Application Report
SPRA702 - November 2000

Data Encryption Standard (DES) Implementation on the TMS320C6000

R. Stephen Preissig *C6000 Applications*

ABSTRACT

This application report studies the implementation of the Data Encryption Standard (DES) on the TMS320C6000™ family of processors. C source code from the public domain was optimized using the version 3.01 Optimizing C Compiler. Data rates on the C6201 (200 MHz) are measured at 53 Mbits per second for DES and 22 Mbits per second for triple-DES. Data rates on the C6211 (150 MHz) device are measured at 39 Mbits per second for DES and 18 Mbits per second for triple-DES. This application note details the feedback generated by the Optimizing C Compiler and the corresponding methods used to optimize DES for the C6201 and C6211. Discussion of the structure of the DES algorithm is provided only as necessary to elucidate the optimization methods employed, with reference to more detailed descriptions.

Contents

List of Figures

List of Tables

1 Introduction

Security is a prevalent concern in information and data systems of all types. Historically, military and national security issues drove the need for secure communications. Recently, security issues have pervaded the business and private sectors. E-commerce has driven the need for secure internet communications. Many businesses have firewalls to protect internal corporate information from competitors. In the private sector, personal privacy is a growing concern. Products are available to scramble both e-mail and telephone communications.

One means of providing security in communications is through encryption. By encryption, data is transformed in a way that it is rendered unrecognizable. Only by decryption can this data be recovered. Ostensibly, the process of decryption can only be performed correctly by the intended recipient(s). The validity of this assertion determines the “strength” or “security” of the encryption scheme.

Many communications products incorporate encryption as a feature to provide security. This application report studies the implementation of one of the most historically famous and widely implemented encryption algorithms, the Data Encryption Standard (DES). Implementation of DES is studied on the Texas Instruments TMS320C6000 family of processors.

Since the C6000 family is the DSP industry’s performance leader, it efficiently implements not only DES, but the mathematically intensive communications algorithms for which DES is often a complement. Many communications solutions are able to upgrade their products to incorporate an encryption feature while maintaining the same channel density of the original solution. In some cases it is necessary to upgrade to one of the higher performing devices in the constantly growing C6000 family, but in lower data rate applications, it is possible to add encryption on the same processor which is already used. This allows the addition of encryption functionality with no extra manufacturing cost, and no increase in the board space required.

The TMS320C6000 family provides an excellent platform for encryption. A primary concern in encryption applications is the broad and rapidly changing field of encryption standards. Since it is a re-programmable device, the C6000 can be programmed with a library of encryption standards. If an unknown encryption standard is encountered, the new standard may be downloaded and added to this library.

The re-programmability is augmented by the C6000 family's industry leading compiler tools. Emerging encryption standards may be rapidly integrated. As the following application report demonstrates, C code may be optimized quickly and efficiently using the compiler tools' optimization feedback. This is especially useful because many encryption algorithms are available, downloadable for free as C code, from the public domain.

1.1 The Data Encryption Standard (DES)

Developed in 1974 by IBM in cooperation with the National Securities Agency (NSA), DES has been the worldwide encryption standard for more than 20 years. For these 20 years it has held up against cryptanalysis remarkably well and is still secure against all but possibly the most powerful of adversaries [4]. Because of its prevalence throughout the encryption market, DES is an excellent interoperability standard between different encryption equipment.

The predominant weakness of DES is its 56-bit key which, more than sufficient for the time period in which it was developed, has become insufficient to protect against brute-force attack by modern computers [5] (see table 1). As a result of the need for a greater encryption strength, DES evolved into triple-DES. Triple-DES encrypts using three 56-bit keys, for an encryption strength equivalent to a 168-bit key. This implementation, however, requires three times as many rounds for encryption and decryption and highlights a second weakness of DES – speed. DES was developed for implementation on hardware, and DES implementations in software are often less efficient than other standards which have been developed with software performance in mind.

As is highlighted in the Results section of this application report, however, the C6000 core is able to achieve very impressive data rates for DES and triple-DES in software. Data rates on the C6201 (200 MHz) are measured as high as 53 Mbits per second for DES and 22 Mbits per second for triple-DES. Data rates on the C6211 (150 MHz) device are measured as high as 39 Mbits per second for DES and 18 Mbits per second for triple-DES. On a 300 MHz device such as the C6203, we can extrapolate data rates of 80 Mbits per second for DES and 33 Mbits per second for triple-DES.

This performance is typically more than sufficient for multichannel applications involving 8 Kbit per second vocoders and/or 56 Kbytes per second modems. Even DES encryption at the full G.lite ADSL rate of 1.5 Mbytes per second requires less than 50 MHz performance on the C6000 core and could be added to an application with a simple upgrade from C6201 to the C6202 or from the C6202 to the C6203. Due to the C6000 platform's strong performance in multi-channel communications applications, DES may be implemented as part of a one-chip re-programmable multi-channel solution. This provides great savings in board space, cost, power consumption and design time. Furthermore, because the solution is re-programmable, it may be easily evolved to match the rapidly changing encryption market.

Table 1. Key Strength versus Modern Computers

Type of Attacker	Budget	Time per key recovered, 40 bits	Time per key recovered, 50 bits
Pedestrian hacker	$400	1 week	infeasible
Small Business	$10,000	12 minutes	556 days
Corporate Department	$300,000	0.18 seconds	3 hours
Big Company	$10,000,000	0.005 seconds	6 minutes
Intelligence Agency	$300,000,000	0.002 seconds	12 seconds

1.2 Modes of Operation for Encryption Algorithms

DES belongs to a category of ciphers called block ciphers. Block ciphers, as opposed to stream ciphers, encrypt messages by separating them into blocks and encrypting each block separately. Stream ciphers, on the other hand, operate on streams of data one bit at a time as a continuous stream.

DES encrypts 64-bit blocks of plaintext into 64-bit blocks of ciphertext. Plaintext, used in the context of cryptography, is the name commonly given to the body of a message before it is encrypted, i.e. the unaltered text of the message which is to be sent. Likewise, ciphertext is the name commonly given to the encrypted version of the message body which is meant to be indecipherable to any person who does not have the decryption key.

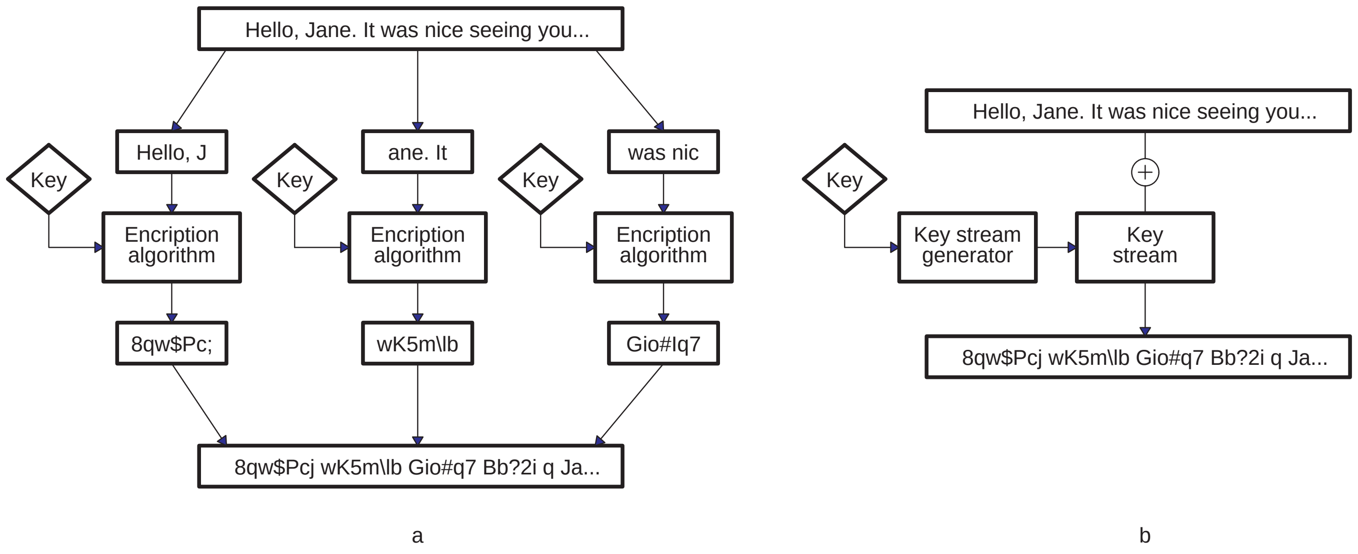

Hello, Jane. It was nice seeing you...
Hello, J
ane. It
was nic
Key
Key
Key
Encription algorithm
Encription algorithm
Encription algorithm
8qw$Pc;
wK5m\lb
Gio#lq7
8qw$Pcj wK5m\lb Gio#q7 Bb?2i q Ja...
a
Hello, Jane. It was nice seeing you...
Key
Key stream generator
Key stream
8qw$Pcj wK5m\lb Gio#q7 Bb?2i q Ja...
b

Figure 1. Types of Ciphers: a) Block Cipher, b) Stream Cipher

The simplest implementation of a block cipher is to separate the plaintext into contiguous blocks, encrypt each block into ciphertext blocks, and group these ciphertext blocks together as the ciphertext output. This mode of operation is referred to as Electronic Code Book (ECB) mode (see Figure 2A). The distinguishing property of this mode is that identical blocks of plaintext always encrypt to the same ciphertext. This is undesirable in some applications.

It is possible to introduce feedback between blocks by feeding the results of the previous encryption block into the input of the current block. The first block of feedback is a randomly generated block called the initialization vector. When this is done, each ciphertext block is not only dependent on the plaintext block that generated it but on each of the preceeding blocks as well, including the initialization vector. Identical plaintext blocks will now only encrypt to the same ciphertext block if each of the proceeding blocks are identical and the initialization vectors are identical. This mode of operation is referred to as Cipher Block Chaining (CBC) mode (See Figure 2B).

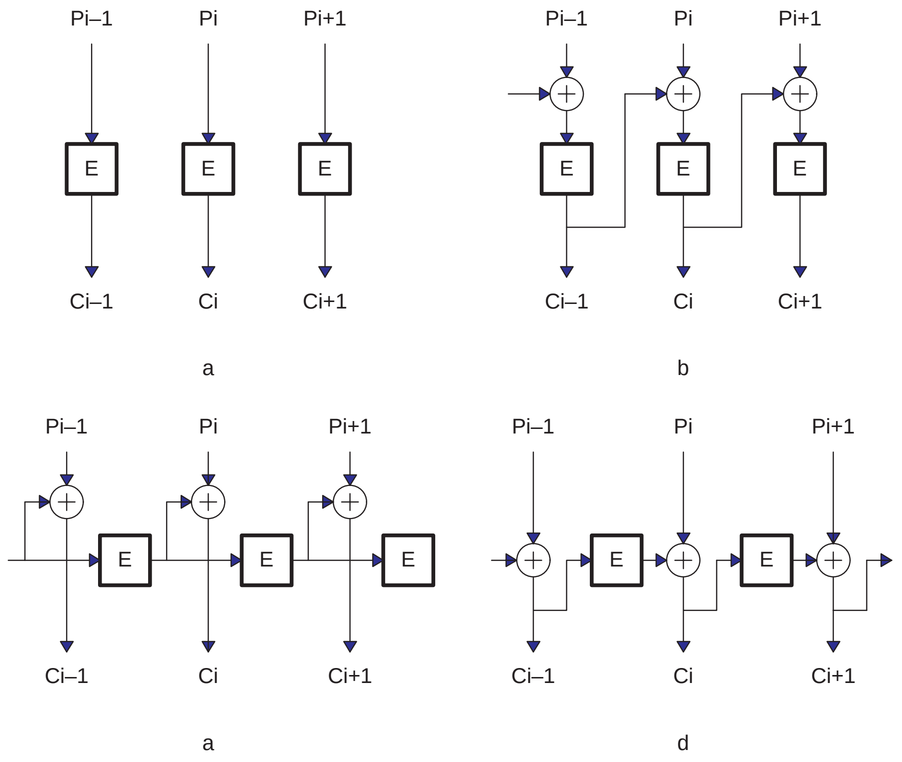


Figure 2. Modes of Operation for Block Ciphers: a) Electronic Code Book, b) Cipher Block Chaining, c) Ouput-Feedback, d) Cipher-Feedback

Other modes of operation exist as well. Two other common modes, Output Feed Back (OFB) mode and Cipher Feed Back (CFB) modes use a block encryption algorithm to generate a stream cipher (see Figure 3C,D). Only ECB and CBC modes are examined in this application report. For details on other modes of operation, as well as a more in-dept discussion of the ECB and CBC modes of operation, see reference [4], Bruce Schneider’s “Applied Cryptography”.

1.3 Implementation of DES

DES relies upon the encryption techniques of confusion and diffusion. Confusion is accomplished through substitution. Specially chosen sections of data are substituted for corresponding sections from the original data. The choice of the substituted data is based upon the key and the original plaintext. Diffusion is accomplished through permutation. The data is permuted by rearranging the order of the various sections. These permutations, like the substitutions, are based upon the key and the original plaintext.

The substitutions and permutations are specified by the DES algorithm. Chosen sections of the key and the data are manipulated mathematically and then used as the input to a look-up table. In DES these tables are called the S-boxes and the P-boxes, for the substitution tables and the permutation tables, respectively. In software these look-up tables are realized as arrays and key/data input is used as the index to the array. Usually the S- and P-boxes are combined so that the substitution and following permutation for each round can be done with a single look-up.

In order to calculate the inputs to the S- and P-box arrays, portions of the data are XORed with portions of the key. One of the 32-bit halves of the 64-bit data and the 56-bit key are used. Because the key is longer than the data half, the 32-bit data half is sent through an expansion permutation which rearranges its bits, repeating certain bits, to form a 48-bit product. Similarly the 56-bit key undergoes a compression permutation which rearranges its bits, discarding certain bits, to form a 48-bit product. The S- and P-box look-ups and the calculations upon the key and data which generate the inputs to these table look-ups constitute a single round of DES (see Figure 3B).

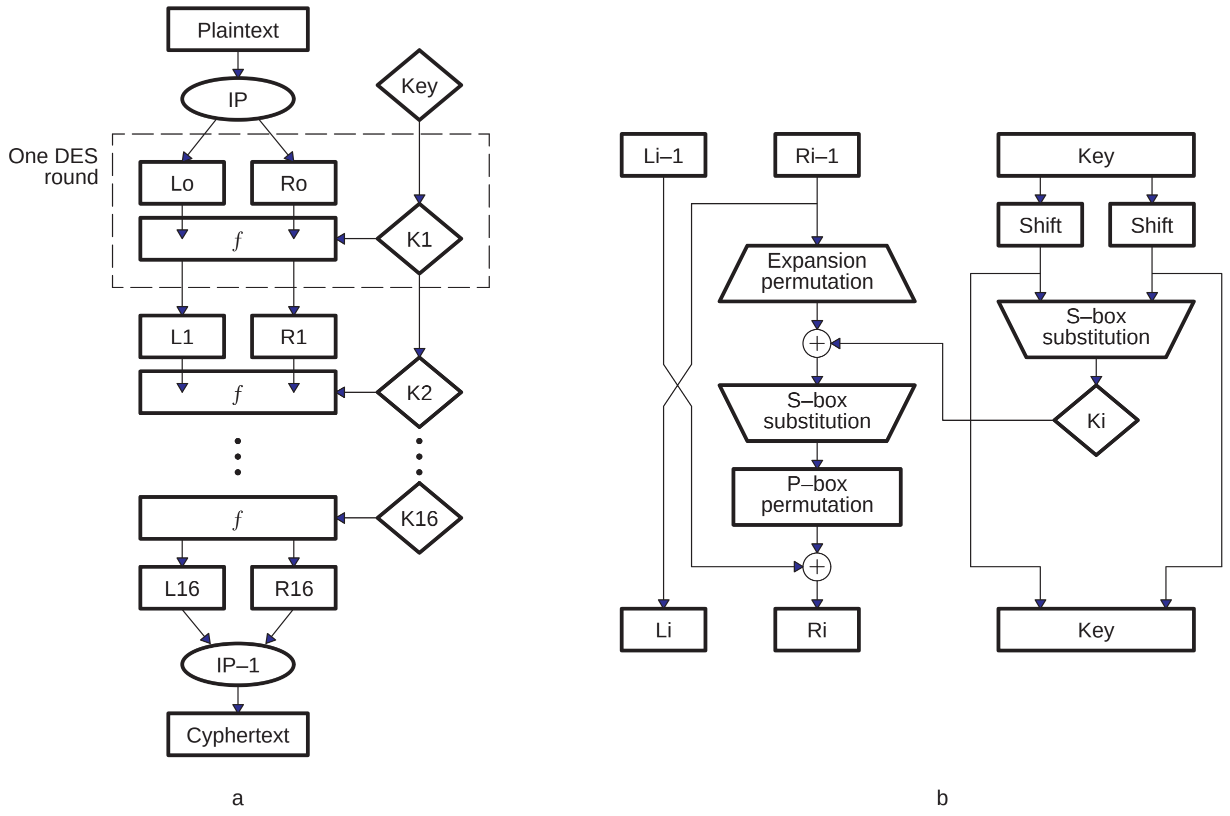


Figure 3. a) DES Core Algorithm, b) Single Round Expanded

This same process of S- and P-box substitution and permutation is repeated sixteen times, forming the sixteen rounds of the DES algorithm (see Figure 3A). There are also initial and final permutations which occur before and after the sixteen rounds. These initial and final permutations exist for historical reasons dealing with implementation on hardware and do not improve the security of the algorithm. For this reason they are sometimes left out of implementations of DES. They are, however, included in this analysis as they are part of the technical definition of DES.

As is evident from the description above, DES is not a typical digital signal processing (DSP) application. DSP applications tend to rely upon the sum of products as their core operation. DES relies primarily on bit manipulation operations and actually uses no multiplication operations. None the less, as will be evinced later in this application report, the C6000 platform is able to perform the DES algorithm with high efficiency.

It is possible to go into much greater detail on the inner workings of the DES algorithm, as is done in the following references: [1],[2],[3],[4]. The purpose of this application report is to investigate the performance of DES on the C6000 platform using C code which is compiled by Texas Instruments' Optimizing C Compiler. Thus, we only go into the level of detail that is need as background to the investigations presented later in this application report.

2 Methods

This application report investigates the performance of the DES algorithm on the C6000 Digital Signal Processing platform using C code compiled on the Texas Instruments' Optimizing C Compiler. DES C code was obtained from the public domain in a library compiled by Eric Young [6]. This code was optimized in C (no assembly code is used), using feedback from the Optimizing C Compiler. Reference [7], the TMS320C6000 Compiler Optimization Tutorial, outlines the use of the Optimizing C Compiler and how the feedback from the compiler is used to improve performance of C code.

The following are the version numbers and part numbers of each of the tools used to generate the results of this application report:

Code Composer Studio, version 1.0	– TMDX3246856-07
Compiler tools, version 3.01	– TMDS3246555-07
C62xx fast simulator, little endian	– Included with CCS, v. 1.0 (TMDX3246856-07)
C6211 DSP Starter Kit (DSK)	– TMDX320006211
C6201 Evaluation Module (EVM)	– TMDS3260A6201

2.1 Configuration

This code is derived from an encryption library created by Eric Young. The files have been modified, primarily for readability. The files have also been modified to improve performance, using feedback from the Optimizing C Compiler, as is outlined in the sections to follow.

All code was compiled using the following compiler flags:

- –o3
- –pm
- –oi0
- –mx
- –k
- –mw

Table 2. Explanation of Compiler Options Used

Compiler Flag	Description
–o3	Represents the highest level of optimization available. Various loop optimizations are performed such as software pipelining, unrolling, SIMD.
–pm	Combines source files to perform program-level optimization.
–oi0	Disables all automatic size-controlling inlining (which is enabled by –o3).
–mx	Tries multiple software pipeline heuristics and chooses the best one.
–k	Instructs the compiler to keep the assembly file, and to allow analysis of the compiler feedback (see –mw).
–mw	Produces compiler feedback.

These are the compiler flags recommended by the *C6000 Compiler Optimization Tutorial* [7] with one exception. The “C6000 Compiler Optimization Tutorial” recommends the use of the compiler flag –op2. This flag specifies that the module contains no functions or variables that are called or modified from outside the source code provided to the compiler. In this example, the source code was separated into multiple files for compilation speed and ease of use. For this reason, there were global variables which were modified between files, so the –op2 option was not appropriate.

The compiled functions were benchmarked using statistics objects from DSP/BIOS package which is a part of Code Composer Studio. These results were confirmed both by using profile points, a second option available in Code Composer Studio, and using the clock functions included in the <time.h> library. All three methods produced nearly identical results. It should be noted that the <time.h> library is an older development and not meant for use with Code Composer Studio. If this library is used to benchmark functions, the program should be run from a DOS environment. For more explanation of Statistics Object, Profile Points, and the Code Composer Studio Integrated Development Environment, please refer to the Code Composer Studio User’s Guide [11] and the TMS320C6000 DSP/BIOS User’s Guide [12].

2.2 Core Algorithm Optimization Methods

The C code which was obtained from the public domain produces the following output from the optimizing C compiler when compiled using the above outlined procedure. The C code which generated this output is given in the zip file, DES.zip. The name of the subroutine is “des_encrypt_core(…).”

Figure 4 shows the feedback from the optimizing C compiler on the des_encrypt_core() function. The trip count is the number of times the loop will execute. In this case, DES has exactly 16 rounds in every instance (it is part of the definition of DES) and therefore, both the Known Minimum Trip Count and the Know Maximum Trip Count are 16.

The Loop Carried Dependency bound represents the largest loop carry path. A loop carry path occurs when one iteration of the loop writes a value that must be read in a future iteration. In the DES core algorithm, the 64-bit block is separated into two 32-bit variables, denoted in the C code as “l” and “r,” for “left” and “right” variables, respectively. Both variables, “l” and “r,” are modified in each iteration of the loop, thus, all operations upon them from one iteration of the loop must be completed before the next iteration of the loop may begin. The “r” variable has a longer loop carry path and therefore determines the Loop Carried Dependency bound of 15, as shown in Figure 5. The instructions in the bubbles of Figure 5 map directly to the corresponding instruction of the original C code, as given in the zip file, DES.zip. The name of the subroutine is “des_encrypt_core(…).”

The Loop Carried Dependency bound of 15 severely limits the performance of this system. As we see from the output above, the partitioned resource bound is only 7. The unpartitioned resource bound and partitioned resource bounds indicate the minimum number of cycles per loop required based solely upon the resources available on chip. The unpartitioned resource bound is measured before the variables are partitioned between the two banks of 16 registers and the operations are correspondingly partitioned between the two banks of .L, .S, .D and .M units. The partitioned resource bound of 7 indicates that the C6000 has the resources to do the 36 assembly instructions of this loop in 7 cycles.

This means that if the Loop Carried Dependency did not exist, the compiler might be able to find a schedule of as few as 7 cycles per loop iteration. Because the Loop Carried Dependency bound does exist, however, the compiler is forced to start looking for schedules of 15 cycles per loop iteration and actually finds the schedule for 16 cycles per loop iteration.

```
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Known Minimum Trip Count         : 16
;*      Known Maximum Trip Count         : 16
;*      Known Max Trip Count Factor      : 16
;*      Loop Carried Dependency Bound(^) : 15
;*      Unpartitioned Resource Bound     : 6
;*      Partitioned Resource Bound(*)    : 7
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                    0        0
;*      .S units                    5        6
;*      .D units                    6        4
;*      .M units                    0        0
;*      .X cross paths              1        4
;*      .T address paths            6        4
;*      Long read paths             0        0
;*      Long write paths            0        0
;*      Logical  ops (.LS)          6        7     (.L or .S unit)
;*      Addition ops (.LSD)         1        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)            6        7*
;*      Bound(.L .S .D .LS .LSD)    6        6
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 15 Did not find schedule
;*         ii = 16 Schedule found with 2 iterations in parallel
;*      Done
;*
```

Figure 4. Compiler Feedback for DES Core Algorithm, 1 Channel

The remaining section of the compiler feedback gives a detailed analysis of the partitioning of the .L, .S, .D and .M functional units in the C6000 core. This is also very important feedback, but as shown in the paragraph above, is overshadowed in this case by the loop carry path. Thus, the loop carry path obstacle will be solved first, and then this next section of compiler feedback will be discussed.

The loop carry path is a property of the DES algorithm and cannot be modified; however, with the provision of this information from the optimizing compiler, it is possible to modify the code in order to remove the adverse effect of the loop carry path. The loop carry path is lowering performance because it limits the number of operations that can be done each cycle. Because the C6000 core is capable of doing 8 operations per cycle, the DES algorithm, in its current form, does not take advantage of all of the C6000 core's computational resources.

In order to take advantage of the C6000 core's computational resources, the code is modified to encrypt three 64-bit blocks in parallel. The modified code is provided in the zip file, DES.zip. The name of the subroutine is "des_encrypt_core_3ch(…)." The loop carry path for each block is still 15 cycles, however, because each block is encrypted independently, these loop carry paths are independent, and the total loop carry path for the three encryptions in parallel is still 15. So what has doing three encryptions in parallel gained? Examination of the optimizing C compiler's output will elucidate (see Figure 6).

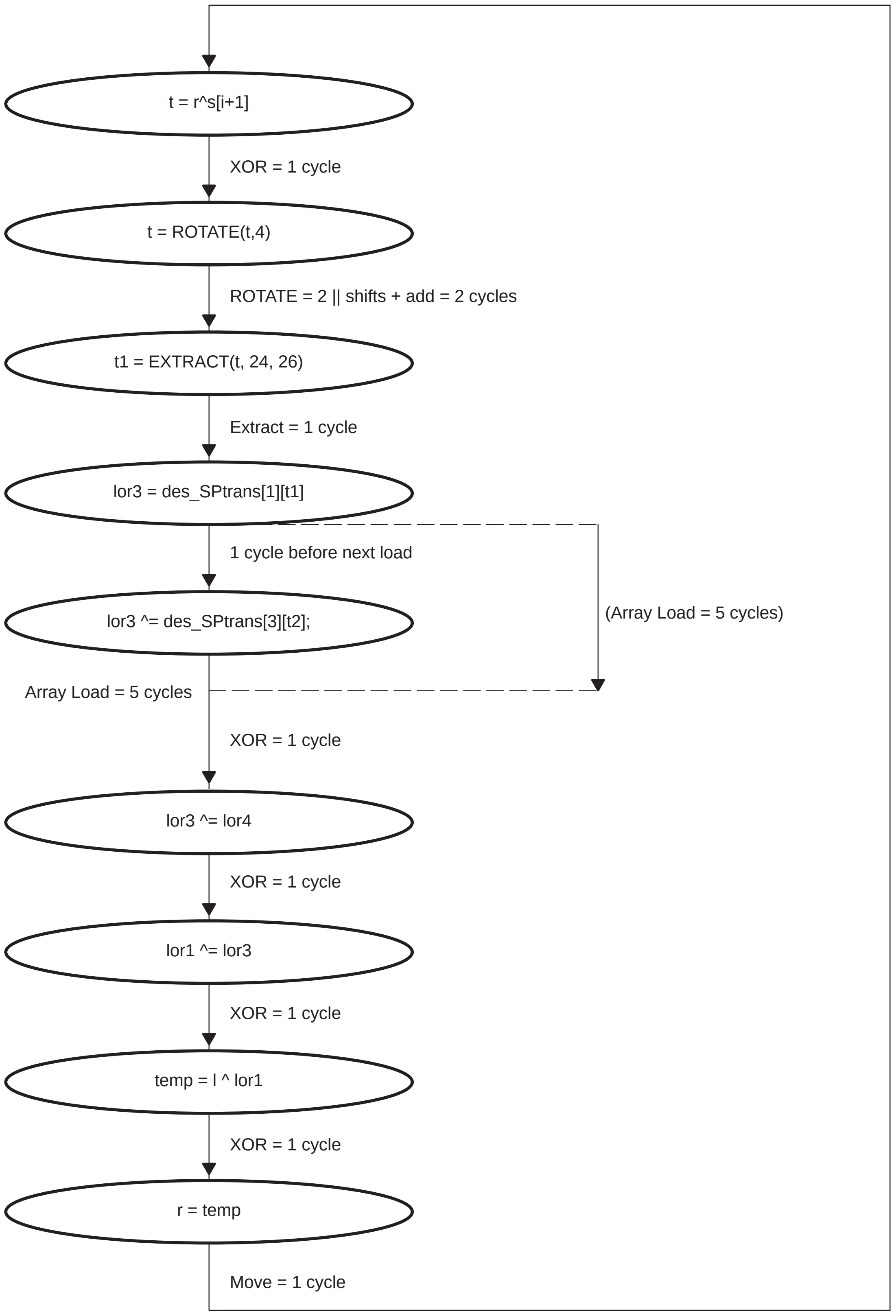
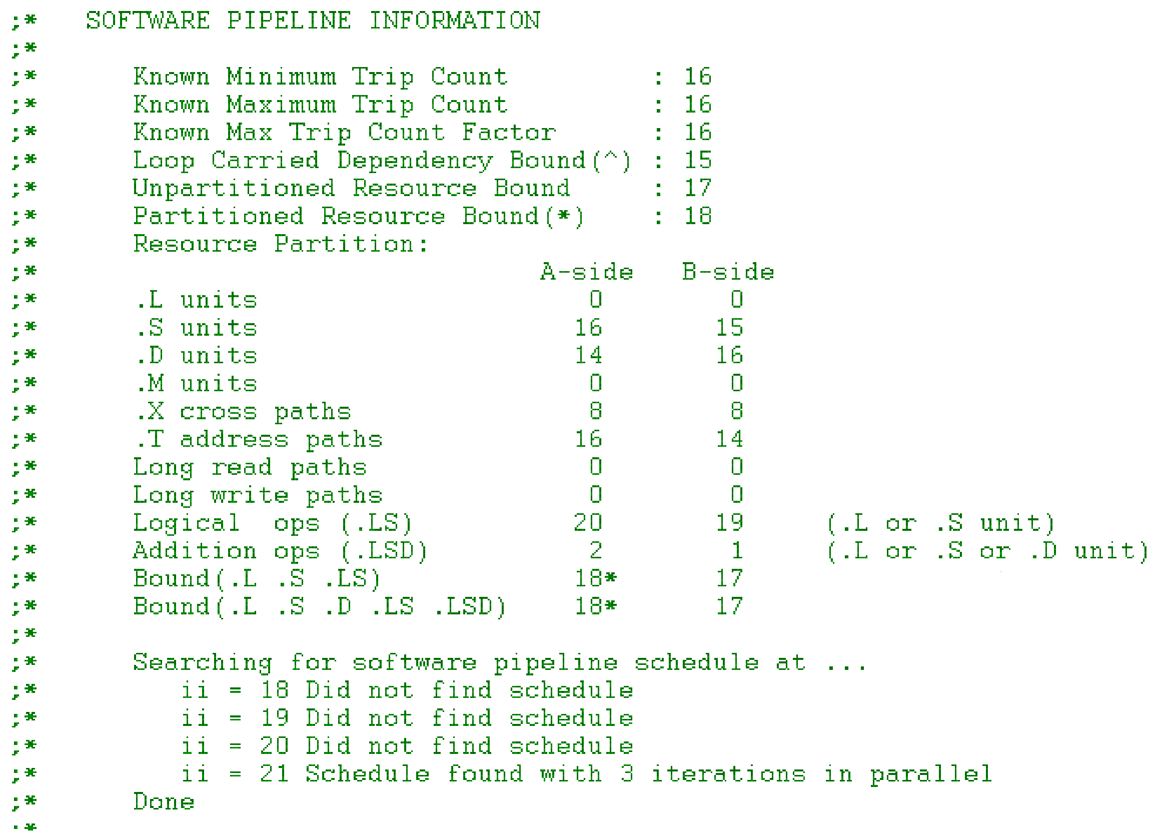


Figure 5. Loop Carry Path for DES Core Algorithm



```
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Known Minimum Trip Count         : 16
;*      Known Maximum Trip Count         : 16
;*      Known Max Trip Count Factor      : 16
;*      Loop Carried Dependency Bound(^) : 15
;*      Unpartitioned Resource Bound     : 17
;*      Partitioned Resource Bound(*)    : 18
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                    16       15
;*      .D units                    14       16
;*      .M units                     0        0
;*      .X cross paths               8        8
;*      .T address paths            16       14
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)          20       19     (.L or .S unit)
;*      Addition ops (.LSD)          2        1     (.L or .S or .D unit)
;*      Bound(.L .S .LS)            18*      17
;*      Bound(.L .S .D .LS .LSD)    18*      17
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 18 Did not find schedule
;*         ii = 19 Did not find schedule
;*         ii = 20 Did not find schedule
;*         ii = 21 Schedule found with 3 iterations in parallel
;*      Done
;*
```

Figure 6. Compiler Feedback for DES Core Algorithm, 3 Channel

As before, the known minimum and maximum trip counts are 16, and the known max trip count factor is 16. Also, the loop carried dependency bound remains unchanged at 15 cycles. However, because three 64-bit blocks of data are being encrypted instead of just one, the unpartitioned resource bound has correspondingly tripled from 6 to 17. This is reflective of the fact that there are now almost three times as many instructions that must be issued per loop, so the loop now requires three times the on-chip resources. (The reason the resource requirement has not exactly tripled is because the loop overhead, i.e. loop counter incrementing and the branch to the beginning of the loop, has not increased.)

Now the loop carried dependency bound is no longer limiting the efficiency of this loop. Correspondingly, the loop is now performing 3 64-bit encryption operations in 21 cycles per loop iteration (7 cycles per loop iteration per block) instead of 1 64-bit encryption operation in 16 cycles per loop iteration (16 cycles per loop iteration per block). With one simple modification performance is more than doubled.

Examination of the detailed resource analysis shows that this loop contains 31 commands which must be performed on the .S units; 30 commands which must be performed on the .D units; 39 commands which may be performed on either the .L units or the .S units; and 3 commands which may be performed on the .L units, .S units or .D units. (The compiler also breaks out this information between the A and B sides, but since the partitioning is even, this is not discussed here.) From this information, the compiler calculates the (.L .S .LS) resource bound and the (.L .S .D .LS .LSD) resource bounds. Both of these bounds are 18 on the A side and 17 on the B side. This is the best case partitioning because the resources have been partitioned evenly.

Evident from the above figure, however, is that the .M units are not utilized. In fact, the .M units are not used a single time in this loop. This is because the .M units are multipliers and do not perform any operations other than multiplications. Because DES does not use any multiplications, these units go unused.

Programmers who are using hand coded assembly might attempt to make use of the .M units in order to perform the 9 move instructions and the 3 left shifts that occur in this loop. The .M units are used to multiply by 1, the result is the equivalent of a two cycle move. Likewise, if the .M units are used to multiply by 2, 4, 8, etc., the equivalent of two-cycle left shifts of 1, 2, 3, etc. result.

However, using this approach will increase the loop carried dependency bound because the multiplication operations of the .M units require two cycles apiece, whereas normally instituted move and shift operations require only one. Six of the moves contribute to the loop carried dependency bound as do all three of the shifts. It may be possible, however, to utilize the .M units to perform the three moves which do not contribute to the loop carried dependency bound as well as three of the remaining nine operations. This would increase the loop carried dependency bound from 15 to 16, and would decrease the unpartitioned resource bound from 17 to 16. Thus, it is theoretically possible (though highly unlikely) that a hand coded assembly approach could reduce the iteration interval from 21 to 16 cycles.

Thus, using the iteration interval from the optimizing C compiler feedback (ii = 21 schedule found), it is determined that the C code for this loop is compiled to greater than 76% of the theoretical performance limit compared to the 16 cycle theoretical minimum. In actuality it is likely that even hand coded assembly could not achieve 16 cycles per loop iteration as partitioning and scheduling issues are likely to arise. The optimizing C compiler is most probably operating at much better than 76% of the hand coded assembly efficiency, likely at 86% or 90% efficiency.

2.3 Operational Modes Optimization Methods

The core DES routine has now been optimized. Each DES mode spends between 75% and 98% of its cycles in the core algorithm, depending on which mode and whether DES or 3DES is used. There are no other major subroutines which require focused attention.

In implementing the ECB and CBC modes for DES and 3DES, calling routines are developed which use the core DES algorithm to process each 64 bit block and provide any necessary feedback between the blocks. In the case of the Electronic Code Book mode, there is no feedback between individual 64-bit blocks. The ECB mode routine is basically an outer loop which calls the core DES algorithm to process each block. In the case of triple-DES ECB, the core DES algorithm is called 3 times for each block, once to encrypt using key #1, again to decrypt using key #2 and finally to encrypt using key #3.

As was studied in the previous section, the most highly optimized core DES algorithm processes three 64-bit blocks in parallel. In the case of DES and triple-DES in the ECB mode, the ECB mode routines simply process blocks 1–3 in parallel and then, for the next iteration process blocks 4–6 in parallel, and continue to process 192 bits (3 64-bit blocks) at a time. In the cases of CBC mode, however, the necessary feedback between each 64-bit block precludes parallel processing of blocks 1–3 of a single channel in parallel.

As is evident from Figure 2(B), the input to each 64-bit encryption block is dependent not only on the input data stream, but upon the output of the previous 64-bit encryption block. This necessarily precludes the ability to process three blocks at the same time. Thus, for CBC mode, a single channel implementation may only use the core DES algorithm which processes a single 64-bit block at a time. It is possible, however, to implement CBC such that three independent channels are processed in parallel. When this is done, blocks #1 of each independent channel may be processed in parallel, using the 3-channel DES core algorithm. Next, blocks #2 of each independent channel may be processed in parallel, using the output of the first block of each independent channel.

Though the need to process three independent channels in parallel in order to achieve maximum performance in the CBC mode may seem like a constraint, it is easily worked around. Firstly, many systems which will be implemented on the C6000 will be multi-channel systems anyway. Secondly, even if only one channel is desired, a single input stream may be broken into three independent channels by using a Time Division Multiplexed (TDM) strategy. A single stream is “de-serialized” into three independent streams by placing blocks #1, #4, #7, etc. into the first independent stream; placing blocks #2, #5, #8,etc. into the second independent stream; and placing blocks #3, #6, #9, etc. into the third independent stream. These three independent streams may be encrypted, sent, and decrypted, then re-serialized into the desired single-channel output stream on the other side. This method does not disrupt security, in fact, an attacker must now break three times as many keys from 1/3 the data per channel which heightens security. The only drawback is the need for the generation and communication of three times as many keys. This is a minor concern considering the boost in efficiency which this method provides.

Output Feedback and Cipher Feedback modes were not investigated in this study. Because they have a feedback structure similar to CBC mode, it is expected that the results of their implementations would be similar to that of CBC mode.

2.4 Memory Structure Optimization Methods

To this point, development of the DES code has been accomplished using an infinite memory model. In this model, there is an infinite amount of on chip memory, and there are no access penalties to be concerned with such as bank hits or cache misses. Bank hits occur when the same bank of memory is accessed twice in the same cycle. Cache misses occur in a cache-based system when the desired memory location is not represented in the cache layer.

The C620x devices have an interleaved memory structure. Memory is separated into four separate banks. Each bank may be accessed only once per cycle. These banks are interleaved such that words #1, #5, #9, etc. are in bank one; words #2, #6, #10, etc. are in bank two; words #3, #7, #11, etc. are in bank three; and words #4, #8, #12,etc. are in bank four. This interleaved structure facilitates vector operations. For instance, in a dot product, if the two input vectors are aligned on different bank boundaries, they will never have a bank hit, because the first elements of each vector will be on different banks, as will the second element of each vector, etc. There is no method for effectively utilizing this interleaved structure, however, when memory is accessed at random locations, as is the case with look-up tables.

The DES core algorithm utilizes a look-up table in order to perform S-box substitutions and P-box permutations. Though the index of these look-ups is deterministically dependent upon both the input block and the encryption key, it cannot be predicted without knowing these variables in advance. Thus, when two of these look-ups are performed, it is not possible to take advantage of the interleaved memory structure to ensure that there is not a bank hit.

This is why the memory structure of many C620x devices provides a second means of reducing bank hits. In addition to being partitioned into four interleaved banks, the internal memory of these devices is separated into two blocks (each of which is interleaved into banks). Any location within block #1 may be accessed in the same cycle as any location within block #2. The bank hits caused by the random access of the S- and P-box look-up table is avoided by making two copies of this table; the first copy is placed in memory bank #1, and the second is placed in memory bank #2. If this table must be accessed twice in the same cycle, the first access is made from the first table and the second access is made from the second table.

The C6211 does not have an interleaved memory structure. Instead, the internal memory of the C6211 is organized into a two-level cache. The first level of the cache is separated into program and data caches, L1P and L1D, respectively. Both L1P and L1D are 4 Kbytes in size. The second cache level, L2, may contain both program and data. The L2 is 64 Kbytes wide, though it is configurable by fourths as either internal SRAM or L2 cache. This internal memory may either be 64 Kbytes cache, 48 Kbytes cache and 16 Kbytes SRAM, 32 Kbytes of cache and 32 Kbytes of SRAM, 16 Kbytes of cache and 48 Kbytes of SRAM, or 64 Kbytes of SRAM. In this study, the entire library of DES functions is about 20 Kbytes in size, so the L2 was configured as 64 Kbytes SRAM (0 Kbytes Cache.)

In order to better understand the cache optimizations, it is necessary to first examine the structure of the subroutines. Each DES mode has a Mode subroutine, a Block subroutine, and a Core subroutine. There are two core subroutines; one processes three channels in parallel, and the other processes a single channel. The core routine performs sixteen iterations of the DES inner loop. These Core routines are called by four Block subroutines; the four Block routines perform DES or triple-DES for either one or three channels. The block routines perform the initial and final permutations and call the DES core routine either once or three times depending on whether DES or triple-DES is used. There are six Mode routines; the Mode routines perform ECB, CBC or 3 channel CBC modes, each for either DES or triple-DES. The mode routines perform the outer loop which call the Block routines to encrypt each 64-bit block and then perform the necessary feedback (if any) between blocks.

Caching relies upon the fact that in many algorithms, the same instructions and data are used over and over again within a single loop structure. The cache of the C6211 also relies upon spatial proximity of program instructions and data values. Spatial proximity is particularly important for program memory because the L1P cache uses the least significant 12 bits of the address in order to hash from the L2 to the L1P, and this cache is one-way. Thus, contiguous memory locations will not thrash. However, memory locations whose addresses are separated by more than 4 Kbytes will thrash if their memory values MOD 4096 are the same. Spatial proximity is also important because when either data or program cache misses occur, the cache not only fetches the missed word, but pre-fetches other contiguous memory locations. More detail is available on the two level cache from reference [9], the TMS320C6000 Peripherals Reference Guide.

In order to maximize the efficiency of the two level cache of the C6211, it is important that the Block subroutines are placed contiguously in memory with the Core subroutines which they call. This ensures that the block subroutines do not thrash with the Core subroutines. This was accomplished by defining sections in the linker command file (named “block1” and “block2”) and placing each Block and Core routine triplet (ordered Block, Core, Block) into each memory block. This ensures that the proper Block and Core routines are contiguous in memory. Caching efficiency is highly dependent upon the situation, but in general this method improved the efficiency of the C6211 cache from approximately 66% efficiency to 90% efficiency and above.

3 Results

The following cycle counts were measured for the DES core algorithm and the 3 channel DES core algorithm. Also, the one- and three-channel block algorithms for DES and triple-DES were measured. All measurements done on the C6211 DSK used Statistics Objects from DSP/BIOS. Cycle counts are averaged from no fewer than 300 measurements. All measurements done on the C6201 EVM used probe points from Code Composer Studio. The cycle counts are presented below. Also included are the memory requirements of each algorithm. The S- and P-Box lookup table is required by both core algorithms but only needs be present once if both algorithms are used. Also, if the dual block optimization is used on the C6201, a copy of this table is made and put into the second block of memory, so it must be accounted for twice.

3.1 TMS320C6211 DSP Starter Kit

Table 3. Memory Required for DES Core Algorithms

DES Algorithm	Cycle Count, C6211	Memory Required
DES core algorithm	304	416 Bytes
DES core algorithm, 3 channel	522	2080 Bytes
DES block algorithm	380	480 Bytes
DES block algorithm, 3 channel	806	1760 Bytes
Triple-DES block algorithm	1056	576 Bytes
Triple-DES block algorithm, 3 channel	1713	1984 Bytes
S- and P-Box Lookup Table	n/a	2048 Bytes

The following cycle counts were measured for DES and triple-DES encryption operating in the CBC and ECB modes. Also, data rates were measured for CBC mode with three independent channels in parallel. All results are averaged from no fewer than 300 measurements. The cycle counts are presented below.

Table 4. DES Mode Cycle Counts, Measured on TMS320C6211

DES Mode	Key Scheduler	Encrypt	Decrypt
Triple-DES, CBC	1448	133,611	129,680
Triple-DES, CBC 3ch	1444	209,544	220,499
Triple-DES, ECB	1445	68,884	68,770
DES, CBC	1425	52,540	51,576
DES, CBC 3ch	1440	96,068	105,836
DES, ECB	1447	31,644	31,722

Cycle counts for Table 4 are measured for the encryption and decryption of 1024 Bytes of data for each mode. (Three channel CBC encrypts 1024 bytes per channel.) Data was measured on the C6211 DSK using Statistics Objects from DSP/BIOS. Results are reported after having subtracted the 86 cycle calculated overhead of STS_set and STS_delta. Cycle counts are averaged from no fewer than 300 iterations.

Table 5. DES Mode Data Rates, Calculated for TMS320C6211

DES Mode	Memory Required	Encrypt	Decrypt
Triple-DES, CBC	992 bytes (4032 bytes)	9.2 Mbps	9.4 Mbps
Triple-DES, CBC 3ch	2560 bytes (8672 bytes)	17.6 Mbps	16.7 Mbps
Triple-DES, ECB	1024 bytes (7136 bytes)	17.8 Mbps	17.8 Mbps
DES, CBC	640 bytes (3584 bytes)	23.4 Mbps	23.84 Mbps
DES, CBC 3ch	1408 bytes (7296 bytes)	38.4 Mbps	34.8 Mbps
DES, ECB	640 bytes (6528 bytes)	38.8 Mbps	38.7 Mbps

Table 5 shows the equivalent data rates for each mode on the C6211 DSK (150 MHz) measured in millions of bits per second. Data rates are calculated directly from cycle counts of Table 4 using Data Rate = 150 MHz/(# cycles/8192 bits). Memory required is given is for the mode algorithm only; the number in parenthesis includes needed block and core algorithms. If multiple modes are used, the S- and P-Box look-up table, block algorithms and core algorithms need only have one copy present, significantly reducing the memory requirement.

3.2 TMS320C6201 Evaluation Module

Table 6. DES Mode Cycle Counts, Measured onTMS320C6201

DES Mode	Key Scheduler	Encrypt	Decrypt
Triple-DES, CBC	760	136,666	136,762
Triple-DES, CBC 3ch	760	210,369	211,822
Triple-DES, ECB	760	73,589	71,268
DES, CBC	760	52,687	53,466
DES, CBC 3ch	760	93,828	100,301
DES, ECB	760	31,266	30,958

Cycle counts for Table 6 are measured for the encryption and decryption of 1024 Bytes of data for each mode. (Three channel CBC encrypts 1024 bytes per channel.) Data was measured on the C6201 Multi-Channel Evaluation Module (McEVM) using Profile Points from Code Composer Studio. Cycle counts are averaged from no fewer than 300 iterations.

Table 7. DES Mode Data Rates, Calculated for TMS320C6201

DES Mode	Memory Required	Encrypt	Decrypt
Triple-DES, CBC	992 bytes (4032 bytes)	12.0 Mbps	12.0 Mbps
Triple-DES, CBC 3ch	2560 bytes (8672 bytes)	23.4 Mbps	23.2 Mbps
Triple-DES, ECB	1024 bytes (7136 bytes)	22.3 Mbps	23.0 Mbps
DES, CBC	640 bytes (3584 bytes)	31.1 Mbps	30.6 Mbps
DES, CBC 3ch	1408 bytes (7296 bytes)	52.4 Mbps	49.0 Mbps
DES, ECB	640 bytes (6528 bytes)	52.4 Mbps	52.9 Mbps

Table 7 shows the equivalent data rates for each DES mode on the C6201 McEVM (200 MHz) measured in millions of bits per second. Data rates are calculated directly from cycle counts of Table 6 using Data Rate = 200 MHz/(# cycles/8192 bits). Data rates for other C620x processors should scale by MHz. Memory required is given is for the mode algorithm only; the number in parenthesis includes needed block and core algorithms. If multiple modes are used, the S- and P-Box look-up table, block algorithms and core algorithms need only have one copy present, significantly reducing the memory requirement.

As is evident from the above charts, cycle counts for each mode are very similar between the C6201 and the C6211 devices. The key scheduler does have very different cycle counts on the two platforms. The key scheduler is run only one time in a row each time that it is run. Thus, the key scheduler does not run very efficiently on the cache-based architecture of the C6211. On the C6201, however, no cache penalty is paid. This should not be a factor in the choice of platform, however, as the key scheduler is a relatively small contributor to overall processor loading.

3.3 Comparison to Infinite Memory Model

The following chart gives a comparison of the cycle counts on the C6211 and C6201 versus the C62xx fast simulator, little endian. This simulator does not take into account cycle delays due to bank hits or cache misses. Thus it is a useful metric against which to measure the efficiency of the C6201 and C6211 memory structures.

Table 8. Memory Efficiency for the TMS320C6201 and TMS320C6211

DES Mode	C62xx Fast Simulator Little Endian Cycle Count	C6201 Percent Efficiency	C6211 Percent Efficiency
Triple-DES, CBC ENCRYPT	123,971	91%	93%
Triple-DES, CBC DECRYPT	124,422	91%	96%
Triple-DES, CBC 3ch ENCRYPT	195,013	93%	93%
Triple-DES, CBC 3ch DECRYPT	195,103	92%	89%
Triple-DES, ECB ENCRYPT	64,502	88%	94%
Triple-DES, ECB DECRYPT	62,180	87%	91%
DES, CBC ENCRYPT	48,529	92%	92%
DES, CBC DECRYPT	49,342	92%	96%
DES, CBC 3ch ENCRYPT	83,485	89%	87%
DES, CBC 3ch DECRYPT	90,788	90%	86%
DES, ECB ENCRYPT	27,625	88%	87%
DES, ECB DECRYPT	27,664	89%	87%

4 Conclusion

Using only C code (no assembly code) and the feedback from the optimizing C compiler, DES is implemented at data rates as high as 52.4 Mbits per second for DES and 22.3 Mbits per second for triple-DES on the C6201 McEVM (200 MHz). Using the C6211 DSK (150 MHz), data rates were measured as high as 38.8 Mbits per second for DES and 17.8 Mbits per second for triple-DES.

The performance of the C coded core algorithm was shown to be 76% of the theoretical maximum performance. It is likely that hand coded cannot achieve this theoretical maximum performance and that the C coded version of the core algorithm is greater than 76% efficient when compared to hand coded assembly.

The efficiency of the dual block, 4 way interleaved bank memory of the C6201b was shown to be between 87–93% efficient on the Multi-channel Evaluation Module compared to the infinite memory model. The two level cache of the C6211 was shown to be between 86–96% efficient on the DSP Starter Kit compared to the infinite memory model.

Overall the C6000 platform boasts fantastic performance for DES using C code. Because most encryption algorithms are in the public domain and available as C code for free with no licensing restrictions, C programming is a quick and inexpensive way to add encryption functionality to a design. Because this is a software approach, this methodology is not only easily extensible to quickly building an entire library of encryption standards, but provides the flexibility to evolve designs from DES to any algorithm which may replace it as the mainstream encryption algorithm in the future.

To begin development of your own system on the TMS320C6000 platform, contact your local Texas Instruments sales representative or one of the Product Information Centers listed on the last page of this document. The Texas Instruments website, listed on the same page, is another resource to find out more information about the C6000 DSP Starter Kit, Evaluation Modules, Code Composer Studio, and the variety of other tools available to assist in product development.

5 References

Refer to the following documents to learn more about DES and the TMS320C6000.

1. National Bureau of Standards, NBS FIPS PUB 81, “Guidelines for Implementing and Using the NBS Data Encryption Standard,” U.S. Department of Commerce
2. ANSI X3.92, “American National Standard for Data Encryption Algorithm (DEA),” American National Standards Institute, 1981.
3. ANSI3.106, “American National Standard for Information Systems—Data Encryption Algorithm—Modes of Operation,” American National Standards Institute, 1983.
4. “Applied Cryptography, Second Edition,” Bruce Schneider, John Wiley & Sons, Inc., New York, NY, 1996.
5. “Chip, algorithm bridge security gap,” Walter Boyles, Electronic Engineering Times, p.94, November 8, 1999.
6. Eric Young, Cryptsoft, eay@cryptsoft.com
7. *TMS320C6000 Compiler Optimization Tutorial*, Literature Number SPRH046, Texas Instruments.
8. *TMS320C6000 Optimizing C Compiler User’s Guide*, Literature Number SPRU187E, Texas Instruments.
9. *TMS320C6000 Peripherals Reference Guide*, Literature Number SPRU190C, Texas Instruments, Tarrant Dallas Printing, Dallas, Texas, 1999
10. *TMS320C62x/C67x CPU and Instruction Set Reference Guide*, Literature Number SPRU189C, Texas Instruments, Tarrant Dallas Printing, Dallas, Texas, 1998
11. *Code Composer Studio User’s Guide*, Literature Number SPRU328A, Texas Instruments, Tarrant Dallas Printing, Dallas Texas, 1999
12. *TMS320C6000 DSP/BIOS User’s Guide*, Literature Number SPRU303, Texas Instruments, Tarrant Dallas Printing, Inc., Dallas, Texas, 1999