Finding the Right Processing Architecture for AES Encryption

Bruce Schulman, VICI

6/18/2003 (This article originally appeared in Communications Systems Design, which was purchased by EE Times)

Finding the Right Processing Architecture for AES Encryption

Even before September 11, 2001, security processing has been a mainstay concern for today's communication design engineers. With rogue attacks growing, broadband deployments on the rise, e-commerce activities increasing, and distributed storage becoming a necessity, designers must build systems that provide strong security features.

To make this happen, many designers are turning to the advanced encryption standard (AES) in their wireless and wireline networking designs. But, implementing AES is not an easy task. AES is a processing-intensive algorithm that will require a significant amount of MIPS on a general-purpose processor or even a specialized signal processing architecture. Thus, when building a system, engineers must evaluate how efficiently a processor will perform when running the AES algorithm.

In this article, we'll examine how efficient and effective existing processor architectures, such as ARM, MIPS, and TI DSP processors, are at handling a 128-bit AES encryption algorithm. We'll then show how a vector signal processor can be used to execute the AES algorithm in significantly less cycle counts.

The Algorithm

Developed by the US National Institute of Standards and Technology (NIST), the AES algorithm is a complex cryptographic algorithm. **Figure 1** shows the typical data flow for one "round" of the algorithm. For a 128-bit key, there are 10 rounds.

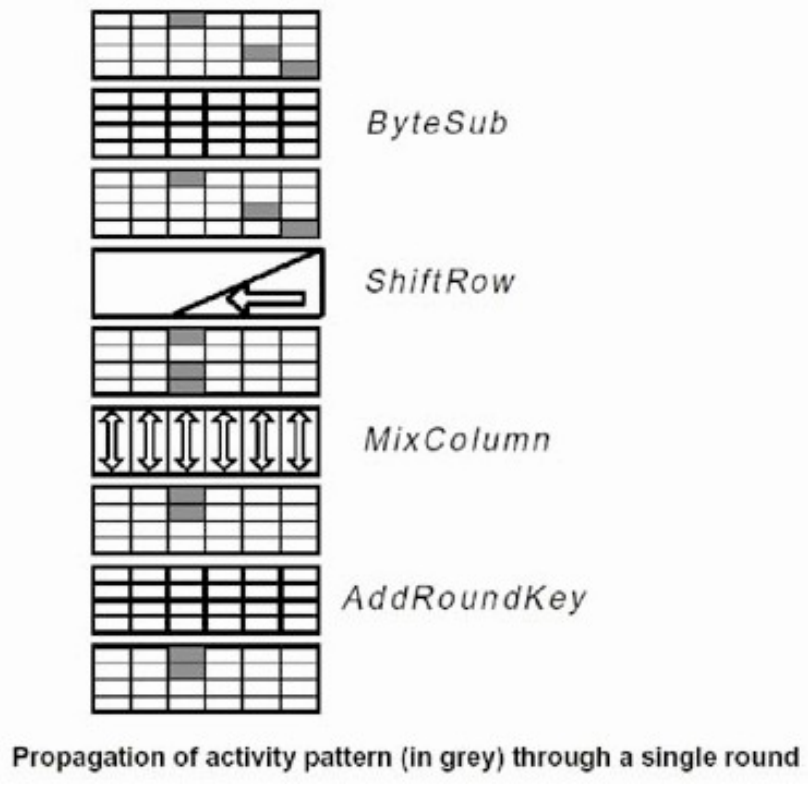

Figure 1: Typical data flow for one-round of the AES algorithm. (Source: NIST website)

Without exploring the details of Finite Field Mathematics, Gaolis Field Theory, Matrix Transformations, linear and non-linear mixing algorithms, and other math properties that lead to reversibility of these operations, the main loop of the Rijndael AES algorithm implementation can be implemented with a series of table lookups and functionally reduces to operations that will be of the form:

C or Java code:

```
a[i] = (T1[(t[ i       ] >> 24) & 0xFF] ^
       T2[(t[(i + s1)] >> 16) & 0xFF] ^
       T3[(t[(i + s2)] >>  8) & 0xFF] ^
       T4[ t[(i + s3)]        & 0xFF]  ) ^ Key[r][i];
```

This code can be implemented as combinations of the following basic operations:

1. BYTE SELECT => t[i+...] is a byte select
2. TABLE LOOK UP => T1[xxx] (the byte selected is the index for the table)
3. SHIFT/MASK => >> 24,16,8 & 0xFF
4. XOR => ^ (bit-wise logical Exclusive OR)

Key [][] is the extended encryption key.

Comparing Architectures
The 128-bit AES algorithm was selected because it could be ported to all existing CPUs. Selection of the desired architecture depends on the performance desired. Performance criteria include:

- data rate
- cost
- area: including both die are and board area
- power consumption: including battery life and heat dissipation in high channel count applications

In the sections that follow, we will use these performance criteria to determine how well existing processor architectures do at handling the AES encryption. During the discussion we'll look at the following architectures.

1.Scalar Processor—Single instruction executes on a single data element. RISC processors are in this category. This the baseline for comparison.

2. Superscalar Processor— RISC processors that feature at least one additional data path (pipeline). To allow the complex, multiple cycle instructions to be issued to both pipes, scheduling hardware is also added to minimize pipeline stalls due to data hazards. As this improvement's impact saturated, "out of order" execution scheduling hardware has been added to try and keep the pipes full.

3. Single Instruction, Multiple Data (SIMD) Extensions—These are new extra-wide data paths that can be used in place of the existing data paths with a single instruction to execute on parallel data channels packed into dedicated registers. As with many communications applications, the AES data type is 8-bit, smaller than the intrinsic 32-bit registers. Thus, when coupled with the high data parallelism intrinsic in the algorithm, many programmers assume that data is easily packed and processed. However, this regular data path processor quickly becomes instruction bound.

4. Very Longer Instruction Word (VLIW)—Commonly found at the heart of digital signal processors (DSPs), VLIW architectures issue multiple parallel instructions, usually for different parts of the data path. For instance, a five-issue VLIW might have parallel instructions, one each for the load unit, store unit, multiply-and-accumulate (MAC) unit, arithmetic logic unit (ALU), and interprocessor communication unit. This allows overlapped load/store/execute/communicate in a single instruction. Executing multiple instructions in a single clock cycle can help in applications where the algorithm is instruction issue limited. But, this approach can increase in the raw data path performance. Some processors do add parallel data path and larger VLIW instructions to address data parallelism opportunities. Quickly, the instruction bus bandwidth or load bus bandwidth can become the dominant cost and bottleneck.

5. Vector Processor—Vector processors are based on a single-instruction, multiple data architecture that is distinctly different than SIMD extension to scalar/superscalar processors. Each vector data path has some data independence from the others allowing data path dependent operations. This allows easier control for wider machines. Single chip vector processors can still be low power and easy to program even with eight parallel vector units. For many communications algorithms, characterized by high data parallelism, vector single instruction machines end up being the ideal balance of instruction/programming simplicity and compactness, while still supporting complex processing requirements and high performance.

Figure 2 shows how a vector processor implements an indirect table read single instruction (Read_Table (source.sp,destination.sp);). Each data path will read from a table (base address in serial R3 register), with an offset stored in its local register (offset stored in parallel R3), and store the result in its local parallel register R4— 1 instruction, 8 reads.

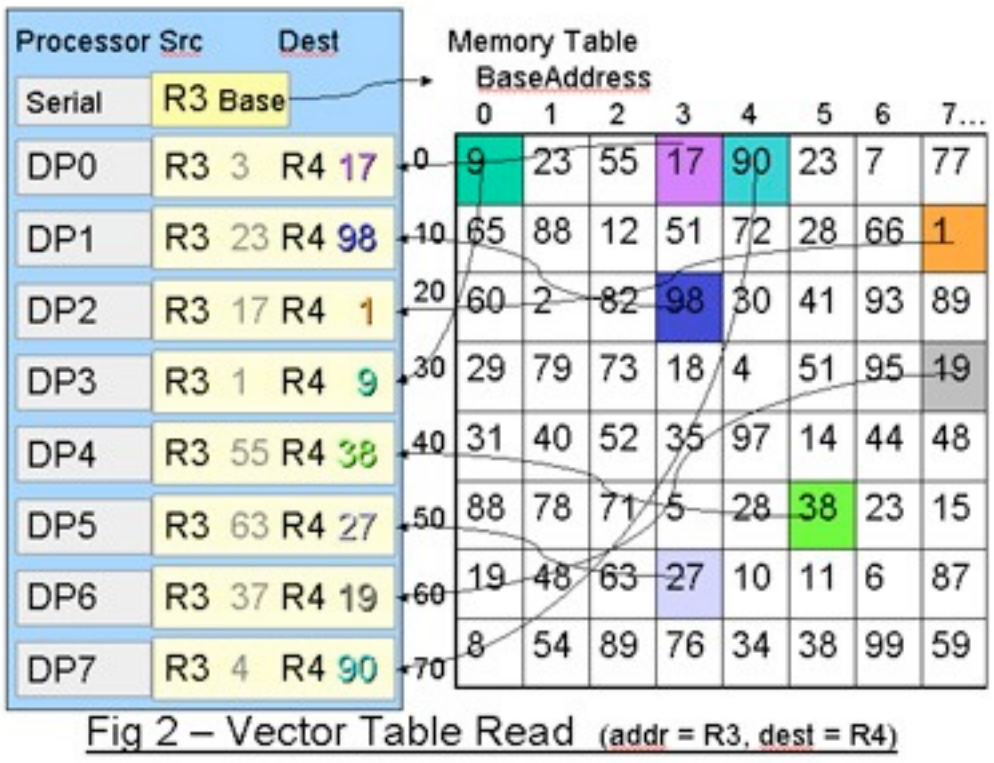

Figure 2: Indirect table read implemented on a vector processor.

Before moving forward, it's important to note that adding data paths is not an end it itself. Memory architecture and bandwidth must also be improved to accommodate the larger demand for data movement. A vector table read allows a single instruction to specify eight parallel table reads based on a data value within each vector unit. Typical SIMD architectures cannot accomplish this task because they only added the data path and not the load/store units or memory architecture.

Existing Implementations
Since most of the complex mathematical operations can be pre-computed and stored in tables, the AES algorithm can be easily implemented on a wide variety of processors without the need for specialized hardware.

AES software cycle counts have been reported in various publications. For consistency, all cycle counts in this article are for benchmarks regarding 128-bit block and 128-bit key, although other key and blocks sizes are allowed. In addition, to achieve these performance numbers, all inner loops have been implemented in optimized assembly code. C or Java implementations are significantly lower performance.

Table 1 shows the cycle count required to implement AES on various processor architectures. Note: this information is taken from various links on the NIST website.

Processor	Cycle Count Reported
Motorola 68HC08 (8-bit)	8390
TI C54x (16-bit)	3518
MIPS 32 (32-bit RISC)	730
Strong Arm (32-bit RISC)	690
MIPS 64 (64-bit RISC)	240
Intel IA-32 Pentium III / IV	228
TI C6X (256 bit VLIW)	228
Intel IA-64 McKinley	124
Chipwrights CW4011 (8 x 32-bit vector processor)	67

Since the data type is 8-bit and all processing can be performed on 8-bits, a logical option would be to use an 8-bit scalar microprocessor, in which all operations are executed sequentially. Also, all ancillary operations such as address calculations are also executed sequentially. The Motorola 68HC08 is a representative example of a standard 8-bit microcontroller with a reported AES encryption performance of 8390 cycles for a 128-bit block. With its very low power in current semiconductor processes, and relatively low cost, the 8-bit micro is a good choice where throughput is not a key issue.

The first improvement in cycle count is achieved with a 16-bit scalar processor, such as the TI C54x DSP. It moves 2x more data per instruction than an 8-bit processor due to its 16-bit data path. It also computes addresses in conjunction with processing instead of sequentially. These features enabled a greater than 2x performance improvement to 3518 cycles per 128-bit block.

The next performance enhancement is achieved by applying a 32-bit scalar microprocessor, such as an ARM or MIPS processor. To achieve the speedup expected, an algorithmic optimization is performed on the AES formulas that combine the pre-computed tables such that 32-bit table look-ups can be performed instead of 8-bit table look-ups, reducing the number of table read

operations by a factor of 4x. Additionally, the XOR functions are computed on four 8-bit values at a time (32-bit values). In total this averages to an additional gain of 5x performance over the 16-bit processors to achieve cycle counts of approximately 700. This is a good improvement, as the gate count of a RISC core is less than for a 16-bit DSP lowering the effective cost for this application.

A superscalar processor, such as the Pentium 4 with dual issue pipelines, out-of-order execution and MMX extensions allows 64-bit packed integer processing. The dual-issue pipelines with out-of-order execution typically yield a factor of 1.5x performance enhancement. The 64-bit execution units yield another factor of 2x performance enhancement for a total of 3x over 32-bit scalar processors as shown by the 228 instructions reported. Given the cost, power and area to embed a Pentium however, this may not be the most competitive solution for an embedded application. However if a Pentium is already being used, no added hardware is required.

The lowest software cycle count reported thus far, 124 cycles per block, is on the McKinley in IA-64 assembly language. This processor can operate on 128-bit blocks in a single instruction. This is approximately 2x performance enhancement over the Pentium 4. The cost and power consumption are also increased disproportionately from Pentium 4.

A New Hardware Approach

While existing architectures provide mechanisms for improving the performance of AES encryption, they still fall short in providing the cycle counts required for next-generation communication system designs. With this in mind, VICI explored the use of a vector processing architecture to handle the 128-bit AES algorithm. The processor selected was based on a RISC architecture with eight attached 32-bit vector units. The use of eight 32-bit data paths creates a 256-bit wide machine, which has a factor of 2x more data path than the McKinley.

As mentioned above, along with a wide data path, memory bandwidth needs to be increased. This processor allows a single indirect table read instruction to access eight data-path dependent table values in a single instruction, achieving the desired balance. Because of this processor's architecture and its instruction set, the same code for one data path can also run on eight data paths. With some planning, but no additional code, we were able to simply run the same single data path code on the 8-data path processor with no changes and achieved an 8x speed up over the single data path software.

This one path in the vector processor delivers performance levels better than traditional 32-bit RISC architectures (530 vs. 720). With 8 data paths, the result is a 10x speed up compared to regular RISC processors for a cycle count of 67 cycles per 128-bit block encryption.

The vector processor provides some special instructions that deal with packed data, making the vector processor instruction set more efficient at executing the 128-bit AES algorithm. These include:

- Byte select — an extract instruction allowed selecting any byte in a 32-bit register in a single instruction. This eliminates the RISC processor's typical SHIFT and MASK instructions.

- Indirect table look up — Using a table read instruction, each vector unit can do an indirect table read in one cycle based on its own address offset byte. This is VECTOR parallelism, not available in MMX type SIMD. The RISC processors typically use instructions to calculate the offset into the table and then read the table.
- XOR — With all data in 32-bit registers, the vector processor performs four 8-bit XOR instructions per vector unit per cycle (32-byte XORs per clock in SIMD).

From the results it was shown that running the AES encryption routine on a vector processor, such as the ChipWrights CW4011, require 530 instructions to perform the Rijndael algorithm on 128-bit blocks of data. By using the 8 vector units, the processor encrypts eight 128-bit blocks per iteration, an effective 67 instructions per block.

Embedded processors have dedicated DMA engines for input and output, to address the potential bottlenecks getting the data on/off the chip and eliminating the overhead of doing this task with the processor. At 0.3 W, the dedicated AES throughput we achieved with the vector signal processor was 310 Mbit/s, which is significantly lower mW per Mbit/sec than traditional processing architectures.

Wrap Up

This article reviewed criteria for choosing processor architectures when dealing with 128-bit AES encryption algorithm. This communication algorithm's intrinsic properties were exploited to achieve a full parallel speed up—8x performance for 8 data paths—in a vector Processor. Thus, a vector architecture achieves a level of performance exceeding superscalar, SIMD, and VLIW implementations, while still being relatively easy to program and scale. We achieved the lowest cost, lowest instruction issue rate, and the highest reported software performance of 67 instructions per block.

About the Author

Bruce Schulman *is the founding partner of VICI, a sales/licensing representative for silicon IP cores from Elliptic Semiconductor, Soft Mixed Signal, JPI, SPIRIT Corp. and others. VICI applications group develops parallel processor algorithms for embedded processors for Video, Audio and Communications. Bruce has also served as VP of Business Development at BOPS Inc., co-founder of Spectrum Signal Processing, Product Line Director at Xicor, and Strategic Marketing Manager at VLSI Technology (Philips) and National Semiconductor. Bruce was a fellow at MIT Center for Advanced Engineering Studies and earned a BSEE from Cornell University. He can be reached at bruce@vici.us.*