functions [13], [14], [15]. Moreover, many licensable and extendible processor cores are also available for the same purpose [16], [17], [18], [19].

The process to develop a Federal Information Processing Standard (FIPS) was initiated by NIST to specify an Advanced Encryption Algorithm to replace the Data Encryption Standard (DES) which expired in 1998 [3]. In October 2000, NIST chose Rijndael as the AES Advanced Encryption Algorithm. One of the core operations of AES is a Galois Field fixed field constant multiplication, which is also a core operation of other block ciphers, such as Magenta, MISTY1, MISTY2, SHARK, SQUARE, and Twofish [20], [21], [22], [23], [24], [25], [26]. Unfortunately, Galois Field fixed field constant multiplication does not map well to traditional processor instruction sets. However, moving the execution of operations such as Galois Field fixed field constant multiplication from software to hardware has been demonstrated to have a significant impact upon performance [27]. Most previous work examining fast and efficient AES implementation has targeted either hardware-only or software-only implementations, with hardware implementations outperforming software implementations by an order of magnitude in terms of throughput. In contrast, the use of instruction set extensions follows the hardware/software co-design paradigm to achieve the performance and physical security associated with hardware implementations while providing the portability and flexibility traditionally associated with software implementations [28]. Moreover, when considering alternative solutions, instruction set extensions result in significant performance improvements versus traditional software implementations with considerably reduced logic resource requirements versus hardware-only solutions such as co-processors [27], [29], [30], [31], [32], [33], [34], [35], [36].

What follows is a brief overview of previous work regarding implementations of AES via software, hybrid architectures, cryptographic co-processors, and instruction set extensions. Implementations of fast and efficient Galois Field fixed field constant multiplication will also be considered. Following this examination, a hardware architecture that achieves efficient implementation of generalized Galois Field fixed field constant multiplication will be presented. The proposed implementation will be analyzed in terms of system performance and resource utilization. The results of this analysis will then be compared to the results of other AES implementations using instruction set extensions.

II. Previous Work

As detailed in Section I, the flexibility of a software implementation is often undermined by the performance degradation evidenced when targeting a general purpose processor whose instruction set cannot provide a fast and efficient implementation. Conversely, custom hardware implementations offer a significant performance advantage versus software implementations but at increased system cost with virtually no flexibility. This concept holds especially true in the case of the AES algorithm, where the bottlenecks occur in the SubBytes and MixColumns transformations, one or both of which are usually implemented via look-up tables [36], [37], [38], [39]. Often most of the AES round transformations — SubBytes, ShiftRows, and MixColumns — are combined into large look-up tables termed *T tables* [40]. Such implementations require up to three *T tables* whose size may be either 1 KB or 4 KB where the smaller tables require performing an additional rotation operation. The goal of the *T tables* is to avoid performing the MixColumns and InvMixColumns transformations as these operations perform Galois Field fixed field constant multiplication, an operation which maps poorly to general purpose processors [36]. However, the use of *T tables* has significant disadvantages. The *T tables* significantly increase code size, their performance is dependent on the memory system architecture as well as cache size, and their use causes key expansion for AES decryption to become significantly more complex [28], [36].

As an alternative to the *T tables* implementation method, it is also feasible to have the processor perform all of the AES round transformations. Row-based implementations have been demonstrated to allow for greater efficiency in the implementation of the MixColumns and InvMixColumns transformations versus column-based implementations [41]. However the SubBytes transformation still remains as a bottleneck, requiring separate 256 byte look-up tables for encryption and decryption [36].

Hybrid architectures comprised of a processor core combined with reconfigurable function blocks are typically used to accelerate the performance of a general purpose processor for specific applications. Reconfigurable function blocks may support on-the-fly reconfiguration to provide more optimized implementations and further improve system performance. The mapping of complex functions to adaptable hardware reduces the instruction fetch and execute bottleneck common to a software implementation [42]. However, the delay associated with communication between the processor and the reconfigurable logic block often becomes the bottleneck within the system [43]. This overhead can be reduced by caching multiple configurations within the reconfigurable function blocks at the cost of more expensive and less flexible hardware [44], [45], [46], [47]. Hybrid architectures have been targeted at accelerating applications such as symmetric-key cryptography, digital signal processing, data compression, image processing, video processing, multimedia, block matching, automated target recognition, and wireless communications. Symmetric-key implementations targeting the ConCISe, Garp, and MorphoSys architectures have all demonstrated significant performance improvements by off-loading inefficient operations from the processor to the reconfigurable logic blocks [43], [45], [48], [49], [50]. The Garp architecture is of particular interest in that it combines a standard single-issue MIPS processor with a reconfigurable array used as a hardware accelerator that attaches to the MIPS processor as a co-processor. The reconfigurable array is composed of a matrix of logic blocks whose configuration is controlled by the MIPS processor and accelerated by a configuration cache. The operation of the reconfigurable array is implemented via extensions to the MIPS instruction set. A theoretical implementation of DES in the Garp architecture operating at 133 MHz achieved a factor 24 speed-up versus an equivalent implementation on a 167 MHz Sun UltraSPARC 1/170. The Garp implementation was able to directly implement the S-Box look-up tables in parallel within the reconfigurable array. By avoiding referencing external memory the cycle count was greatly reduced [43], [49], [50].

Numerous other co-processors have been developed to accelerate cryptographic algorithm implementations. The CryptoManiac VLIW co-processor [51] was developed as a result of instruction set extensions designed to accelerate the performance of a number of the AES candidate algorithms [30]. CryptoManiac features the execution of up to four instructions per cycle and the use of instructions with up to three operands to allow for the combination of short latency instructions for single cycle execution. Similarly, the Cryptonite co-processor is also VLIW based, with two 64-bit datapaths and special instructions combined with dedicated memories to support AES implementations [52]. Both co-processors improve the performance of AES implementations versus implementations targeting general purpose processors. The implementations in [53] and [54] couple an FPGA co-processor with a LEON-2 processor core. The co-processors connect to the LEON-2 processor core via either a dedicated interface or as a memory-mapped peripheral and were able to significantly improve the performance of AES implementations [53], [54].

Examples of instruction set extensions designed to improve the performance of cryptographic algorithms include those implemented to perform arithmetic over the Galois Field GF(2^m), usually targeting elliptic curve cryptography (ECC) systems. Word-level polynomial multiplication was shown in [55] to be the time-critical operation when targeting an ARM processor and a special Galois Field multiplication instruction resulted in significant performance improvement. Instruction set extensions targeting a SPARC V8 processor core were used to accelerate the multiplication of binary polynomials for arithmetic in GF(2^m) in [56], resulting in almost double the performance for the Galois Field GF(2^{191}) and a fixed reduction polynomial. Similar results were shown in [57] using the same instruction set extensions retargeted to a 16-bit RISC processor core. The implementation in [34] targets a MIPS32 architecture and also attempts to accelerate word-level polynomial multiplication through the use of Comba's method of handling the inner loops of the multiplication operation, resulting in a performance improvement by a factor 6. Numerous generalized Galois Field multipliers have also been proposed for use in elliptic curve cryptosystems [58], [59], [60], [61], [62], [63]. These implementations focus on accelerating exponentiation and inversion in Galois Fields $GF(2^m)$ where $m \approx 160-256$. Because they do not employ a Galois Field fixed field constant matrix, these implementations are more generalized than is necessary when targeting block cipher implementations. Moreover, block ciphers employ Galois Field fixed field constant multiplication for small m, resulting in hardware that performs the multiplication at the bit level with no complex multiplication algorithms.

Instruction set extensions designed to minimize the number of memory accesses and accelerate the performance of AES implementations have been proposed for a wide range of processors [28], [64], [65], [66]. The extensions in [64] target a general-purpose RISC architecture with multimedia instructions. Strategies are presented to implement AES using multimedia instructions while specifically attempting to minimize the number of memory accesses. While the processor is datapath-scalable, the strategies in [64] do not map well to 32-bit architectures. Extensions proposed in [65] are designed to combine the SubBytes and MixColumns AES functions into one *T table* look-up operation to speed up algorithm execution. However, the functional unit in [65] requires a significant amount of hardware to implement and cannot be used for either the final AES round (where the MixColumns function is not used) or key expansion (where the SubBytes function is used without the MixColumns function). However, *T table* performance is heavily dependent upon available cache size [28]. The extensions proposed in [66] target the Xtensa 32-bit processor and improve the performance of AES encryption but worsen the performance of decryption. Unfortunately, functionality and area overhead information for the associated instruction extensions is not provided. The implementation in [28] targets a LEON-2 processor core and combines the SubBytes and ShiftRows AES functions through the use of an instruction set extension termed *sbox*. Special instructions are also provided to efficiently compute the MixColumns AES function through the use of ECC instruction set extensions as proposed in [35]. The combination of these extensions results in a performance improvement of up to 3.68 for encryption and 2.76 for decryption versus an AES implementation with no instruction set extensions.

III. Efficient Generalized Galois Field Fixed Field Constant Multiplication

The goal of a Galois Field fixed field constant multiplier is to perform a fixed field multiplication over a given Galois Field. As an example, consider the representative Galois Field GF(2^8), used by AES. Note that $[A_3 : A_0]$ are the input bytes and $[B_3 : B_0]$ are the output bytes [67]:

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} K_{00} & K_{01} & K_{02} & K_{03} \\ K_{10} & K_{11} & K_{12} & K_{13} \\ K_{20} & K_{21} & K_{22} & K_{23} \\ K_{30} & K_{31} & K_{32} & K_{33} \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} \tag{1}$$

The core operation in this fixed field multiplication is an 8-bit inner product that must be performed sixteen times, four per row. The four inner products of each row are then XORed to form the final output word. For a known primitive polynomial $p(x)$, $k(x)$ (representing the 8-bit constant), and a generic input $a(x)$, we create a polynomial equation of the form $b(x) = a(x) \times k(x)\ mod\ p(x)$ where each coefficient of $b(x)$ is a function of $a(x)$. This results in an 8-bit $\times$ 8-bit matrix representing the coefficients of $b(x)$ in terms of $a(x)$ [67]. To illustrate the creation of this matrix, the following example is provided. Let:

$$\begin{aligned} k(x) &= (02)_{16} = (00000010)_2 = x \\ p(x) &= x^8 + x^4 + x^3 + x + 1 \\ a(x) &= a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \end{aligned}$$

Therefore, we see that $b(x) = a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \bmod p(x)$. Reducing modulo $p(x)$ results in $b(x) = a_6x^7 + a_5x^6 + a_4x^5 + [a_3 + a_7]x^4 + [a_2 + a_7]x^3 + a_1x^2 + [a_0 + a_7]x + a_7$. This yields the resultant mapping:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \quad (2)$$

An 8×8 matrix must be generated for each K_{xy}, resulting in a total of sixteen matrices. Note that this analysis holds true for Galois Fields other than GF(2^8) with corresponding adjustments to the mapping used to calculate $b(x) = a(x) \times k(x) \; mod \; p(x)$.

A. Expected Hardware Resource Requirements

The 8×8 matrix used to calculate $b(x)$ requires 64 storage bits to hold the matrix contents. Eight 2-input AND gates are required to perform the coefficient multiplication of $a_i \times k_{i,j}$ when calculating each b_i, resulting in a total of 64 2-input AND gates. The eight resultant products for each row must be added modulo two to form the final b_i. Modulo two addition is performed via 2-input XOR gates. Using a binary tree to perform the additions results in the use of seven 2-input XOR gates when calculating each b_i for a total of 56 2-input XOR gates. Therefore, the total cost in logic resources of calculating an 8-bit inner product $A_i \times K_{i,j}$ is 64 storage bits, 64 2-input AND gates, and 56 2-input XOR gates. Finally, the cost of calculating 16 8-bit inner products is 1,024 storage bits, 1,024 2-input AND gates, and 896 2-input XOR gates.

Each $B_i = (A_0 \times K_{i,0}) + (A_1 \times K_{i,1}) + (A_2 \times K_{i,2}) + (A_3 \times K_{i,3})$. The modulo two addition is again performed via 2-input XOR gates using a binary tree, requiring the use of three sets of eight 2-input XOR gates for a total of 96 2-input XOR gates to calculate the four B_i. An additional 32 storage bits are required to hold the input values for A_i. This brings the logic resource total to 1,056 storage bits, 1,024 2-input AND gates, and 992 2-input XOR gates. Using the unit-gate model approximations in [68], a 2-input XOR gate is counted as two gate equivalents. Assuming that storage bits are implemented as SRAM, and using an estimate of four gates per SRAM bit [69], the total number of gate equivalents required to create the generalized Galois Field fixed field constant multiplier over GF(2^8) is 7,232.

The logic resource requirements of the matrix based implementation compare favorably versus a look-up table based implementation, the most likely implementation alternative based on the discussion in Section II. 8-bit to 8-bit look-up tables may be used to calculate each $A_j \times K_{i,j}$, requiring a total of sixteen look-up tables. Each look-up table has 256 addresses and eight data bits for a total of 2,048 storage bits per look-up table, resulting in 32,768 total storage bits. The modulo two addition required to calculate each B_i is again performed via 2-input XOR gates using a binary tree, resulting in an additional 24 2-input XOR gates per B_i for a total of 96 2-input XOR gates. This brings the logic resource total to 32,768 storage bits and 96 2-input XOR gates. Using the same approximations from [68] and [69], the total number of gate equivalents required to create the look-up table based implementation is 131,264, an increase by a factor of 18 in terms of gate equivalents versus a matrix based implementation. However, it should be noted that the look-up table based implementation has the added flexibility of being used to integrate the SubBytes transformation with the MixColumns transformation to form a *T table* look-up operation.

B. Expected Hardware Performance

Based on the implementation described in Section III-A, the 8×8 matrix used to calculate an 8-bit inner product requires one level of logic to perform the coefficient multiplication of $a_i \times k_{i,j}$ when calculating each b_i. Another three levels of logic are required to implement the binary tree to perform the modulo two additions to form the final b_i. Therefore the cost of calculating an 8-bit inner product is four levels of logic. Each B_i is calculated using modulo two addition performed via 2-input XOR gates using a binary tree that requires two levels of logic, resulting in the total cost of calculating each B_i being six levels of logic. Each of the four B_i may be calculated in parallel, resulting in no additional cost for calculating all of the elements.

C. Expected Software Performance

From a software perspective, we assume the use of a processor IP core that will access the matrix based implementation as part of its datapath. Table I details the instructions required to use the multiplier based on the processor word size. Multiplier configuration occurs prior to algorithm execution as part of the set-up phase. Once the matrices have been configured, the execution of additional instructions is required to load a value into the A_i storage elements. A final instruction is required to load the final value of A_i and read the result of the generalized Galois Field fixed field constant multiplication.

TABLE I
Instructions for accessing extended datapath

Operation	8-Bit	16-Bit	32-Bit	64-Bit
Matrix Config.	128	64	32	16
Look-up table Config.	4,096	2,048	1,024	512
Loading/Reading	4	2	1	1

Comparing the instruction usage for the matrix based implementation to that of a look-up table based implementation yields extremely favorable results. As shown in Table I, the number of instructions required to compute a multiplication is the same. However, the matrix based implementation results in an improvement by a factor of 32 in terms of the number of instructions required for system set-up versus a look-up table based implementation.

Note that clock cycle count per instruction is dependent upon the number of clock cycles per machine cycle and the number of machine cycles per instruction, both of which are determined by the processor IP core in use. Both the matrix based implementation and the comparison look-up table based implementation result in one instruction requiring one machine cycle to execute independent of the processor IP core in use. However, it is clear that the performance advantage of a look-up table based solution for performing a generalized Galois Field fixed field constant multiplication in one machine cycle is outweighed by both the overhead cost and logic resource requirements detailed in Table I and Section III-A, respectively. Alternative fixed constant matrix instruction set extensions exist that bridge the gap between the generalized matrix based and look-up table based implementations [35], [36]. A comparison of our proposed generalized matrix based multiplier instruction set extension versus the alternative instruction set extensions will be performed in Section IV-B.

Standard high-speed and efficient software implementations of Galois Field fixed field constant multiplication typically employ software look-up tables for algorithms such as AES. These implementations are not generalized, utilizing large fixed arrays of bytes designed to map each A_i to its corresponding B_i for a specific constant matrix of elements $K_{i,j}$ for the Galois Field GF(2^8). Such software implementations require four 8-bit to 8-bit software look-up table operations and three bit-wise XOR operations to calculate each B_i, resulting in a total of 16 look-up table operations and 12 bit-wise XOR operations [37], [38].

Table II illustrates the cost in machine cycles for such an implementation for a range of processor word sizes. Note that the machine cycle costs are based on data available in [70] and that the best/worst case data presented for the 8088 processor is dependent upon the addressing mode used to access operands in memory. Each $B_i = (A_0 \times K_{i,0}) + (A_1 \times K_{i,1}) + (A_2 \times K_{i,2}) + (A_3 \times K_{i,3})$. We assume an optimal mapping of B_i to assembly language instructions such that the look-up table operation $(A_0 \times K_{i,0})$ is performed as a MOV instruction from memory to a register. The value stored in the register is then combined with the look-up table result for $(A_1 \times K_{i,1})$ and stored back in the same register via an XOR instruction that uses $(A_1 \times K_{i,1})$ as the memory address for one of the source operands. As a result, the register contains $(A_0 \times K_{i,0}) + (A_1 \times K_{i,1})$. Similarly, another register is used to perform the same sequence of operations to calculate $(A_2 \times K_{i,2}) + (A_3 \times K_{i,3})$. One final XOR instruction is required to combine the results. This process must be repeated four times to calculate each element B_i.

Comparing the instruction usage for the matrix based implementation to that of software based implementations yields extremely favorable results. Based on the data in Tables I and II, the matrix based implementation results in a reduction in the number of machine cycles by factors ranging from 1.65 to 12.23 when computing a Galois Field

TABLE II
Cycle count comparison — software vs. proposed extension

Processor	Multiplication	Total
8088	168/208	168/208
80186	188	188
80286	104	104
80386	88	88
80486	28	28
Pentium	28	28
Matrix	1	17/33/66/132

fixed field constant multiplication. The overhead associated with system set-up in both implementation methods is incurred when the program is initially loaded. In the case of a software implementation, no additional instructions are required to populate the look-up tables while the matrix based implementation requires instructions to configure the matrices as detailed in Table I. However, this overhead becomes negligible if we assume significant usage of the Galois Field fixed field constant multiplication operation, a valid assumption when considering bulk data encryption via block ciphers such as AES.

IV. Implementation Results

A VHDL implementation of the matrix based Galois Field fixed field constant multiplier targeting the Xilinx XC2V500-6FG256 FPGA using the Xilinx ISE Project Navigator Release Version 8.1i tools was generated to validate the hardware resource requirements estimates of Section III-A [71]. The implementation assumes deployment in a processor with a 32-bit word size and results were obtained from the Xilinx ISE Project Navigator. Note that the XC2V500 embedded RAM elements are not used to hold the matrices for the matrix based implementation as the elements are more suited for use in coarse grained memory applications. Instead, the matrices are mapped to the flip-flops contained within the CLB slices.

A. *Hardware Resource Requirements Evaluation*

The resultant gate count for the matrix based implementation is 12,785. The implementation used flip-flops to implement the storage bits and the Xilinx tools use an estimate of eight gates per flip-flop rather than the estimate of four gates per SRAM bit detailed in [69] and discussed in Section III-A. Adjusting the resultant gate count by using the estimate of four gates per flip-flop yields an updated gate count of 8,557. Moreover, it is important to note that the resultant gate count from FPGA synthesis does not correspond well to actual standard-cell gate counts for for implementations of the same circuit, and thus the gate count is within expected bounds.

B. *Performance Evaluation*

Table III details the performance of the matrix based implementation versus estimates of software implementations, extending the data presented in Tables I and II. Note that throughput is calculated ignoring instructions required to configure the matrix based implementation as these instructions are performed once per algorithm implementation. As a result, the effect of these instructions on throughput is minimal when the number of Galois Field fixed field constant multiplication operations is large, as is the case when performing bulk data encryption using AES.

Based on the output of the Xilinx tools, the matrix based implementation achieved a maximum operating frequency of 200 MHz when provided with a 5 ns maximum clock period timing constraint. Note that the maximum operating frequencies for the processors in Table III were obtained from [72]. The data in Table III indicates that the matrix based implementation results in a performance improvement by factors ranging from 28.00 to 8,311.68 versus software implementations when computing a Galois Field fixed field constant multiplication. As discussed in Section I, VPNs can demand encryption throughputs at speeds exceeding ATM rates of 622 Mbps. Based on the data in Table III, the matrix based implementation is the only implementation that meets ATM encryption throughput requirements for performing Galois Field fixed field constant multiplication operations for use in block ciphers such as AES. Clearly the matrix based implementation offers a cost effective solution for accelerating Galois Field fixed field constant multiplication operations.

TABLE III
Throughput comparison

Processor	Maximum Frequency (MHz)	Throughput (Mbps)	Improvement Factor
8088	5	0.95/0.77	6,736.84/8,311.68
80186	10	1.70	3,764.70
80286	20	6.15	1,040.66
80386	40	14.55	439.86
80486	133	152.00	42.10
Pentium	200	228.57	28.00
Matrix	200	6,400.00	1.00

Table IV details the cycle counts when using the proposed matrix based implementation as an instruction set extension (termed *mixcolm*) for a LEON-2 processor core [73] versus the instruction set extensions *mixcol4* and *mixcol4s* developed in [36] targeting the same processor core. Note that the abbreviation PC-KS is used to indicate a pre-computed key schedule, OTF-KS is used to indicate an on-the-fly key schedule, and that *UR* indicates an implementation with unrolling. Cycle counts are obtained from [36] with the *mixcolm* instruction directly replacing the *mixcol4* and *mixcol4s* instructions, resulting in identical reductions in cycle count for both AES encryption and decryption [36]. These reductions are based on the instruction set extensions calculating the result of the entire 32-bit Galois Field fixed field constant multiplication in one cycle, requiring a total of four instructions to perform the operation over the entire 128-bit AES state.

It is clear from the data in Table IV and that the AES S-Box look-up table operation is the primary performance bottleneck as compared to the Galois Field fixed field constant multiplication. This finding is consistent with the findings in [43], [49], [50], where avoiding referencing external memory resulted in a significant reduction in cycle count when implementing the DES S-Box look-up tables. The effect of the instruction set extensions used to accelerate Galois Field fixed field constant multiplication are thus more pronounced when combined with the *sbox* or *sbox4s* instruction set extensions proposed in [36].

TABLE IV
AES-128 performance comparison — cycle counts

Software	Enc./Dec. PC-KS	Enc./Dec. OTF-KS
No Extensions	1,637/1,955	2,239/2,434
mixcol4/mixcolm	939/970	1,186/1,497
sbox + mixcol4/mixcolm	337/330	397/415
sbox4s + mixcol4s/mixcolm	219/218	255/300
sbox4s + mixcol4s/mixcolm UR	196/196	226/262

A critical differentiator between the two instruction set extensions is that the implementations in [36] have the values for the AES MixColumns and InvMixColumns transformation matrices hard-coded in the hardware. As a result, the implementations in [36] require roughly 700 equivalent gates and remove the need for overhead instructions used to configure the matrix based implementation. However, because the implementations in [36] are not generalized, the block ciphers detailed in Section I that also employ Galois Field fixed field constant multiplication cannot be accelerated using these instruction set extensions. While it is expected that AES will dominate implementations in fielded products, the ability to support other block ciphers at a relatively small hardware cost coupled with equivalent cycle counts versus the implementations in [36] make the matrix based implementation an attractive option.

V. Conclusions

With the advent of intellectual property processor cores, the opportunity exists to augment existing datapaths and instruction sets to add acceleration modules to achieve efficient and high-speed implementations of critical algorithms, such as the AES block cipher, used for bulk data encryption. A hardware architecture that achieves fast and efficient implementation of generalized Galois Field fixed field constant multiplication, a core operation of the AES Algorithm Rijndael, has been presented for use as an instruction set extension. A detailed discussion of the architecture has been provided and an analysis of system performance and resource utilization has been performed. This analysis has demonstrated that the proposed matrix based multiplier implementation provides a fast, efficient, and generalized solution for performing Galois Field fixed field constant multiplication at a cost of less than 7,600 additional gate equivalents. The performance analysis has shown that the matrix based multiplier outperforms software based implementations while matching the cycle count reductions of previously reported results based on AES-specific instruction set extensions.

VI. Acknowledgement

We would like to thank Stefan Tillich and Johann Großschädl from the Graz University of Technology, Institute for Applied Information Processing and Communications, for their useful discussions regarding their instruction set extensions used to accelerate AES targeting the LEON-2 processor and the examination of their associated assembly language source code.

References

[1] P. Gil, "How Big is the Internet?," World Wide Web — http://netforbeginners.about.com/cs/technoglossary/f/FAQ3.htm, 2005.

[2] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Florida, USA, 1997.

[3] B. Schneier, *Applied Cryptography*, John Wiley & Sons Inc., New York, New York, USA, 2nd edition, 1996.

[4] W. Stallings, *Network and Internetwork Security – Principles and Practice*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.

[5] R. Doud, "Hardware Crypto Solutions Boost VPN," *Electronic Engineering Times*, , no. 1056, pp. 57–64, April 12 1999.

[6] K. Aoki and H. Lipmaa, "Fast Implementations of AES Candidates," in *The Third Advanced Encryption Standard Candidate Conference*, New York, New York, USA, April 13–14 2000, pp. 106–122, National Institute of Standards and Technology.

[7] L. Bassham, III, "Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standard," in *The Third Advanced Encryption Standard Candidate Conference*, New York, New York, USA, April 13–14 2000, pp. 136–148, National Institute of Standards and Technology.

[8] J. Dray, "NIST Performance Analysis of the Final Round Java^{TM} AES Candidates," in *The Third Advanced Encryption Standard Candidate Conference*, New York, New York, USA, April 13–14 2000, pp. 149–160, National Institute of Standards and Technology.

[9] A. Sterbenz and P. Lipp, "Performance of the AES Candidate Algorithms in Java^{TM}," in *The Third Advanced Encryption Standard Candidate Conference*, New York, New York, USA, April 13–14 2000, pp. 161–168, National Institute of Standards and Technology.

[10] T. Wollinger, M. Wang, J. Guajardo, and C. Paar, "How Well Are High-End DSPs Suited for the AES Algorithms?," in *The Third Advanced Encryption Standard Candidate Conference*, New York, New York, USA, April 13–14 2000, pp. 94–105, National Institute of Standards and Technology.

[11] Altera Corporation, "Altera IP MegaStore — Embedded Processors," World Wide Web — http://www.altera.com/products/ip/processors/ipm-index.jsp.

[12] Xilinx Inc., "Xilinx Embedded Processing Technology Solutions," World Wide Web — http://www.xilinx.com/products/design_resources/proc_central/.

[13] M. Gschwind, "Instruction Set Selection for ASIP Design," in *Proceedings of the Seventh International Symposium on Hardware/Software Codesign — CODES'99*, A. A. Jerraya, L. Lavagno, and F. Vahid, Eds., Rome, Italy, March 1999, pp. 7–11, ACM Press.

[14] K. Küçükçakar, "An ASIP Design Methodology for Embedded Systems," in *Proceedings of the Seventh International Symposium on Hardware/Software Codesign — CODES'99*, A. A. Jerraya, L. Lavagno, and F. Vahid, Eds., Rome, Italy, March 1999, pp. 17–21, ACM Press.

[15] A. Wang, E. Killian, D. E. Maydan, and C. Rowen, "Hardware/Software Instruction Set Configurability for System-On-Chip Processors," in *Proceedings of the 38th Design Automation Conference — DAC 2001*, Las Vegas, Nevada, USA, June 18-22 2001, pp. 184–188, ACM Press.

[16] P. Faraboschi, G. M. Brown, J. A. Fisher, G. Desoli, and M. O. Homewood, "Lx: A Technology Platform for Customizable VLIW Embedded Processing," in *Proceedings of the 27th Annual International Symposium on Computer Architecture — ISCA 2000*, Vancouver, British Columbia, Canada, June 10-14 2000, pp. 203–213, ACM Press.

[17] R. E. Gonzalez, "Xtensa: A Configurable and Extensible Processor," *IEEE Micro*, vol. 20, no. 2, pp. 60–70, March/April 2000.

[18] ARC International, "Technical Summary of the ARCtangent^{TM}-A4 Processor Core," World Wide Web — http://www.arc.com/upload/download/ARCIntl_0311_TechSummary_DS.pdf, 2001.

[19] MIPS Technologies Inc., "Pro Series^{TM} Processor Cores," World Wide Web — http://www.mips.com/ProductCatalog/P_ProSeriesFamily/proseries.pdf, 2003.

[20] J. Daemen, L. Knudsen, and V. Rijmen, "The Block Cipher SQUARE," in *Fourth International Workshop on Fast Software Encryption*, Berlin, Germany, 1997, vol. LNCS 1267, pp. 149–165, Springer-Verlag.

[21] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," in *First Advanced Encryption Standard (AES) Conference*, Ventura, California, USA, 1998.

[22] K. Huber and S. Wolter, "Telekom's MAGENTA Algorithm for En-/Descryption in the Gigabit/sec Range," in *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, New York, New York, USA, 1996, vol. 6, pp. 3233–3235.

[23] M. Matsiu, "New Block Encryption Algorithm MISTY," in *Fourth International Workshop on Fast Software Encryption*, Berlin, Germany, 1997, vol. LNCS 1267, Springer-Verlag.

[24] J. Nakajima and M. Matsui, "Fast Software Implementations of MISTY1 on Alpha Processors," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82-A, no. 1, pp. 107–116, 1999.

[25] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win, "The Cipher SHARK," in *Third International Workshop on Fast Software Encryption*, D. Gollmann, Ed., Berlin, Germany, 1996, vol. LNCS 1039, Springer-Verlag, Conference Location: Cambridge, UK.

[26] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, "Twofish: A 128-Bit Block Cipher," in *First Advanced Encryption Standard (AES) Conference*, Ventura, California, USA, 1998.

[27] R. B. Lee, Z. Shi, and X. Yang, "Efficient Permutation Instructions for Fast Software Crytography," *IEEE Micro*, vol. 21, no. 6, pp. 56–69, November/December 2001.

[28] S. Tillich and J. Großschädl and A. Szekely, "An Instruction Set Extension for Fast and Memory-Efficient AES Implementation," in *Proceedings of the Ninth International Conference on Communications and Multimedia Security — CMS 2005*, J. Dittmann, S. Katzenbeisser, and A. Uhl, Eds., Salzburg, Austria, September 19-21 2005, vol. LNCS 3677, pp. 11–21, Springer-Verlag.

[29] R. B. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Micro*, vol. 15, no. 2, pp. 22–32, April 1995.

[30] J. Burke, J. McDonald, and T. M. Austin, "Architectural Support for Fast Symmetric-Key Cryptography," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems — AS-PLOS 2000*, Cambridge, Massachusetts, USA, November 12-15 2000, pp. 178–189.

[31] J. Großschädl, "Instruction Set Extension for Long Integer Modulo Arithmetic on RISC-Based Smart Cards," in *Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing — SBAC-PAD'02*, Vitoria, Espirito Santo, Brazil, October 28-30 2002, pp. 13–19.

[32] J. Großschädl and G.-A. Kamendje, "Optimized RISC Architecture for Multiple-Precision Modular Arithmetic," in *First International Conference on Security in Pervasive Computing*, Boppard, Germany, March 12-14 2003, vol. LNCS 2802, pp. 253–270, Springer-Verlag.

[33] J. Großschädl and G.-A. Kamendje, "Architectural Enhancements for Montgomery Multiplication on Embedded RISC Processors," in *Applied Cryptography and Network Security — ACNS 2003*, J. Zhou, M. Yung, and Y. Han, Eds. 2003, vol. LNCS 2846, pp. 418–434, Springer-Verlag.

[34] J. Großschädl and E. Savas, "Instruction Set Extensions for Fast Arithmetic in Finite Fields GF(p) and GF(2^m)," in *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2004*, M. Joye and J.-J. Quisquater, Eds., Cambridge, Massachusetts, USA, August 11-13 2004, vol. LNCS 3156, pp. 133–147, Springer-Verlag.

[35] S. Tillich and J. Großschädl, "Accelerating AES Using Instruction Set Extensions for Elliptic Curve Cryptography," in *International Conference on Computational Science and Its Applications — ICCSA 2005*, O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, Eds., Singapore, May 9-12 2005, vol. LNCS 3481, pp. 665–675, Springer-Verlag.

[36] S. Tillich and J. Großschädl, "Instruction Set Extensions for Ef-

ficient AES Implementation on 32-bit Processors," in *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2006*, L. Goubin and M. Matsui, Eds., Yokohama, Japan, October 10-13 2006, Springer-Verlag.

[37] P. S. L. M. Barreto, "Optimized Rijndael C Code v3.0," http://homes.esat.kuleuven.be/ rijmen/rijndael/.

[38] "Rijndael Reference Code in ANSI C v2.2," http://homes.esat.kuleuven.be/ rijmen/rijndael/.

[39] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf, 1999.

[40] J. Daemen and V. Rijmen, *The Design of Rijndael*, Springer, New York, New York, USA, 2002.

[41] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, and S. Marchesin, "Efficient Software Implementation of AES on 32-Bit Platforms," in *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2002*, B. S. Kaliski Jr., Çetin K. Koç, and Christof Paar, Eds., Redwood City, California, USA, August 13-15 2002, vol. LNCS 2523, pp. 159–1714, Springer-Verlag.

[42] K. Bondalapati and V. K. Prasanna, "Reconfigurable Computing: Architectures, Models and Algorithms," *Current Science*, vol. 78, no. 7, pp. 828–837, 2000.

[43] K. K. Bondalapati, *Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures*, Ph.D. thesis, University of Southern California, Los Angeles, California, USA, August 2001.

[44] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The Chimaera Reconfigurable Function Unit," in *Fifth Annual IEEE Symposium on Field-Programmable Custom Computing Machines — FCCM '97*, Napa Valley, California, USA, April 16–18 1997, pp. 87–96.

[45] B. Kastrup, A. Bink, and J. Hoggerbrugge, "ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator," in *Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines — FCCM '99*, Napa Valley, California, USA, April 21-23 1999, pp. 92–101.

[46] M. J. Wirthlin and B. L. Hutchings, "A Dynamic Instruction Set Computer," in *Third Annual IEEE Symposium on Field-Programmable Custom Computing Machines — FCCM '95*, Napa Valley, California, USA, April 19-21 1995, pp. 99–107.

[47] M. J. Wirthlin and B. L. Hutchings, "DISC: The Dynamic Instruction Set Computer," in *Proceedings of Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, J. Schewel, Ed., Philadelphia, Pennsylvania, USA, October 25–26 1995, vol. 2607, pp. 92–103, SPIE - The International Society for Optical Engineering.

[48] H. Singh, M. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, May 2000.

[49] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The Garp Architecture and C Compiler," *Computer*, vol. 33, no. 4, pp. 62–69, April 2000.

[50] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor With A Reconfigurable Coprocessor," in *Fifth Annual IEEE Symposium on Field-Programmable Custom Computing Machines — FCCM '97*, Napa Valley, California, USA, April 16–18 1997.

[51] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: A Fast Flexible Architecture for Secure Communication," in *Proceedings of the 28th Annual International Symposium on Computer Architecture — ISCA-2001*, B. Werner, Ed., Goteborg, Sweden, June 30–July 4 2001, pp. 110–119.

[52] D. Oliva, R. Buchty, and N. Heintze, "AES and the Cryptonite Crypto Processor," in *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems — CASES 2003*, J. H. Moreno, P. K. Murthy, T. M. Conte, and P. Faraboschi, Eds., San Jose, California, USA, October 30-November 1 2003, pp. 198–209, ACM Press.

[53] P. Schaumont, K. Sakiyama, A. Hodjat, and I. Verbauwhede, "Embedded Software Integration for Coarse-Grain Reconfigurable Systems," in *Proceedings of the Eighteenth International Parallel and Distributed Processing Symposium — IPDPS 2004*, Santa Fe, New Mexico, USA, April 26-30 2004, pp. 137–142.

[54] A. Hodjat and I. Verbauwhede, "Interfacing a High Speed Crypto Accelerator to an Embedded CPU," in *Proceedings of the 38th Asilomar Conference on Signals, Systems, and Computers*, Los Angeles, California, USA, November 7-10 2004, vol. 1, pp. 488–492.

[55] S. Bartolini, I. Branovic, R. Giorgi, and E. Martinelli, "A Performance Evaluation of ARM ISA Extension for Elliptic Curve Cryptography Over Binary Finite Fields," in *Proceedings of the Sixteenth Symposium on Computer Architecture and High Performance Computing — SBC-PAD 2004*, Foz do Iguaçu, Brazil, October 27-29 2004, pp. 238–245.

[56] S. Tillich and J. Großschädl, "A Simple Architectural Enhancement for Fast and Flexible Elliptic Curve Cryptography Over Binary Finite Fields GF(2^m)," in *Proceedings of the Ninth Asia-Pacific Conference on Advances in Computer Systems Architecture — ACSAC 2004*, Beijing, China, September 7-9 2004, vol. LNCS 3189, pp. 282–295, Springer-Verlag.

[57] J. Großschädl and G.-A. Kamendje, "Instruction Set Extension for Fast Elliptic Curve Cryptography Over Binary Finite Fields GF(2^m)," in *Proceedings fo the Fourteenth IEEE Conference on Application-Specific Systems, Architectures and Processors — ASAP 2003*, The Hague, The Netherlands, June 24-26 2003, pp. 455–468.

[58] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "An FPGA Implementation of a GF(p) ALU for Encryption Processors," *Microprocessors and Microsystems*, vol. 28, no. 5-6, pp. 253–260, August 2004.

[59] M. A. Hasan and M. Ebtedaei, "Efficient Architectures for Computations Over Variable Dimensional Galois Fields," *IEEE Transactions on Circuits and Systems I*, vol. 45, no. 11, pp. 1205–1211, November 1998.

[60] P. Kitsos, G. Theodoridis, and O. Koufopavlou, "An Efficient Reconfigurable Multiplier Architecture for Galois Field $GF(2^m)$," *Microelectronics Journal*, vol. 34, no. 10, pp. 975–980, October 2003.

[61] P. Kitsos, G. Theodoridis, and O. Koufopavlou, "An Reconfigurable Multiplier in $GF(2^m)$ for Elliptic Curve Cryptosystem," in *Proceedings of the Tenth IEEE International Conference on Electronics, Circuits and Systems — ICECS 2003*, Sharjah, United Arab Emirates, December 14–17 2003, vol. 2, pp. 699–702.

[62] M. A. G. Martinez, G. M. Luna, and F. R. Henriquez, "Hardware Implementation of the Binary Method for Exponentiation in $GF(2^m)$," in *Proceedings of the Fourth Mexican International Conference on Computer Science*, E. Chávez, J. Favela, M. Mejía, and A. Oliart, Eds., Tlaxcala, Mexico, September 8–12 2003, pp. 131–134.

[63] E. M. Popovici and P. Fitzpatrick, "Algorithm and Architecture for a Galois Field Multiplicative Arithmetic Processor," *IEEE Transactions on Information Theory*, vol. 49, no. 12, pp. 3303–3307, December 2003.

[64] J. Irwin and D. Page, "Using Media Processors for Low-Memory AES Implementation," in *Proceedings of the Fourteenth IEEE International Conference on Application-Specific Systems, Architectures and Processors — ASAP 2003*, The Hague, The Netherlands, June 24-26 2003, pp. 144–154.

[65] K. Nadehara, M. Ikekawa, and I. Kuroda, "Extended Instructions for the AES Cryptography and Their Efficient Implementation," in *Proceedings of the Eighteenth IEEE Workshop on Signal Processing Systems — SIPS 2004*, Austin, Texas, USA, October 13–15 2004, pp. 152–157.

[66] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass, "System Design Methodologies for a Wireless Security Processing Platform," in *Proceedings of the 2002 Design Automation Conference — DAC 2002*, New Orleans, Louisiana, USA, June 10-14 2002, pp. 777–782.

[67] A. J. Elbirt, *Reconfigurable Computing for Symmetric-Key Algorithms*, Ph.D. thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, April 2002, Available at http://faculty.uml.edu/aelbirt/thesis.pdf.

[68] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1163–1170, October 1993.

[69] B. Penner, "What is Gate Count? What Are Gate Count Metrics for Virtex/Spartan-II/4K Devices?," Electronic Mail Personal Correspondance, January 2003, Xilinx Inc.

[70] "80x86 Instruction Set (8088-Pentium)," http://fux0r.phathookups.com/programming-tutorials/Assembly/opcode.html.

[71] Xilinx Inc., San Jose, California, USA, *Virtex^{TM}-II Complete Data Sheet*, 2005, Available at: http://direct.xilinx.com/bvdocs/publications/ds031.pdf.

[72] "AMD x86 (8086 - K5) Identification," World Wide Web — http://www.cpu-world.com/info/id/AMD-x86-identification.html, 2003.

[73] J. Gaisler, "The LEON-2 Processor User's Manual (Version 1.0.24)," World Wide Web — http://www.gaisler.com/doc/leon2-1.0.24-xst.pdf, September 2004.