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

`The Third Advance Encryption Standard (AES3) Candidate Conference,`
April 13-14, 2000, New York, USA.

Abstract

The National Institute of Standards and Technology (NIST) has announced that one of the design criteria for the Advanced Encryption Standard (AES) algorithm was the ability to efficiently implement it 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

[*]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.

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 [14]. NIST has solicited candidate algorithms for inclusion in AES, resulting in 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 DSP contributions looked into the software implementation of the AES algorithms on various platforms [1]. However, there was only one publication dealing with the implementation of the candidate algorithms on a Digital Signal Processor (DSP) [9].

Digital Signal Processors are a distinct family of micro processors. In comparison to the more common general purpose processors such as those 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 over-generalize 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 pervasive computing, it is not unreasonable to expect that DSPs and DSP-like processors will become more commonplace. At the same time, it seems likely that many future embedded applications will need some form of encryption capability, 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 the implementation of the five AES finalists on a Texas Instruments TMS320C6000 DSP platform. In particular, the implementations are on a 200 MHz 'C62x/'C64x which performs up to 1600/8800 million instructions per second (MIPS) and provides thirty-two/sixty-four 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 [3, 7, 10] 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 [7] 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 [9] commented on the set of the AES Round 1 candidate algorithms, based on the timings obtained on the TI TMS320C541 DSP. Reference [9] 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 16 bit operations which are not ideal for 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. RC6 was also implemented on the C64x 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 [8].

It is important to point out the way we chose to code each algorithm, because they all offer several implementation options. In [6], the authors of Rijndael proposed a way of combining the different steps of the round transformation into a single set of table lookups. Each table has 256 4-byte word entries. Similarly, our Twofish implementation uses the "Full Keying" option as described in the specification [13]. In other words we used 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 [11]. The version of RC6 that we implemented is RC6-32/20/16. Mars was coded in the original version as stated in the algorithm specifications in [4], with 8, 16, and 8 rounds of "forward mixing", "main keyed transformation", and "backwards mixing", respectively. Finally, in [2] 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 [15, 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 thirty-two 32-bit registers and eight independent functional units. As shown in Figure 1, the 'C62x has four pairs of functional units. The architecture of the DSP

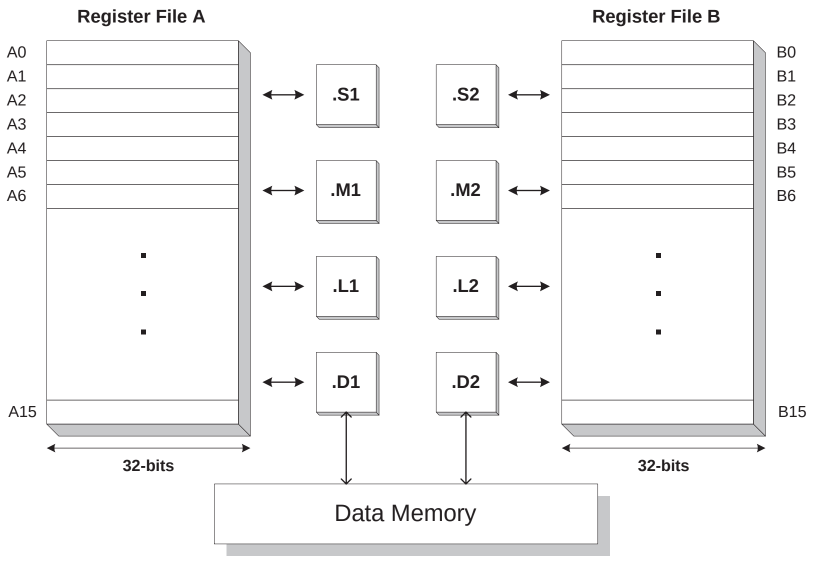

Figure 1: TMS32062x Functional Units [16]

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 [16].

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 MS 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

4.1 Results on the TMS320C6201 DSP

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 refer to either a C or a 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 a lower level. Hence, the performance figures for decryption are

[*] C implementation using compiler version 3.0

[†] Linear assembly 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 [17]	1.5
	decryption	172	148.8	290	88.3	95.0 [17]	1.6
RC6	encryption	200 †	128.0	292	87.7	97.8 [12]	1.3
	decryption	220 †	116.4	281	91.1	112.8 [8]	1.03
Rijndael	encryption	228 ‡	112.3	228 ‡	112.3	70.5 [8]	1.6
	decryption	269 ‡	95.2	269 ‡	95.2	70.5 [8]	1.4
Mars	encryption	285	89.8	406	63.1	69.4 [8]	1.3
	decryption	280	91.4	400	64.0	68.1 [8]	1.3
Serpent	encryption	772	33.2	871 *	29.4	26.8 [8]	1.2
	decryption	917*	27.9	917 *	27.9	28.2 [8]	1.0

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

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 Rijndael 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.

- In all cases, except for RC6 encryption, we encrypted and decrypted 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.

4.1.1 Results in Multi-Block Mode

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 mode 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 with 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. Rijndael encryption and 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 based on their performance 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 Rijndael 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.

4.1.2 Results in Single-Block Mode

The results stated above refer only to the cases in which we used multi-block mode. If we look at the single-block mode case, Rijndael encryption and decryption as well as Serpent encryption perform better on the DSP than on a Pentium. Rijndael encryption with 112.3 Mbit/sec is almost 60% faster than the corresponding Pentium implementation and Rijndael decryption at 95.2 Mbit/sec is almost 40% faster. Judged by their speed performance on the C62x, Serpent decryption, Mars encryption and decryption, and Twofish decryption are slightly worse than on a general-purpose computer. The remaining functions, Twofish encryption and RC6 encryption and decryption, are much slower than the corresponding Pentium functions.
If we had ranked the algorithms based on their mean speed in single-block mode, Rijndael with 103.8 Mbit/sec would be the fastest, followed by RC6 with 89.4 Mbit/sec, and Twofish with 85.7 Mbit/sec. Mars with 63.6 Mbit/sec and Serpent with 28.7 Mbit/sec are not as good in single-block mode.

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.

4.1.3 Comparison of the Results with the Critical Path of the Algorithms

Craig S.K. Clapp analyzes the critical path of Crypton, E2, and the five AES finalists. In his analysis, [5] only counts instructions and cycles associated with the transformation of a plaintext block into a ciphertext block in ECB mode. In other words, instructions associated with loading of plaintext, storing of ciphertext, and loop overhead are ignored. Clapp concludes that based on the length of its critical path, Rijndael stands well ahead of the pack with 71 cycles/block. Twofish (162 cycles/block), RC6 (encryption with 181 cycles/block and decryption with 161 cycles/block), and Mars (214 cycles/block) form the second tier. Finally, Serpent's critical path is a factor of two longer than the next nearest candidate (encryption with $\leq$ 526 cycles/block and decryption with $\leq$ 436 cycles/block).

The results that we achieved in single-block mode are in agreement with those obtained by analyzing the critical path. Rijndael is in both cases by far the fastest algorithm. The throughput of RC6 is slightly better than the throughput of Twofish on the DSP, even though the critical path of Twofish is a little shorter than the one from RC6. Mars is ranked in both, the DSP speed analysis and the critical path analysis of [5], the same. Serpent results trail the nearest candidate in both analyses by more than a factor of two. It is important to point out that while the critical path for decryption is shorter than that for encryption in Serpent, decryption is actually slower than encryption in the DSP implementation.

The discrepancies are due to our use of automatic optimization. The optimizer tries to create the best machine code possible. Nevertheless, the optimizer might not be able to reach the cycle count of the critical path for some algorithms. We might be able to overcome these differences by rewriting the functions in full assembly. We were not able to do this because of time constraints.

4.1.4 Memory Usage

Embedded system applications have often memory constrains. Hence this subsection looks at the memory requirements of our implementation. The 'C6201 has three 16 Mbyte regions of external memory. These regions can support synchronous or asynchronous 32-bit access. There is also one 4 Mbyte region of asynchronous external memory which is typically used to store the boot information. The 'C6201 contains one megabit of internal RAM which is split between program and data memory. All this internal memory is zero wait-state. Table 2 summarizes the memory usage of the algorithms in our implementation.

		Memory Usage multi-block mode		Memory Usage single-block mode	
		Data ROM /Bytes	Program /Bytes	Data ROM /Bytes	Program /Bytes
Mars		3072		3072	
	encryption		3280		2428
	decryption		2956		2372
RC6		0		0	
	encryption		608		576
	decryption		672		576
Rijndael		16384		16384	
	encryption		2360		1180
	decryption		2960		1480
Serpent		0		0	
	encryption		5844		3568
	decryption		6016		5104
Twofish		168		168	
	encryption		1416		700
	decryption		1420		708

Table 2: Memory Usage on the TMS320C6201

As it can be seen from Table 2, the memory usage of the algorithms varies almost by an order of magnitude. RC6 uses the least program memory and Serpent the most. In some cases, e.g. for Serpent, the algorithms require a large amount of program memory, because we optimized them for speed. Hence we calculated the look-up tables on the "fly" with boolean-algebra and this increases the program code. The data ROM represents constant arrays, which in our cases correspond to the look-up tables. RC6, for example, uses no tables, hence the data ROM is zero.

4.2 Results on the TMS320C64x

The TMS320C64x clock can be scaled to up to 1.1 GHz and can perform up to 8800 MIPS. The C64x has extended parallelism support with quad 8-bit and dual 16-bit operations. Also, the sixty-four 32-bit registers and 8 functional units lead to better performance. We also took advantage in our implementation of the better data access and the extended instruction set of the C64x (for example, rotation, Galois field multiplication, etc.).

We chose RC6 to be implemented on the C64x. The results that we present in this section are based on a C implementation and are compiled with compiler version 4.0 beta.

The results in Table 3 for RC6 achieved with the 'C64x in multi- and single-block mode are better than the results we got from the 'C6201. RC6 encryption in multi-block mode is almost 70% faster than on a general-purpose machine.

At this point it is important to remark that the optimizer tools are quite advanced for the 'C62x, but are still in a very early stage for the 'C64x. That means if we only perform C code optimizations, we will not get good performance numbers on the 'C64x. We expect an improvement when we rewrite the functions in linear assembly. We did a detailed analysis for hand coded assembly RC6 and we estimated a performance of 229 cycles/block (for each encryption- and decryption-function) in single-block mode.

		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	
RC6	encryption	155	165.2	277	92.4	97.8 [12]	1.7
	decryption	154	166.2	278	92.1	112.8 [8]	1.5

Table 3: Performance results of RC6 on the TMS320C64x

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. We noticed that in almost all cases the AES finalists' encryption and decryption functions reach higher speeds on the 'C6000 DSPs than the best known Pentium Pro II implementations, at identical clock rates. It was observed that some of our implementations on the 'C6201 were over 50% faster than the best known performance numbers on the Pentium platform. In addition, our implementation of RC6 on the 'C64x reached speeds which were almost 70% faster than those of the Pentium. RC6 on the 'C64x encrypts with a throughput of 165.2 Mbit/sec and decrypts with a speed of 166.2 Mbit/sec. Twofish with an encryption speed of 139.1 Mbit/sec and decryption of 148.8 Mbit/sec was by far the fastest throughput that we obtained on the 'C6201. 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] Second Advanced Encryption Standard (AES) Conference. Rome, Italy, March 1999. National Institute of Standards and Technology (NIST).

[2] 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.

[3] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Processor. In A. M. Odlyzko, editor, *Advances in Cryptology - Crypto '86*, volume 263, pages 311–326, Berlin, Germany, August 1986. Springer-Verlag.

[4] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, and Nevenko Zunic. Mars - a candidate cipher for AES. In *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[5] Craig S.K. Clapp. Instruction-level Parallelism in AES Candidates. Second AES Conference, March 1999. `http://csrc.nist.gov/encryption/aes/reound1/conf2/papers/clapp.pdf`

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

[7] Stephen R. Dussé and Burton S. Kaliski Jr. A Cryptographic Library for the Motorola DSP56000. In Ivan B. Damgård, editor, *EuroCrypt '90*, volume 473 of *Lecture Notes in Computer Science*, pages 230–244, Berlin, Germany, May 1990. Springer-Verlag.

[8] Brian Gladman. AES Algorithm Efficiency, 2000.
`http://www.btinternet.com/~brian.gladman/cryptography_technology/Aes2/index.htm`

[9] Karol Gorski and Michal Skalski. Comments on the AES Candidates. Technical report, 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`

[10] Kouichi Itoh, Masahiko Takenaka, Naoya Torii, Syouji Temma, and Yasashi Kurihara. Fast Implementation of Public-Key Cryptography on a DSP TMS320C6201. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, pages 61–72, Berlin, Germany, August 1999. Springer-Verlag.

[11] R. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6$^{\text{TM}}$ Block Cipher. In *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

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

[13] 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.

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

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

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

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