How Well Are High-End DSPs Suited for the AES Algorithms? *

AES Algorithms on the TMS320C6x DSP

Thomas J. Wollinger[1], Min Wang[2], Jorge Guajardo[1], Christof Paar[1]

[1]ECE Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609, USA
Email: {wolling, guajardo, christof}@ece.wpi.edu

[2] Texas Instrument Inc.
12203 S.W. Freeway, MS 722
Stafford, TX 77477, USA
Email: minwang@micro.ti.com

Abstract

The National Institute of Standards and Technology (NIST) has announced that one of the design criteria for the Advanced Encryption Standard (AES) algorithm is the efficient implementation in hardware and software. Digital Signal Processors (DSPs) are a highly attractive option for software implementations of the AES finalists since they perform certain arithmetic operations at high speeds, they are often smaller and more energy-efficient than general purpose processors, and they are commonly used for the rapidly growing market of embedded applications. In this contribution we investigate how well modern high-end DSPs are suited for the five final candidates chosen after the second AES conference. As a result of our work we will compare the optimized implementations of the algorithms on a state-of-the-art DSP.

Keywords: cryptography, DSP, block cipher, implementation

1 Introduction

The National Institute of Standards and Technology (NIST) has initiated a process to develop a Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard (AES), specifying an encryption algorithm to replace the Data Encryption Standard (DES) which expired in 1998 [1]. NIST has solicited candidate algorithms for inclusion in AES, resulting in

*This research was supported in part through a graduate fellowship by secunet Security Networks AG and a grant from the Texas Instrument University Research Program.

fifteen official candidate algorithms of which five have been selected as finalists. Unlike DES, which was designed specifically for hardware implementation, one of the design criteria for the AES candidate algorithms is that they can be efficiently implemented in both hardware and software. Thus, NIST has announced that both hardware and software performance measurements will be included in their efficiency testing. Several earlier contributions looked into the software implementations of the AES algorithms on various platforms [2]. However, there was only one publication dealing with the implementation of the candidate algorithms on a Digital Signal Processor (DSP) [3].

Digital Signal Processors are a distinct family of micro processors. In comparison to the more common general purpose processors such as offered by, e.g., Intel and Motorola, DSPs allow for fast arithmetic, special instructions for signal processing applications, real-time capabilities, relatively lower power, and relatively lower price (obviously, those statements tend to overgeneralize and should not be taken too literally.) The main application areas of DSPs are embedded systems, such as wireless devices, cable and Digital Subscribe Line (DSL) modems, various consumer electronic devices, etc. With the predicted increase of embedded applications and persuasive computing, it is not unreasonable to expect that DSPs and DSP-like processor are becoming more commonplace. At the same time, it seems likely that many future embedded applications need some form of encryption capabilities, for instance for assuring privacy over wireless channels.

The questions that we try to address in this contribution are: How well are high-end DSPs suited for the implementation of the AES finalists? Can modern DSPs compete with general purpose computers in terms of speed?

In this paper, we focus on implementation of the five AES finalists on a Texas Instruments TMS320C6000 DSP platform. In particular, the implementations are on a 200 MHz TMS320C6201 which performs up to 1600 million instructions per second (MIPS) and provides thirty two 32-bit registers and eight independent functional units.

2 Previous Work: Cryptography on DSPs

The field of implementing cryptographic algorithms on special platforms is very active. However, the research done on implementation of cryptographic schemes on a DSP is limited. There are a few papers that deal with public-key cryptography. There is one previous paper about the implementation of the AES candidates on a DSP. The papers [4, 5, 6] deal primarily with the implementation of public key algorithms on DSP processors. The main conclusion of these papers is that DSPs are a good choice for these algorithms due to the integer arithmetic capabilities of DSPs.

Reference [5] also describes the implementation of DES on a Motorola DSP 56000. It was found that the algorithm encrypts at roughly the same speed as a contemporary PC (20 MHz Intel 80386).

Karol Gorski [3] commented on the set of the AES Round 1 candidate algorithms, based on the timings obtained on the TI TMS320C541 DSP. Reference [3] used the C implementation by Brian Gladman, compiled with full compiler level optimizations. The resulting low speeds of the algorithms were due to the 'C54x DSP's 16 bit operations which are not ideal for the most of the AES candidates. There was also no effort made to optimize the algorithms beyond those

optimizations automatically performed by the C compiler.

3 Methodology

3.1 The Implementation of the Five AES Finalists

We implemented Mars, RC6, Rijndael, Serpent and Twofish on a TMS320C6201 DSP. As the basis of the implementations we used either the reference or optimized C code provided by the algorithm's authors, or the C code written by Brian Gladman [7].

It is important to point out the way we chose to code each algorithm, because they all offer several implementation options. In [8], the authors of Rijndeal proposed a way of combining the different steps of the round transformation into a single set of table lookups. Thus, our implementation uses 4 tables with 256 4-byte word entries. Similarly, in our Twofish implementation we used the "Full Keying" option as described in the specification [9]. That means using 4 KByte tables which combine both the S-box lookups and the multiplication by the column of the MDS matrix. RC6 is a fully parameterized encryption algorithm [10]. The version of RC6 that we implemented is RC6-32/20/16. Mars was coded as stated in the algorithm specifications in [11], with 8, 16, and 8 rounds of "forward mixing", "main keyed transformation", and "backwards mixing", respectively. Finally, in [12] the authors described an efficient way to implement Serpent. Thus, we implemented the S-boxes as a sequence of logical operations which were applied to the four 32-bit input blocks.

3.2 Tools and Optimization Effort

The source code was first compiled using the standard Texas Instruments C compiler (versions 3.0 and 4.0 alpha), utilizing the highest level of optimizations (level 3) available. For further information about the levels of optimization performed by the compiling tools, see [13, page 3–2 and 3–3].

After the implementation of the C code version, we optimized the encryption and decryption functions of the algorithms so that the compiler could further optimize it. In order to do so, we took advantage of the 32-bit data bus which is capable of loading 32-bit words at a time. We performed math operations with *Intrinsic Functions* to speed up the C code. *Intrinsic Functions* are similar to an additional mathematical Run-Time Support (RTS) library. They allow the C code to access hardware capabilities of the 'C6x devices while still following ANSI C coding practices. We also tried to use as many of the functional units in parallel as possible, e.g., by replacing constant multiplication by shifts, by unrolling loops, or by preserving loops.

We further rewrote the encryption and decryption function for most algorithms in linear assembly to achieve performance improvements. Linear assembly is assembly code that has not been register-allocated and is unscheduled. The assembly optimizer assigns registers and uses loop optimization to turn linear assembly into highly parallel assembly. However, we did not program in pure assembly which is a very challenging and time consuming task on a complex processor such as the 'C6201, with eight independent functional units.

3.3 Parallel Processing: Single-Block Mode vs. Multi-Block Mode

In addition to the optimizations described above, we implemented a second version of code in which data blocks can be processed in parallel. With parallel processing, the encryption and the decryption functions can operate on more than one block at a time using the same key. This allows better utilization of the DSP's functional units which leads to better performance.

With parallel processing, however, the speedups may only be exploited in modes of operations which do not require feedback of the encrypted data, such as Electronic Code-Book (ECB) or Counter Mode. When operating in feedback modes such as Ciphertext Feedback mode, the ciphertext of one block must be available before the next block can be encrypted. For the remainder of our discussion, single-block mode will denote feedback modes and multi-block mode will denote non-feedback modes.

3.4 The TMS320C62x Digital Signal Processor

We chose the TMS320C6201 fixed point digital signal processor out of the TMS320C62x family. In this subsection we introduce the key architectural features of the DSP which are relevant for our implementation.

The 'C6201 performs up to 1600 million instructions per second (MIPS) at a clock rate of 200 MHz. These processors have 32 registers of 32-bit word length and eight independent functional units.

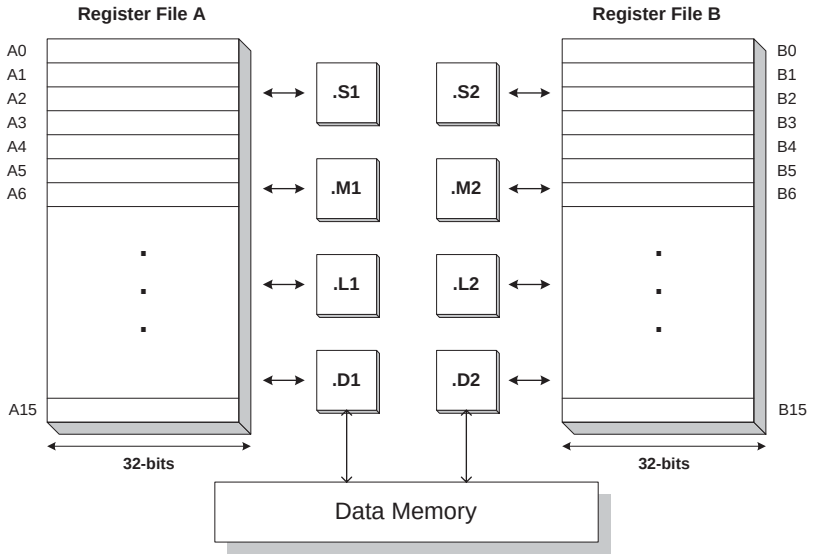

Figure 1: TMS3206x Functional Units [14]

As shown in Figure 1, the 'C6x has four pairs of functional units. The architecture of the DSP has effectively been divided in two identical halves. Each half is composed of four independent functional units (*.S*, *.M*, *.L*, and *.D*) and a bank of sixteen 32-bit registers. The processor also allows limited communication between the two halves.

The multiplier unit is indicated by *.M* and accepts two 16-bit words as an input and outputs a 32-bit result. In addition to the two multipliers, the processor provides six arithmetic logic units (ALUs). The *.L* unit, that has the ability to perform 32/40-bit arithmetic operations,

comparisons, normalization count for 32/40-bits, and 32-bit logical operations. With the *.D* unit we can add 32-bit words, subtract, do linear and circular address calculation, and write to and load from memory. The *.S* unit provides the functionality for 32-bit arithmetic operations, 32/40-bit shifts, 32-bit bit-field operations, 32-bit logical operations, branching, constant generation, and register transfers to/from the control register file [14].

The 'C6201 includes a bank of on-chip memory and a set of peripherals. Program memory consists of a 64K-byte block that is configurable as cache or memory-mapped program space. A 64K-byte block of RAM is used for data memory. The peripheral set includes two serial ports, two timers, a host port interface and an external memory interface.

The 'C6000 development environment includes: a C Compiler, an Assembly Optimizer to simplify programming and scheduling, and the Code Composer Studio^{TM}, which is a Windows debugger interface for visibility into source execution. All of the 'C6000 devices are based on the same CPU core featuring VelociTI^{TM}, a highly parallel architecture that provides software-based flexibility and good code performance for multi-channel and multi-function applications.

4 Results

All the figures presented in this section refer to a 128-bit block encryption or decryption with a key of 128 bits. The algorithms are timed with the Code Composer Simulator, which is part of the Code Composer Studio^{TM} for the TMS320C6201 DSP. Code Composer Simulator uses the simulated on-chip analysis of a DSP to gather profiling data.

The reported results in Table 1 are either C or Linear Assembly implementation. In the cases where we had the possibility to choose between two implementations we referenced the fastest results found by us. All the timings shown are obtained from a C implementation using the compiler version 4.0 alpha unless otherwise indicated.

To convert cycle counts into encryption or decryption rates expressed in bits per second, we divided $128 * 200 * 10^6$ by the cycle count. For example, the encryption speed of Twofish in multi-block mode is computed as: $128 * 200 * 10^6/184 = 139.1$Mbit/sec.

The order of the algorithms is based on the mean speed of encryption and decryption in multi-block mode. The mean speed can simply be calculated, by adding the speed of the encryption and decryption functions and then dividing the sum by two. For instance, the mean speed in multi-block mode for RC6 equals $(128.0 + 116.4)/2 = 122.2$Mbit/sec.

Here are comments about the results in Table 1:

- The highest level of optimizations were used for all algorithms, with the exception of Serpent decryption. The loop in Serpent is too complex and too long so the optimizer was only able to schedule the code in level 2. Hence, the performance figures for decryption are slightly worse than the numbers for encryption. In addition, the throughput of the decryption function is the same for single-block and multi-block modes.

- The linear assembly code of Rijndeal can be optimized by the tools very efficiently. In this case we could not gain a performance advantage by parallel processing, which results in the same speed for single-block and multi-block modes.

[*] Linear assembly implementation using compiler version 3.0

[†] C implementation using compiler version 3.0

[‡] Linear assembly implementation using compiler version 4.0 alpha

		DSP multi-block mode @ 200MHz		DSP single-block mode @ 200MHz		Pentium-Pro @ 200MHz	DSP multi-block mode/Pentium
		cycles	Mbit/sec	cycles	Mbit/sec	Mbit/sec	
Twofish	encryption	184	139.1	308	83.1	95.0 [15]	1.5
	decryption	172 †	148.8	290	88.3	95.0 [15]	1.6
RC6	encryption	200 *	128.0	282 †	90.8	97.8 [16]	1.3
	decryption	220 *	116.4	306	83.7	112.8 [7]	1.03
Rijndeal	encryption	228 ‡	112.3	228 ‡	112.3	70.5 [7]	1.6
	decryption	269 ‡	95.2	269 ‡	95.2	70.5 [7]	1.4
Mars	encryption	285	89.8	406	63.1	69.4 [7]	1.3
	decryption	280	91.4	400	64.0	68.1 [7]	1.3
Serpent	encryption	772	33.2	871 †	29.4	26.8 [7]	1.2
	decryption	917†	27.9	917 †	27.9	28.2 [7]	1.0

All the timings are obtained from a C implementation using the compiler version 4.0 alpha unless otherwise indicated

Table 1: Performance results of the AES candidates on the TMS320C6201

- In all the cases, except the encryption of RC6, we encrypted and decrypted respectively two blocks at a time in multi-block mode. We were able to process three blocks at a time in parallel for RC6 encryption. Hence, we could use a large number of functional units in parallel and could reach a high throughput. For some of the other algorithms we tried to use three blocks in parallel as well. However, the optimizer was not able to create efficient loops due to the number of instructions.

In Table 1 we compare the throughput speeds of the TMS320C6201 and a 200MHz Pentium Pro. In order to allow for an easy comparison we added the rightmost column to the table, where we divided the highest speed in multi-block on the DSP with the performance numbers on the Pentium. In this way we normalized our numbers by the speed achieved on the Pentium Pro platform. If the ratio is larger than one, the implementation of the algorithm on the DSP is faster than the one on the Pentium. One can see that in all cases but one we could achieve higher throughput on the DSP than the best known results on a Pentium Pro II which the same clock rate. Only for Serpent decryption were the Pentium and the DSP speeds almost identical.

We can also see from the performance ratio in the rightmost column how well the algorithm structure is suited for the DSP. Rijndeal encryption and the Twofish decryption gain the most when implemented on the DSP compared to the implementation on a Pentium. In both cases the quotient of the throughputs is approximately 1.5, which means that the speed of the particular function on the DSP is roughly 50% faster than the same function on the Pentium.

In addition to our above analysis, we ranked the AES finalists in order to see which one is the best in terms of speed on the 'C6000 DSP family. This ranking compares the mean speed of the algorithms in multi-block mode. Twofish with a mean speed of 144.0 Mbit/sec and RC6 with 122.2 Mbit/sec are the fastest algorithms. These two algorithms are followed by Rijndeal with a mean throughput of 103.8 Mbit/sec and Mars with 90.3 Mbit/sec. Serpent with 30.6 Mbit/sec is poor in terms of throughput on the DSP.

We would like to point out that all of our "best" results were achieved using the methodology

described above, and that other coding styles, such as pure assembly, might be able to achieve higher throughputs.

In the final version of the paper we will include one more result section which shows the performance of selected AES algorithms on a next-generation DSP.

5 Conclusions

"How well are high-end DSPs suited for the AES algorithms?" was the main question that we asked ourselves as a motivation to write this paper. The answer to this question is that in almost all cases the encryption and decryption functions of the AES finalists reach a higher speed than the best known Pentium Pro II implementations at identical clock rates of the two processors. We observed an increase of the throughput by up to over 50% compared to the Pentium. The Twofish encryption speed of 139.1 Mbit/sec and decryption of 148.8 Mbit/sec are by far the fastest throughputs that we obtained.Hence, we can conclude from our results, that state-of-the-art DSPs are well suited for the architecture of the AES finalists.

6 Acknowledgment

We would like to thank William Cammack from TI for his helpful comments.

References

[1] W. Stallings, *Cryptography and Network Security*. Upper Saddle River, New Jersey 07458: Prentice Hall, 2nd ed., 1999.

[2] "Second Advanced Encryption Standard (AES) Conference," (Rome, Italy), National Institute of Standards and Technology (NIST), March 1999.

[3] K. Gorski and M. Skalski, "Comments on the AES Candidates," tech. rep., National Institute of Standards and Technology, ENIGMA SOI Sp. z o.o., Warsaw, Poland, April 1999. `http://csrc.nist.gov/encryption/aes/round1/comments/R1comments.pdf`

[4] P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Processor," in *Advances in Cryptology - Crypto '86* (A. M. Odlyzko, ed.), vol. 263, (Berlin, Germany), pp. 311–326, Springer-Verlag, August 1986.

[5] S. R. Dussé and B. S. K. Jr., "A Cryptographic Library for the Motorola DSP56000," in *EuroCrypt '90* (Ivan B. Damgård, ed.), vol. 473 of *Lecture Notes in Computer Science*, (Berlin, Germany), pp. 230–244, Springer-Verlag, May 1990.

[6] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast Implementation of Public-Key Cryptography on a DSP TMS320C6201," in *Cryptographic Hardware and Embedded Systems* (Çetin K. Koç and C. Paar, eds.), vol. 1717 of *Lecture Notes in Computer Science*, (Berlin, Germany), pp. 61–72, Springer-Verlag, August 1999.

[7] B. Gladman, "AES Algorithm Efficiency," 2000.
http://www.btinternet.com/~brian.gladman/cryptography_technology/Aes2/index.htm

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

[9] 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, CA), 1998.

[10] R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6™ Block Cipher," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, CA), 1998.

[11] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, Stephen M. Matyas Jr., L. O'Connor, M. Peyravian, D. Safford, and N. Zunic, "Mars - a candidate cipher for AES," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, CA), 1998.

[12] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, CA), 1998.

[13] Texas Instruments Incorporated, *TMS320C6x Optimizing C Compiler User's Guide*. Owensville, Missouri: Custom Printing Company, February 1998.

[14] Texas Instruments Incorporated, *TMS320C6x/C67x Programmer's Guide*. Owensville, Missouri: Custom Printing Company, February 1998.

[15] D. Whiting, "Twofish Timing Measurements." electronic mail personal correspondence, January 2000.

[16] RSA Security, "The RC6 Block Cipher - Performance," 1999.
http://www.rsasecurity.com/rsalabs/aes/rc6_performance.html

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265