Algorithm Mapping Strategies for Advanced Encryption Standard on VLIW Digital Signal Processors

Chris Chung
Texas Instruments Incorporated
12500 TI Blvd. MS 8635
Dallas, TX 75243, USA
cchung@ti.com

David Elam
Texas Instruments Incorporated
12500 TI Blvd. MS 8635
Dallas, TX 75243, USA
d-elam2@ti.com

Abstract

The Advance Encryption Standard (AES) is a symmetric block cipher selected as a replacement of the Data Encryption Standard (DES). In this paper, we introduce several optimization considerations for VLIW DSPs and discuss the performance of two AES implementations on TMS320C64 running at 600 MHz. The first approach utilizes multiple lookup tables and results in the encryption speeds of 354 Mbps, 305 Mbps and 273 Mbps for the key lengths of 128-bit, 192-bit and 256-bit, respectively. The second approach reduces the memory requirement by utilizing a Galois-field arithmetic instruction available in TMS320C64. It is slower than the first approach, but may be useful and more effective when linked with other application code since its reduced memory requirement may result in a lower memory conflict overhead.

Keywords: AES, VLIW, DSP

1. Introduction

The Advanced Encryption Standard (AES) was issued in 2001 as a new Federal Information Processing Standard (FIPS). It is intended to replace the Data Encryption Standard (DES) that has long been used for protecting electronic data. The need for more reliable cryptographic algorithms has been significantly increased in many applications such as internet and wireless communications. The AES specifies the Rijndael cryptographic algorithm, which is faster and more compact than other cryptographic algorithms. It is also flexible for larger keys for improved security at the cost of additional rounds of transformations. The AES has a lot of inherent parallelism, which makes it easy to implement in both software and hardware.

While dedicated hardware approaches could be solutions to realize the AES [1-3], it is desirable to utilize a digital signal processor (DSP) due to its flexibility and lower implementation cost. DSPs have found widespread use for many computationally intensive signal processing applications such as communications, imaging and video. Due to the increasing demand for a high-level of security, integrating the AES into these applications is become crucial.

Today's high-performance DSPs often employ (1) a very long instruction word (VLIW) that enables the execution of multiple instructions in parallel and (2) a deep pipeline that supports high-frequency operation. In addition, the use of single-instruction-multiple-data instructions (SIMD), available in many high-performance DSPs, makes the performance even higher [4].

The AES is inherently an 8-bit algorithm in that most of the transformations operate on 8-bit pieces of a 128-bit data. Due to its inherent parallelism, the algorithm can be efficiently mapped onto today's high-performance DSPs. Efficient algorithm mapping to multiple parallel functional units in the VLIW architectures is a challenge since architectural constraints such as interaction latency and resource bounds need to be carefully managed.

In this paper, we introduce several techniques that can be applied to efficiently map the AES algorithm onto VLIW DSPs.

2. The AES Algorithm

The AES algorithm performs a set of operations called *round* on a 4 x 4 block. A higher security level can be easily achieved with a larger key, which results in more rounds of operations. For example, 10 and 12 rounds are

required for a 128-bit key and a 192-bit key, respectively. Each round includes four steps: (1) the *ByteSub* transformation, (2) the *ShiftRow* transformation, (3) the *MixColumn* transformation and (4) the *RoundKey* addition. Note that the last round does not include the *MixColumn* transformation.

The *ByteSub* transformation is a byte-substitution using a substitution table called *S-box*. The *ShiftRow* transformation is a circular shifting operation with a different number of shift amount for each row. The *MixColumn* transformation mixes data in each column by multiplication with a polynomial modulo $x^4 + 1$. The *RoundKey* addition performs XOR with a round key derived from the cipher key.

The detailed AES algorithm is well explained in the AES proposal [5] and many publications. For theoretical background and design principles behind the AES, refer to Chapter 5: Propagation and Correlation in the AES proposal.

The AES proposal introduces an efficient implementation based on look-up tables (LUT). This approach combines the first three transformations into one table lookup as shown below.

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} s[a_{0,j}] \\ s[a_{1,j-C1}] \\ s[a_{2,j-C2}] \\ s[a_{3,j-C3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

, where *e* is the round output, *a* is the round input, and *k* is the key. Here we show one column of the output.

The original equation above can be expressed as a linear combination of vectors as below.

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-C1}] \oplus T_2[a_{2,j-C2}] \oplus T_3[a_{3,j-C3}] \oplus k_j$$

, where T_0, T_1, T_2, T_3 are defined as:

$$T_0[a] = \begin{bmatrix} s[a] \bullet 2 \\ s[a] \\ s[a] \\ s[a] \bullet 3 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} s[a] \bullet 3 \\ s[a] \bullet 2 \\ s[a] \\ s[a] \end{bmatrix}$$

$$T_2[a] = \begin{bmatrix} s[a] \\ s[a] \bullet 3 \\ s[a] \bullet 2 \\ s[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} s[a] \\ s[a] \\ s[a] \bullet 3 \\ s[a] \bullet 2 \end{bmatrix}$$

In this paper, we will show three implementations based on this approach.

3. Algorithm Mapping Strategies

A VLIW DSP employs multiple functional units. It is important to note that some instructions must be executed on a specific functional unit while others can be executed on more than one unit. Figure 1 shows example functional units supporting different kinds of instructions.

Figure 1. Example functional units supporting different kinds of instructions

Functional unit	L	S	M	D
Operations performed	LOGICAL ADD/SUB	SHIFT BRANCH ADD/SUB	MULTIPLY	LOAD/ STORE ADD/SUB

Therefore, optimal performance depends on not only the total number of instructions but also the number of instructions required for each functional unit in VLIW DSPs.

Another important factor that affects code optimization and throughput in VLIW DSPs is instruction latency. Although an instruction can be issued to functional units every cycle, the number of cycles before the output is available is typically long due to the deeply pipelined hardware architecture. When the code does not take this latency into account, the delay can considerably offset the advantage of hardware pipelining. To reduce the negative impact of long instruction latency and also to maximally utilize multiple functional units in the VLIW

architectures, software optimization techniques such as loop unrolling and software pipelining need to be considered [4].

Code execution can become bound by recurrences as well as resources. When there is a dependency between loop iterations, the next iteration cannot start until the current iteration finishes. Scheduling instructions can also become more difficult if there are recurrence dependencies. The dependencies can make the scheduled execution of instructions take longer, offsetting the advantages of parallelism. While some dependencies are not optional, others may be introduced unnecessarily. For these, the programmer can use a qualifier in the code to alert the compiler to the presence of a false dependency, e.g., *restrict* keyword in C to avoid false memory dependencies.

Most VLIW DSPs are cluster-based in order to reduce the hardware complexity. Communication between clusters is performed via a limited number of cross-paths, which can easily become a bottleneck if instructions are not well partitioned over clusters. Due to the cluster-based architectures, it is often necessary to rewrite blocks of code in assembly in order to perform partitioning effectively. The more each cluster can be programmed for independent operation, the more these exchanges can be eliminated, thus removing a potential source of slowdowns. The advantage of effective partitioning becomes even more apparent as the code block becomes larger and more complicated.

Some DSPs are equipped with special hardware and/or instructions targeted for specific applications. An example is Galois-field arithmetic that is commonly used in communications. In some cases, it is possible to utilize the special hardware or instruction for other purposes.

Clear understanding on internal memory structure including its size, bank structure and hierarchy is as important as optimal coding in achieving optimal performance. For example, when program and data do not fit in the cache or not well aligned causing conflicts in the cache, it can result in significant slowdowns due to the cache miss penalty. In addition, the internal memory is organized as banks to reduce the complexity and implementation cost. Therefore, inappropriate accesses to the memory causing bank conflicts will also affect the performance.

4. Implementation and Discussion

The table lookup approach described in the AES proposal is very simple and requires four tables with 256 4-byte word entries in each table. Note that the four tables are utilized to avoid expensive Galois-field arithmetic. We implemented this approach on a DSP, Texas Instruments TMS320C64 that is an 8-way VLIW processor running at 600 MHz (or higher). It has a two-level internal memory hierarchy that includes a 16-kbyte level-one program cache, a 16-kbyte level-one data cache, and a 1-Mbyte level-two cache/SRAM.

The first implementation is written in C and best optimization option (-o3) is applied with other performance related options. From the assembly output from the C code, we observed that code partitioning is not optimally performed introducing many cross-path accesses. Since it is not possible to give partitioning hints to the C complier, we implemented an assembly code based on the same lookup table approach.

In the assembly LUT approach, instructions are carefully selected to minimize the numbers of cycles for the encryption loop. In addition, software pipelining is applied with proper loop unrolling to reduce the code size without scarifying the performance. We also tried to reduce the recurrence bound by replacing longer instruction chains with shorter ones. Most importantly, instructions are carefully partitioned over two computing clusters to reduce the cross-path access penalty.

While this approach is effective, the use of multiple tables may result in performance degradation if the size of the tables does not fit in or the table data cause conflicts with other data in the data cache. Since the use of multiple tables was to avoid expensive Galois-filed arithmetic, we explored another approach utilizing the Galois-field arithmetic instructions (GMPY4) available in the TMS320C64. It is important to note that this approach reduces the table size but may result in higher execution cycles. However, when this approach is combined with other application code, it may

lead to overall speedup by reducing cache miss penalty.

Table 1 lists the performance results of the three approaches for encrypting a 128-bit block with three kinds of keys: (1) LUT method in C, (2) LUT method in assembly, and (3) Galois-field arithmetic method.

With several optimization techniques introduced in this paper, we achieved a speedup factor of 1.4 with the assembly LUT code. The GMPY approach resulted in higher execution cycles. However, when it is combined with other application code, the one-fourth of the table size can potentially show a better performance.

Table 1. Performance of AES Encryption for a 128-bit Block with Three Kinds of Keys on TMS320C64.

Key Length	Number of cycles		
	LUT in C	LUT in assembly	GMPY in assembly
128-bit	314	217	304
192-bit	353	252	349
256-bit	390	281	399

As a reference, Table 2 lists the performance results on Pentium Pro, which is retrieved from the AES proposal.

Table 2. Performance of AES Encryption for a 128-bit Block with Three Kinds of Keys on Pentium Pro.

Key Length	Number of cycles	
	AES CD (ANSI C)	Brian Gladman (Visual C++)
128-bit	950	363
192-bit	1125	432
256-bit	1295	500

5. Conclusion and Future Direction

We introduced several optimization techniques for VLIW DSPs in this paper. The techniques are utilized and applied to actual implementations of the AES encryption on the TMS320C64 VLIW DSP. Despite the optimization efforts, functional units are not fully utilized mainly due to a recurrence bound. It is because each round of transformations requires the result of the previous round. This may be alleviated if multiple data blocks are simultaneously processed.

We explored another approach utilizing Galois-field arithmetic instruction available in TMS320C64. With this approach, the required table size has significantly reduced but resulted in higher execution cycles. However, this may lead to better performance in the case where cache miss overheads are significant.

References

[1] C. Su et. al, “A high-throughput low-cost AES processor,” IEEE Communications Magazine, vol. 41, no. 12, December 2003, pp. 86 – 91.

[2] S. Mangard et. al, “A highly regular and scalable AES hardware architecture,” IEEE Transaction on Computers, vol. 52, no. 4, April 2003, pp. 483 – 491.

[3] N. Sklavos et. al, “Architectures and VLSI implementations of the AES proposal Rijndael,” IEEE Transactions on Computers, vol. 51, no. 12, December 2002, pp. 1454 – 1459.

[4] K. Karadayi et. al, “Strategies for mapping algorithm to mediaprocessors for high performance,” IEEE Micro, vol. 23, no. 4, July-August 2003, pp. 58 – 70.

[5] J. Daemen and V. Rijmen, “AES Proposal: Rijndael,” http://csrc.nist.gov/CryptoToolkit/aes