Embedded SDK
(Software Development Kit)

Triple Data Encryption Standard (3DES) Library

SDK119/D
Rev. 2, 07/16/2002
## Contents

### About This Document
- Audience ................................................................. ix
- Organization .............................................................. ix
- Suggested Reading ....................................................... ix
- Conventions .............................................................. x
- Definitions, Acronyms, and Abbreviations ......................... x
- References ................................................................. xi

### Chapter 1
**Introduction**
- 1.1 Quick Start .......................................................... 1-1
- 1.2 Background and Overview of 3DES ............................... 1-1
- 1.3 Features and Performance ........................................... 1-2

### Chapter 2
**Directory Structure**
- 2.1 Required Core Directories ......................................... 2-1
- 2.2 Optional (Domain-Specific) Directories ......................... 2-2

### Chapter 3
**3DES Library Interfaces**
- 3.1 3DES Services ....................................................... 3-1
- 3.2 Interface ............................................................. 3-1
- 3.3 Specifications ....................................................... 3-4
  - 3.3.1 triDesCreate ..................................................... 3-5
  - 3.3.2 triDesInit ........................................................ 3-9
  - 3.3.3 triDesEncrypt ..................................................... 3-11
  - 3.3.4 triDesDecrypt ..................................................... 3-14
  - 3.3.5 triDesControl ..................................................... 3-17
  - 3.3.6 triDesDestroy ..................................................... 3-20

### Chapter 4
**Building the 3DES Library**
- 4.1 Building the 3DES Library ......................................... 4-1
  - 4.1.1 Dependency Build ............................................... 4-1
  - 4.1.2 Direct Build ..................................................... 4-2

---

**FOR MORE INFORMATION ON THIS PRODUCT,**
**Go to:** [www.freescale.com](http://www.freescale.com)
Chapter 5
Linking Applications with the 3DES Library

5.1  3DES Library ................................................................. 5-1
5.1.1  Library Sections ......................................................... 5-1

Chapter 6
3DES Applications

6.1  Test and Demo Applications ........................................... 6-1

Chapter 7
License

7.1  Limited Use License Agreement ...................................... 7-1
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>triDesCreate Arguments</td>
<td>3-5</td>
</tr>
<tr>
<td>3-2</td>
<td>triDesInit Arguments</td>
<td>3-9</td>
</tr>
<tr>
<td>3-3</td>
<td>triDesEncrypt Arguments</td>
<td>3-11</td>
</tr>
<tr>
<td>3-4</td>
<td>triDesDecrypt Arguments</td>
<td>3-14</td>
</tr>
<tr>
<td>3-5</td>
<td>triDesControl Arguments</td>
<td>3-17</td>
</tr>
<tr>
<td>3-6</td>
<td>triDesDestroy Arguments</td>
<td>3-20</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Triple-DES: (a) Encryption and (b) Decryption</td>
<td>1-2</td>
</tr>
<tr>
<td>2-1</td>
<td>Core Directories</td>
<td>2-1</td>
</tr>
<tr>
<td>2-2</td>
<td>security Directory</td>
<td>2-2</td>
</tr>
<tr>
<td>2-3</td>
<td>3des Directory Structure</td>
<td>2-2</td>
</tr>
<tr>
<td>2-4</td>
<td>3des_demo Application</td>
<td>2-3</td>
</tr>
<tr>
<td>4-1</td>
<td>Dependency Build for 3DES Library</td>
<td>4-1</td>
</tr>
<tr>
<td>4-2</td>
<td>3des.mcp Project</td>
<td>4-2</td>
</tr>
<tr>
<td>4-3</td>
<td>Execute Make</td>
<td>4-2</td>
</tr>
</tbody>
</table>
List of Examples

<table>
<thead>
<tr>
<th>Code Example</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>C Header File 3des.h.</td>
<td>3-1</td>
</tr>
<tr>
<td>3-2</td>
<td>mem Library</td>
<td>3-5</td>
</tr>
<tr>
<td>3-3</td>
<td>Use of triDesCreate Interface</td>
<td>3-7</td>
</tr>
<tr>
<td>3-4</td>
<td>Sample Callback Procedure</td>
<td>3-10</td>
</tr>
<tr>
<td>3-5</td>
<td>Use of triDesEncrypt Interface</td>
<td>3-12</td>
</tr>
<tr>
<td>3-6</td>
<td>Use of triDesEncrypt / triDesDecrypt Interface</td>
<td>3-15</td>
</tr>
<tr>
<td>3-7</td>
<td>Use of triDesControl Interface</td>
<td>3-17</td>
</tr>
<tr>
<td>3-8</td>
<td>Use of triDesDestroy Interface</td>
<td>3-20</td>
</tr>
<tr>
<td>5-1</td>
<td>linker.cmd File</td>
<td>5-1</td>
</tr>
</tbody>
</table>
About This Document

This manual describes the Triple Data Encryption Standard, (3DES), algorithm for use with Motorola’s Embedded Software Development Kit, (SDK).

Audience

This document targets software developers implementing encryption/decryption functions within software applications.

Organization

This manual is arranged in the following sections:

- **Chapter 1, Introduction**—provides a brief overview of this document
- **Chapter 2, Directory Structure**—provides a description of the required core directories
- **Chapter 3, 3DES Library Interfaces**—describes all of the 3DES Library functions
- **Chapter 4, Building the 3DES Library**—tells how to execute the system library project build
- **Chapter 5, Linking Applications with the 3DES Library**—describes the organization of the 3DES Library
- **Chapter 6, 3DES Applications**—describes the use of 3DES library through test/demo applications
- **Chapter 7, License**—provides the license required to use this product

Suggested Reading

We recommend that you have a copy of the following references:

- *DSP56800 Family Manual*, DSP56800FM/AD
- *DSP56824 User’s Manual*, DSP56824UM/AD
Conventions

This document uses the following notational conventions:

<table>
<thead>
<tr>
<th>Typeface, Symbol or Term</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier Monospaced Type</td>
<td>Commands, command parameters, code examples, expressions, datatypes, and directives</td>
<td>&quot;...*Foundational include files...&quot; &quot;...a data structure of type vad_tConfigure...&quot;</td>
</tr>
<tr>
<td>Italic</td>
<td>Calls, functions, statements, procedures, routines, arguments, file names and applications</td>
<td>&quot;...the pConfig argument...&quot; &quot;...defined in the C header file, aec.h...&quot; &quot;...makes a call to the Callback procedure...&quot;</td>
</tr>
<tr>
<td>Bold</td>
<td>Reference sources, paths, emphasis</td>
<td>&quot;...refer to the Targeting DSP56824 Platform manual....&quot; &quot;...see: C:\Program Files\Motorola\Embedded SDK\help\tutorials&quot;</td>
</tr>
<tr>
<td>Bold/Italic</td>
<td>Directory name, project name</td>
<td>&quot;...and contains these core directories: applications contains applications software....&quot; &quot;...CodeWarrior project, 3des.mcp, is.....&quot;</td>
</tr>
<tr>
<td>Blue Text</td>
<td>Linkable on-line</td>
<td>&quot;...refer to Chapter 7, License...&quot;</td>
</tr>
<tr>
<td>Number</td>
<td>Any number is considered a positive value, unless preceded by a minus symbol to signify a negative value</td>
<td>3V -10 DES^-1</td>
</tr>
<tr>
<td>ALL CAPITAL LETTERS</td>
<td>Variables, directives, defined constants, files libraries</td>
<td>INCLUDE_DSPFUNC #define INCLUDE_STACK_CHECK</td>
</tr>
<tr>
<td>Brackets [...]</td>
<td>Function keys</td>
<td>&quot;...by pressing function key [F7]...&quot;</td>
</tr>
<tr>
<td>Quotation marks &quot;... &quot;</td>
<td>Returned messages</td>
<td>&quot;...the message, &quot;Test Passed&quot; is displayed....&quot; &quot;...if unsuccessful for any reason, it will return &quot;NULL&quot;....&quot;</td>
</tr>
</tbody>
</table>

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document. As this template develops, this list will be generated from the document. As we develop more group resources, these acronyms will be easily defined from a common acronym dictionary. Please note that while the acronyms are in solid caps, terms in the definition should be initial capped ONLY IF they are trademarked names or proper nouns.

- DES: Data Encryption Standard
- 3DES: Triple Data Encryption Standard
- DSP: Digital Signal Processor or Digital Signal Processing
- FFT: Fast Fourier Transforms
- FIR: Finite Impulse Response
I/O        Input/Output
IDE       Integrated Development Environment
IIR       Infinite Impulse Response
LSB       Least Significant Bit
MAC       Multiply/Accumulate
MIPS      Million Instructions Per Second
MSB       Most Significant Bit
OnCE™     On-Chip Emulation
OMR       Operating Mode Register
PC        Program Counter
SDK       Software Development Kit
SP        Stack Pointer
SPI       Serial Peripheral Interface
SR        Status Register
SRC       Source

References

The following sources were used to produce this book:

1. *DSP56800 Family Manual*, DSP56800FM/AD
2. *DSP56824 User’s Manual*, DSP56824UM/AD
5. Federal Information Processing Standards Publication 81,(FIPS PUB 81), DES Modes of Operation
6. *Embedded SDK Data Encryption Standard (DES) Library*
Chapter 1
Introduction

Welcome to Motorola’s family of Digital Signal Processors, (DSPs). This document describes the 3DES Library, which is a part of Motorola’s comprehensive Software Development Kit, (SDK), for its DSPs. In this document, you will find all the information required to use and maintain the 3DES Library interface and algorithms.

Motorola provides these algorithms to you for use with Motorola DSPs to expedite your application development and reduce the time it takes to bring your own products to market.

Motorola’s 3DES Library is licensed for your use on Motorola processors. Please refer to the standard Software License Agreement in Chapter 7 for license terms and conditions; please consult with your Motorola representative for premium product licensing.

1.1 Quick Start

Motorola Embedded SDK is targeted to a large variety of hardware platforms. To take full advantage of a particular hardware platform, use Quick Start from the Targeting DSP568xx Platform documentation.

For example, the Targeting DSP56824 Platform manual provides more specific information and examples about this hardware architecture. If you are developing an application for the DSP56824EVM board or any other DSP56824 development system, refer to the Targeting DSP56824 Platform manual for Quick Start or other DSP56824-specific information.

1.2 Background and Overview of 3DES

The DES core used in 3DES conforms to the Federal Information Processing Standards Publication 46-2, (FIPS PUB 46-2).

Any secure communication/storage system requires converting data to be transmitted into an unintelligible form called ciphertext, and reconverting the data to its original form, called plaintext. The FIPS PUB 46-2 standard describing DES is the most widely-used encryption standard for these applications. A variant of this algorithm is 3DES, where DES is used three times, with different keys each time. Motorola has internally developed a single library module containing the 3DES.

Figure 1-1 shows the structure of 3DES using DES. The figure itself describes the encryption and decryption processes.
1.3 Features and Performance

The Triple Data Encryption Standard library is multichannel and re-entrant. For details on Memory and MIPS for a particular DSP, refer to the Libraries chapter of the appropriate Targeting manual.
Chapter 2
Directory Structure

2.1 Required Core Directories

Figure 2-1 details required platform directories:

As shown in Figure 2-1, DSP56824EVM has no operating system (nos) support, and contains these core directories:

- **applications** contains applications software that can be exercised on this platform
- **bsp** contains the board support package specific for this platform
- **config** contains default HW/SW configurations for this platform
- **include** contains SDK header files which define the Application Programming Interface
- **sys** contains required system components
- **tools** contains utilities used by system components

There are also optional directories that include domain-specific libraries.
2.2 Optional (Domain-Specific) Directories

As shown in Figure 2-2, the security directory contains algorithms specific to 3DES.

![security Directory Diagram](image1)

Figure 2-2. security Directory

Figure 2-3 shows details of the 3des directory structure.

![3des Directory Structure Diagram](image2)

Figure 2-3. 3des Directory Structure
The `3des` directory includes the following sub-directories:

- **asm_source** - contains all asm sources required for 3DES
- **c_sources** - includes APIs for 3DES
- **test_3des** - includes C source files and configuration necessary for testing 3DES library modules
  - **c_sources** - contains an example test code for 3DES
  - **Config** - contains the configuration files `appconfig.c`, `appconfig.h` and `linker.cmd` specific to 3DES.

The `applications` directory includes high-level software that exercises the 3DES library, including the `3des_demo` application, detailed in **Figure 2-4**.

![Directory Structure](image)
Chapter 3
3DES Library Interfaces

3.1 3DES Services

The 3DES library encrypts/decrypts user-supplied data. The data supplied must be in bytes, in the following format, assuming the word size to be 16 bits:

```
0 0 0 0 0 0 0 i i i i i i i
```

- **MSB**
- **LSB**

- **i** = information bit
- **0** = Most significant 8 bits; must be 0

3.2 Interface

The C interface for 3DES library services is defined in the C header file 3des.h, shown in Code Example 3-1 as a reference.

**Code Example 3-1. C Header File 3des.h**

```c
/* File: 3des.h */

#ifndef __3DES_H
#define __3DES_H

/* Triple Data Encryption Standard (3DES) interface */

/*-------------------------------
 * Foundational include files
 *-------------------------------*/

#include "port.h"

/*-------------------------------
 * #defines for 3DES configuration flags
 *-------------------------------*/
```
#define TRI_DES_ENCRYPT 0x0001
#define TRI_DES_DECRYPT 0x0000

/*---------------------------------------
* Structures for 3DES configuration
*---------------------------------------*/
typedef struct
{
    char key1[8];  /* 64 bit key1 */
    char key2[8];  /* 64 bit key2 */
    char key3[8];  /* 64 bit key3 */
} tri_des_skey;

typedef struct
{
    void (*pCallback) (void *pCallbackArg, char *pChars,
                        UWord16 NumChars);
    void *pCallbackArg;
} tri_des_sCallback;

typedef struct
{
    UWord16 Flags;  /* Encrypt or Decrypt flag */
    tri_des_skey *pKey;
    tri_des_sCallback Callback;
} tri_des_sConfigure;

typedef struct
{
    UWord16 Flags;  /* Encrypt or Decrypt flag */
    char *pSubkey1;
    char *pSubkey2;
    char *pSubkey3;
    char *pOut_Buf;
    char *pContext_buf;
    UInt16 c_len;  /* length of context stored; 0 to start with */
    tri_des_sCallback *cback;
    Word16 arg1;
    Word16 arg2;
    Word16 arg3;
    Word16 arg4;
    Word16 arg5;
    Word16 arg6;
    Word16 K;  /* Number of feedback bits */
    Word16 M;  /* Number of feedback bytes */
    Word16 MM;
}
Word16 NUM;                 /* Number of Remainder bits after M bytes */
Word16 subkey;              /* Buffer to store 16 subkeys */
Word16 *key_ptr;            /* Address of the encryption key */

Word16 *buffer_A_ptr;
Word16 *buffer_C_ptr;
Word16 temp_buf [8];
Word16 word_pointer;
Word16 bit_pointer;
Word16 *input_buffer_ptr;
Word16 *output_buffer_ptr;

Word16 in_pointer_storage; /* Plain text pointer. */
Word16 key_pointer_storage; /* Key pointer. */
Word16 out_pointer_storage; /* Cipher text pointer. */

Word16 *permuted_text_ptr;
Word16 c_word [8];
Word16 *key_temp_ptr;
Word16 encrypt_lc;
Word16 key_gen_lc;
Word16 pc2_lc;
} tri_des_sHandle;

/*---------------------------------------
* Commands for 3DES Control
*---------------------------------------*/
#define TRI_DES_DEACTIVATE 2

/*---------------------------------------
* Function Prototypes
*---------------------------------------*/
EXPORT tri_des_sHandle *triDesCreate (tri_des_sConfigure *pConfig);
EXPORT Result triDesInit (tri_des_sHandle *p3Des, tri_des_sConfigure *pConfig);
EXPORT Result triDesEncrypt (tri_des_sHandle *p3Des, char *pChars, UWord16 NumChars);
EXPORT Result triDesDecrypt (tri_des_sHandle *p3Des, char *pChars, UWord16 NumChars);
EXPORT Result triDesControl (tri_des_sHandle *p3Des, UWord16 Command);
EXPORT Result triDesDestroy (tri_des_sHandle *p3Des);

/*-----------------------------------------
* Some #defines used in the implementation
*-----------------------------------------*/
#define TRI_DES_SUBKEY_SIZE 128 /* 16 8-bit subkeys */
#define TRI_DES_BLOCK_SIZE  128 /* Number of bytes per block */

/* Used to compute remainder bytes, when divided by 128, in a
 * given chunk of data block */
#define TRI_DES_REM_MASK  0x7f
#endif
3.3 Specifications

The following pages describe the 3DES library functions.

Function arguments for each routine are described as in, out, or inout. An in argument means that the parameter value is an input only to the function. An out argument means that the parameter value is an output only from the function. An inout argument means that a parameter value is an input to the function, but the same parameter is also an output from the function.

Typically, the inout parameters are input pointer variables in which the caller passes the address of a preallocated data structure to a function. The function stores its results within that data structure. The actual value of the inout pointer parameter is not changed.
3.3.1 triDesCreate

Call(s):

```c
tri_des_sHandle *triDesCreate (tri_des_sConfigure *pConfig);
```

Required Header: “3des.h”

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConfig</td>
<td>Points to the configuration data for 3DES</td>
</tr>
</tbody>
</table>

**Description:** The `triDesCreate` function creates an instance of 3DES. The `pConfig` argument points to the `tri_des_sConfigure` structure used to configure 3DES operation. For initialization of the `tri_des_sConfigure` structure, refer to `triDesInit`, Section 3.3.2. During the `triDesCreate` call, any dynamic resources required by the 3DES algorithm are allocated. The memory allocated is 1003 external data memory words per instance. The library allocates memory dynamically using the `mem` library shown in **Code Example 3-2**. The library is multichannel and re-entrant.

**Code Example 3-2.  mem Library**

```c
#include "3des.h"
#include "mem.h"

tri_des_sHandle *triDesCreate (tri_des_sConfigure *pConfig)
{
    tri_des_sHandle *p3Des;
    Int16 i, j;
    bool flag = true;
    Result result;

    /* Allocate memory for p3Des */
    p3Des = (tri_des_sHandle *) memMallocEM (sizeof (tri_des_sHandle));
    if (p3Des == NULL) return (NULL);

    p3Des->pSubkey1 = (char *) memMallocEM (TRI_DES_SUBKEY_SIZE * sizeof (char));
    if (p3Des->pSubkey1 == NULL) flag = false;

    p3Des->pSubkey2 = (char *) memMallocEM (TRI_DES_SUBKEY_SIZE * sizeof (char));
    if (p3Des->pSubkey2 == NULL) flag = false;

    p3Des->pSubkey3 = (char *) memMallocEM (TRI_DES_SUBKEY_SIZE * sizeof (char));
    if (p3Des->pSubkey3 == NULL) flag = false;

    p3Des->cback = (tri_des_sCallback *) memMallocEM (sizeof (tri_des_sCallback));
    if (p3Des->cback == NULL) flag = false;

    p3Des->pOut_Buf = (char *) memMallocEM (TRI_DES_BLOCK_SIZE * sizeof (char));
    p3Des->pContext_buf = (char *) memMallocEM (TRI_DES_BLOCK_SIZE * sizeof (char));
    if ((p3Des->pOut_Buf == NULL) || (p3Des->pContext_buf == NULL))
        flag = false;

    p3Des->input_buffer_ptr = (Word16 *)memMallocEM (136 * sizeof (Word16));
    p3Des->output_buffer_ptr = (Word16 *)memMallocEM (136 * sizeof (Word16));
    if ((p3Des->input_buffer_ptr == NULL) || (p3Des->output_buffer_ptr == NULL))
```
flag = false;

p3Des->buffer_A_ptr = (Word16 *)memMallocAlignedEM (8 * sizeof(Word16));
if (p3Des->buffer_A_ptr == NULL)
    flag = false;

flag &= memIsAligned (p3Des->buffer_A_ptr, 8);

p3Des->buffer_C_ptr = (Word16 *)memMallocAlignedEM(8 * sizeof(Word16));
if (p3Des->buffer_C_ptr == NULL)
    flag = false;

flag &= memIsAligned (p3Des->buffer_C_ptr, 8);

p3Des->permuted_text_ptr = (Word16 *)memMallocAlignedEM (8 * sizeof(Word16));
if (p3Des->permuted_text_ptr == NULL)
    flag = false;

flag &= memIsAligned (p3Des->permuted_text_ptr, 8);

p3Des->key_temp_ptr = (Word16 *)memMallocAlignedEM (7 * sizeof(Word16));
if (p3Des->key_temp_ptr == NULL)
    flag = false;

flag &= memIsAligned (p3Des->key_temp_ptr, 8);

p3Des->key_ptr = (Word16 *)memMallocAlignedEM(8 * sizeof(Word16));
if (p3Des->key_ptr == NULL)
    flag = false;

flag &= memIsAligned (p3Des->key_ptr, 8);

if (flag == false)
{
    triDesDestroy (p3Des);
    return (NULL);
}
else
{
    triDesInit (p3Des,pConfig);
    return (p3Des);
}

For details on the tri_des_sHandle structure, please refer to Code Example 3-1.

If a triDesCreate function is called to create an instance, then triDesDestroy Section 3.3.6 should be used to destroy the instance.

Alternatively, the user can allocate memory statically which requires duplicating all statements in the triDesCreate function. In this case, the user can call the triDesInit function directly, bypassing the triDesCreate function. If the user dynamically allocates memory without calling triDesCreate, then the user himself must destroy the memory allocated.

Returns: Upon successful completion, the triDesCreate function will return a pointer to the specific instance of 3DES created. If triDesCreate is unsuccessful for any reason, it will return NULL.
Special Considerations:

- 3DES is multichannel and re-entrant.
- If `triDesCreate` is called, then the user need not call `triDesInit` function as it is called internally in the `triDesCreate` function.

In Code Example 3-3, the application creates an instance of 3DES.

Code Example 3-3. Use of `triDesCreate` Interface

```c
#include "3des.h"
#include "mem.h"

define PT_LEN 441 /* Length of plaintext */

typedef struct /* example structure that can be used in the callback function */
{
    char buffer[512];
    UWord16 offset;
} WriteOutput;

/* Callback function prototype */
void Callback (void *pCallbackArg, char *pChars, UWord16 NumChars);

void test3Des (void)
{
    char plaintext[PT_LEN] = {0x0021, 0x007A, 0x00D5, .....};
    tri_des_sConfigure *pConfig;
    tri_des_sHandle *p3Des;
    UWord16 cntrl;
    tri_des_skey *pkey;
    UWord16 Command = TRI_DES_DEACTIVATE;
    WriteOutput P_Text;
    WriteOutput C_Text;
    Int16 res;

    P_Text.offset = 0;
    C_Text.offset = 0;

    /* Memory allocation */
pConfig = (tri_des_sConfigure *) memMallocEM (sizeof (tri_des_sConfigure));
pkey = (tri_des_skey *) memMallocEM (sizeof (tri_des_skey));

    /* Initialize the keys */
pkey->key1[0] = 0x006C;
pkey->key1[1] = 0x0060;
pkey->key1[2] = 0x00F2;
pkey->key1[3] = 0x0060;
pkey->key1[4] = 0x003E;
pkey->key1[5] = 0x0016;
pkey->key1[6] = 0x00CC;
pkey->key1[7] = 0x00A0;
```

For More Information On This Product, Go to: www.freescale.com
pkey->key2[0] = 0x0061;
pkey->key2[1] = 0x0062;
pkey->key2[2] = 0x00f3;
pkey->key2[3] = 0x0064;
pkey->key2[4] = 0x0035;
pkey->key2[5] = 0x0016;
pkey->key2[6] = 0x00C7;
pkey->key2[7] = 0x00A8;

pkey->key3[0] = 0x002C;
pkey->key3[1] = 0x0030;
pkey->key3[2] = 0x0042;
pkey->key3[3] = 0x0050;
pkey->key3[4] = 0x006E;
pkey->key3[5] = 0x0076;
pkey->key3[6] = 0x008C;
pkey->key3[7] = 0x0090;

/* Initialize pConfig structure for ENCRYPTION */
pConfig->Flags = TRI DES_ENCRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&C_Text);

/* Create and initialize instance */
p3Des = triDesCreate (pConfig); /* triDesInit function is called internally */
....
}
3.3.2 *triDesInit*

**Call(s):**

```c
Result triDesInit (tri_des_sHandle *p3Des, tri_des_sConfigure *pConfig);
```

**Required Header:** “3des.h”

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p3Des</td>
<td>Handle to an instance of 3DES</td>
</tr>
<tr>
<td>pConfig</td>
<td>A pointer to a data structure containing data for initializing the 3DES algorithm</td>
</tr>
</tbody>
</table>

**Description:** The *triDesInit* function will initialize the 3DES algorithm. During the initialization, all resources will be set to their original values in preparation for 3DES operation. Before calling *triDesInit* function, a 3DES instance must be created. The 3DES instance, p3Des, can be created by either calling *triDesCreate* function, shown in Section 3.3.1, or by statically allocating memory, which does not require calling the *triDesCreate* function.

The parameter *pConfig* points to a data structure of type *tri_des_sConfigure*; its fields initialize 3DES operation in the following manner:

**Flags** - A set of configuration options for 3DES. Flags options include:

- TRI_DES_ENCRYPT  - Encrypt the data that is being given to *triDesEncrypt*
- TRI_DES_DECRYPT  - Decrypt the data that is being given to *triDesDecrypt*

**pKey** - A pointer to a structure of type *tri_des_skey* (see *3des.h* file for details)

**Callback** - A structure of type *tri_des_sCallback*; it describes the procedure which 3DES will call once the data bytes are encrypted/decrypted by the algorithm. The callback procedure has the following declaration:

```c
define void (*pCallback)(void *pCallbackArg, char *pChars, UWord16 NumChars);
```

The callback procedure parameter, *pCallbackArg*, is supplied by the user in the *tri_des_sCallback* structure; this value is passed back to the user during the call to the Callback procedure. Typically, *pCallbackArg* points to context information used by the callback procedure, which the user must write.

*pChars* is the pointer to the output buffer containing either encrypted or decrypted characters.

*NumChars* is the number of characters pointed to by *pChars*.

An example callback procedure is shown in Code Example 3-4 as a reference and to aid in writing your own procedure. This callback procedure stores the data (characters) in a buffer specified by the user through the *pCallbackArg* pointer.
Code Example 3-4. Sample Callback Procedure

```c
void Callback (void *pCallbackArg, char *pChars, UWord16 NumChars)
{
    Int16 i;
    WriteOutput *temp = (WriteOutput *) pCallbackArg;
    for (i = 0; i < NumChars; i++)
    {
        temp->buffer[i + (temp->offset)] = pChars[i];
    }
    temp->offset += 128;
    return;
}
```

For details on the `WriteOutput` type of structure, refer to Code Example 3-3, which illustrates the operation of the callback function. Since this function must be written by the user, its content depends on how the user wants to process the 3DES output.

**Returns:** Upon successful completion, a value of “TRUE” will be returned. Otherwise, a value of “FALSE” will be returned.

**Special Considerations:**
- If `triDesCreate` is called, then the user need not call `triDesInit` function as it is called internally in the `triDesCreate` function.

**Code Example:** See Code Example 3-3 to learn how to use the `v2Init` function.
### 3.3.3 triDesEncrypt

**Call(s):**

```c
Result triDesEncrypt (tri_des_sHandle *p3Des,
                     char *pChars,
                     UWord16 NumChars);
```

**Required Header:** “3des.h”

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p3Des</code></td>
<td>Handle to an instance of 3DES</td>
</tr>
<tr>
<td><code>pChars</code></td>
<td>Pointer to the user data bytes to be encrypted by the 3DES algorithm</td>
</tr>
<tr>
<td><code>NumChars</code></td>
<td>The number of data bytes to be encrypted</td>
</tr>
</tbody>
</table>

**Description:** The `triDesEncrypt` function will encrypt the data supplied. Once the encryption is complete, the result is returned to the user by calling the `Callback` procedure. The user can call the `triDesEncrypt` function any number of times, as long as the user has data.

**Returns:** Upon successful completion, `triDesEncrypt` will return “PASS”; if any error occurred, `triDesEncrypt` will return “FAIL”.

**Special Considerations:**

- In place computation is allowed; i.e., input and output buffers could be identical
- The `triDesEncrypt` function makes a call to the `Callback` procedure only when 128 bytes of data are encrypted.

**For Encryption:**

Suppose 89 bytes of data are to be encrypted. The length to be passed for encryption would be shown as:

```c
triDesEncrypt (p3Des, pChars, 89); /* for encryption */
```

However, while decrypting, the call to `triDesDecrypt`, Section 3.3.4, must be made with a length of 128 bytes, as shown below:

```c
triDesDecrypt (p3Des, pChars, 128); /* for decryption */
```

To summarize, the total length while decrypting must satisfy this requirement:

\[
\sum_{i=0}^{\infty} l_i = m \times 128
\]

\( l_i \) = the length passed during the \( i \)-th call to `triDesDecrypt` during decryption

\( m = \) any positive integer, so that \( m \times 128 \) is \( \geq \) the total length passed during encryption

**Code Example 3-5**

In `Code Example 3-5`, during encryption, the total length of plaintext is \((13 + 121 + 300) = 434\) bytes; during decryption, the total length of ciphertext is \((13 + 300 + 199) = 512\) bytes. Thus, 512 is both a multiple of 128, and greater than 434, satisfying the constraint.
#include "3des.h"
#include "mem.h"

#define PT_LEN 434 /* Length of plain text */

typedef struct /* example structure that can be used in the callback function */
{
    char buffer[512];
    UWord16 offset;
} WriteOutput;

/* Callback function prototype */
void Callback (void *pCallbackArg, char *pChars, UWord16 NumChars);

void test3Des (void)
{
    char plaintext[PT_LEN] = {0x0021, 0x007A, 0x00D5, .....};
    tri_des_sConfigure *pConfig;
    tri_des_sHandle *p3Des;
    UWord16 cntrl;
    tri_des_skey *pkey;
    UWord16 Command = TRI_DES_DEACTIVATE;
    WriteOutput P_Text;
    WriteOutput C_Text;
    Int16 res;

    P_Text.offset = 0; /* offset is used in callback, see the example on 
callback above */
    C_Text.offset = 0; /* offset is used in callback, see the example on 
callback above */

    /* Memory allocation */
    pConfig = (tri_des_sConfigure *) memMallocEM (sizeof (tri_des_sConfigure));
    pkey = (tri_des_skey *) memMallocEM (sizeof (tri_des_skey));

    /* Initialize the keys */
    pkey->key1[0] = 0x006C;
    pkey->key1[1] = 0x0060;
    pkey->key1[2] = 0x00F2;
    pkey->key1[3] = 0x0060;
    pkey->key1[4] = 0x003E;
    pkey->key1[5] = 0x0016;
    pkey->key1[6] = 0x00CC;
    pkey->key1[7] = 0x00A0;

    pkey->key2[0] = 0x0061;
    pkey->key2[1] = 0x0062;
    pkey->key2[2] = 0x00F3;
    pkey->key2[3] = 0x0064;
    pkey->key2[4] = 0x0035;
    pkey->key2[5] = 0x0016;
    pkey->key2[6] = 0x00C7;
    pkey->key2[7] = 0x00A8;
pkey->key3[0] = 0x002C;
pkey->key3[1] = 0x0030;
pkey->key3[2] = 0x0042;
pkey->key3[3] = 0x0050;
pkey->key3[4] = 0x006E;
pkey->key3[5] = 0x0076;
pkey->key3[6] = 0x008C;
pkey->key3[7] = 0x0090;

/* Initialize pConfig structure for ENCRYPTION */
pConfig->Flags = TRI_DES_ENCRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&C_Text);

p3Des = triDesCreate (pConfig);
...
res = triDesEncrypt (p3Des, plaintext, 13); /* Encrypt 13 bytes of data */
...
res = triDesEncrypt (p3Des, &plaintext[13], 121); /* Encrypt 121 bytes of data */
...
res = triDesEncrypt (p3Des, &plaintext[134], 300); /* Encrypt 300 bytes of data */
...
......
3.3.4 triDesDecrypt

Call(s):

\[
\text{Result } \text{triDesDecrypt} (\text{tri_des_sHandle } *p3Des, \\
\quad \text{char } *pChars, \\
\quad \text{UWord16 NumChars});
\]

Required Header: “3des.h”

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p3Des</td>
<td>Handle to an instance of 3DES</td>
</tr>
<tr>
<td>pChars</td>
<td>Pointer to the user data bytes to be decrypted by the 3DES algorithm</td>
</tr>
<tr>
<td>NumChars</td>
<td>The number of data bytes to be decrypted</td>
</tr>
</tbody>
</table>

Description: The triDesDecrypt function will decrypt the data supplied. Once the decryption is complete, the result is returned to the user by calling the Callback procedure. The user can call the triDesDecrypt function any number of times, as long as the user has data.

Returns: Upon successful completion, triDesDecrypt will return “PASS”; if any error occurred, triDesDecrypt will return “FAIL”.

Special Considerations:

- In place computation is allowed; i.e., input and output buffers could be identical
- The triDesDecrypt function makes a call to the Callback procedure only when 128 bytes of data are decrypted

For Encryption:

Suppose 89 bytes of data are to be encrypted. The length to be passed for encryption would be shown as:

\[
\text{triDesEncrypt} (\text{p3Des, pChars, 89}); \quad /* \text{for encryption} */
\]

However, while decrypting, the call to triDesDecrypt must be made with a length of 128 bytes, as shown here:

\[
\text{triDesDecrypt} (\text{p3Des, pChars, 128}); \quad /* \text{for decryption} */
\]

To summarize, the total length while decrypting must satisfy this requirement:

\[
\sum_{i=1}^{\infty} l_i = m \times 128
\]

\(l_i\) = length passed during the i-th call to triDesDecrypt during decryption
m = any positive integer, so that \(m \times 128\) is \(\geq\) total length passed during encryption
In Code Example 3-6, during encryption, the total length of plaintext is \((13 + 121 + 300) = 434\) bytes; during decryption, the total length of ciphertext is \((13 + 300 + 199) = 512\) bytes. Thus, 512 is both a multiple of 128, and greater than 434, satisfying the constraint.

**Code Example 3-6. Use of \texttt{triDesEncrypt} / \texttt{triDesDecrypt} Interface**

```c
#include "3des.h"
#include "mem.h"

#define PT_LEN 434 /* Length of plain text */

typedef struct /* example structure that can be used in the callback function */
{
    char buffer[512];
    UWord16 offset;
} WriteOutput;

/* Callback function prototype */
void Callback (void *pCallbackArg, char *pChars, UWord16 NumChars);

void test3Des (void)
{
    char plaintext[PT_LEN] = {0x0021, 0x007A, 0x00D5, .....};
    tri_des_sConfigure *pConfig;
    tri_des_sHandle *p3Des;
    UWord16 cntrl;
    tri_des_skey *pkey;
    UWord16 Command = TRIDES_DEACTIVATE;
    WriteOutput P_Text;
    WriteOutput C_Text;
    Int16 res;

    P_Text.offset = 0; /* offset is used in callback, see the example on callback above */
    C_Text.offset = 0; /* offset is used in callback, see the example on callback above */

    /* Memory allocation */
    pConfig = (tri_des_sConfigure *) memMallocEM (sizeof (tri_des_sConfigure));
    pkey = (tri_des_skey *) memMallocEM (sizeof (tri_des_skey));

    /* Initialize the keys */
    pkey->key1[0] = 0x006C;
    pkey->key1[1] = 0x0060;
    pkey->key1[2] = 0x00F2;
    pkey->key1[3] = 0x0060;
    pkey->key1[4] = 0x003E;
    pkey->key1[5] = 0x0016;
    pkey->key1[6] = 0x00CC;
    pkey->key1[7] = 0x00A0;
    pkey->key2[0] = 0x0061;
    pkey->key2[1] = 0x00A4;
    pkey->key2[2] = 0x00F3;
```

For More Information On This Product, Go to: www.freescale.com
pkey->key2[3] = 0x0064;
pkey->key2[4] = 0x0035;
pkey->key2[5] = 0x0016;
pkey->key2[6] = 0x00C7;
pkey->key2[7] = 0x00A8;

pkey->key3[0] = 0x002C;
pkey->key3[1] = 0x0030;
pkey->key3[2] = 0x0042;
pkey->key3[3] = 0x0050;
pkey->key3[4] = 0x006E;
pkey->key3[5] = 0x0076;
pkey->key3[6] = 0x008C;
pkey->key3[7] = 0x0090;

/* Initialize pConfig structure for ENCRYPTION */
pConfig->Flags = TRC_DES_ENCRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&C_Text);

p3Des = triDesCreate (pConfig);
...  
res = triDesEncrypt (p3Des, plaintext, 13); /* Encrypt 13 bytes of data */
...  
res = triDesEncrypt (p3Des, &plaintext[13], 121); /* Encrypt 121 bytes of data */
...  
res = triDesEncrypt (p3Des, &plaintext[134], 300); /* Encrypt 300 bytes of data */
......
......

/* Initialize pConfig structure for DECRYPTION */
pConfig->Flags = TRC_DES_DECRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&P_Text);

p3Des = triDesCreate (pConfig); /* Create instance for decryption */
/* Create instance for decryption */
res = triDesDecrypt (p3Des, C_Text.buffer, 13); /* Decrypt 13 bytes */
...  
res = triDesDecrypt (p3Des, &C_Text.buffer[13], 300); /* Decrypt 300 bytes */
...  
res = triDesDecrypt (p3Des, &C_Text.buffer[313], 199); /* Decrypt 199 bytes */
......
......}
3.3.5 triDesControl

Call(s):

Result triDesControl(tri_des_sHandle *p3Des, UWord16 Command);

Required Header: “3des.h”

Arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*p3Des</td>
<td>Handle to an instance of 3DES</td>
</tr>
<tr>
<td>Command</td>
<td>The command to be executed by the triDesControl procedure</td>
</tr>
</tbody>
</table>

Description: The triDesControl function provides control functions to the 3DES algorithm. If you must terminate 3DES during the encryption/decryption process, call triDesControl. This function flushes the data to be encrypted/decrypted, even if it doesn’t form a proper block required for 3DES operation, by appending zeros, encrypting/decrypting, then calling the Callback procedure.

The parameter *p3Des must be generated from a call to triDesCreate, or must be generated by the user himself if the triDesCreate function was not used. The parameter Command determines which action the triDesControl algorithm will perform, including:

- TRI_DES_DEACTIVATE - Deactivates the 3DES operation.

Returns: Upon successful completion, triDesControl will return “PASS”; otherwise, “FAIL” is returned.

Special Considerations: Calling the triDesControl function does not free the memory allocated during the triDesCreate function. To deallocate buffers, the triDesDestroy function must be called only if triDesCreate function was used to create the instance. If user himself bypassed the triDesCreate function to create the instance, then the user must free the memory.

Code Example 3-7. Use of triDesControl Interface

```c
#include “3des.h”
#include “mem.h”

#define PT_LEN 434 /* Length of plain text */

typedef struct /* example structure that can be used in the callback function */
{
    char buffer[512];
    UWord16 offset;
} WriteOutput;

/* Callback function prototype */
void Callback (void *pCallbackArg, char *pChars, UWord16 NumChars);

void test3Des (void)
{
    char plaintext[PT_LEN] = {0x0021, 0x007A, 0x00D5, .....};
    tri_des_sConfigure *pConfig;
    tri_des_sHandle *p3Des;
    UWord16 cntrl;
    tri_des_skey *pkey;
    UWord16 Command = TRI_DES_DEACTIVATE;
    WriteOutput P_Text;
```
WriteOutput C_Text;
Int16 res;

P_Text.offset = 0; /* offset is used in callback, see the example on
    callback above */
C_Text.offset = 0; /* offset is used in callback, see the example on
    callback above */

/* Memory allocation */
pConfig = (tri_des_sConfigure *) memMallocEM (sizeof (tri_des_sConfigure));
pkey = (tri_des_skey *) memMallocEM (sizeof (tri_des_skey));

/* Initialize the keys */
pkey->key1[0] = 0x006C;
pkey->key1[1] = 0x0060;
pkey->key1[2] = 0x00F2;
pkey->key1[3] = 0x0060;
pkey->key1[4] = 0x003E;
pkey->key1[5] = 0x0016;
pkey->key1[6] = 0x00CC;
pkey->key1[7] = 0x00A0;

pkey->key2[0] = 0x0061;
pkey->key2[1] = 0x0062;
pkey->key2[2] = 0x00F3;
pkey->key2[3] = 0x0064;
pkey->key2[4] = 0x0035;
pkey->key2[5] = 0x0016;
pkey->key2[6] = 0x00C7;
pkey->key2[7] = 0x00A8;

pkey->key3[0] = 0x002C;
pkey->key3[1] = 0x0030;
pkey->key3[2] = 0x0042;
pkey->key3[3] = 0x0050;
pkey->key3[4] = 0x006E;
pkey->key3[5] = 0x0076;
pkey->key3[6] = 0x008C;
pkey->key3[7] = 0x0090;

/* Initialize pConfig structure for ENCRYPTION */
pConfig->Flags = TRI_DES_ENCRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&C_Text);

p3Des = triDesCreate (pConfig);
...
res = triDesEncrypt (p3Des, plaintext, 13); /* Encrypt 13 bytes of data */
... 
res = triDesEncrypt (p3Des, &plaintext[13], 121); /* Encrypt 121 bytes of data */
... 
res = triDesEncrypt (p3Des, &plaintext[134], 300); /* Encrypt 300 bytes of data */

res = triDesControl (p3Des, Command); /* Stops the encryption process */
/* Initialize pConfig structure for DECRYPTION */

pConfig->Flags = TRI_DES_DECRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&P_Text);

p3Des = triDesCreate (pConfig); /* Create instance for decryption */

res = triDesDecrypt (p3Des, C_Text.buffer, 13); /* Decrypt 13 bytes */
...  
res = triDesDecrypt (p3Des, &C_Text.buffer[13], 300); /* Decrypt 300 bytes */
...  
res = triDesDecrypt (p3Des, &C_Text.buffer[313], 199); /* Decrypt 199 bytes */

res = triDesControl (p3Des, Command); /* Stops the decryption process */

...
3.3.6  \texttt{triDesDestroy}

Call(s):

\[
\text{Result } \text{triDesDestroy} (\text{tri}_\text{des}_\text{sHandle} * p3\text{Des})
\]

\textbf{Required Header:} “3des.h”

\textbf{Arguments:}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textit{p3Des} & \textit{in} & Handle to an instance of 3DES generated by a call to \textit{triDesCreate} \\
\hline
\end{tabular}
\end{table}

\textbf{Description:} The \textit{triDesDestroy} function destroys the instance of the 3DES originally created by a call to \textit{triDesCreate}. It also internally calls an encryption or decryption algorithm to complete the operation on any residual non-complete blocks by appending zeros.

\textbf{Returns:} PASS or FAIL

\textbf{Special Considerations:} Calling the \textit{triDesControl} function frees the memory allocated during the \textit{triDesCreate} function. The \textit{triDesDestroy} function deactivates 3DES and frees the memory allocated during \textit{triDesCreate} function.

\textbf{Code Example 3-8. Use of \textit{triDesDestroy} Interface}

\begin{verbatim}
#include "3des.h"
#include "mem.h"
#define PT_LEN 434 /* Length of plain text */

typedef struct /* example structure that can be used in the callback function */
{
    char buffer[512];
    UWord16 offset;
} WriteOutput;

/* Callback function prototype */
void Callback (void *pCallbackArg, char *pChars, UWord16 NumChars);

void test3Des (void)
{
    char plaintext[PT_LEN] = {0x0021, 0x007A, 0x00D5, ....};
    tri_des_sConfigure *pConfig;
    tri_des_sHandle *p3Des;
    UWord16 cntrl;
    tri_des_skey *pkey;
    UWord16 Command = TRI_DES_DEACTIVATE;
    WriteOutput P_Text;
    WriteOutput C_Text;
    Int16 res;

    P_Text.offset = 0; /* offset is used in callback, see the example on callback above */
    C_Text.offset = 0; /* offset is used in callback, see the example on callback above */

    // Code...
}
\end{verbatim}
/* Memory allocation */
pConfig = (tri_des_sConfigure *) memMallocEM (sizeof (tri_des_sConfigure));
pkey = (tri_des_skey *) memMallocEM (sizeof (tri_des_skey));

/* Initialize the keys */
pkey->key1[0] = 0x006C;
pkey->key1[1] = 0x0060;
pkey->key1[2] = 0x00F2;
pkey->key1[3] = 0x0060;
pkey->key1[4] = 0x003E;
pkey->key1[5] = 0x0016;
pkey->key1[6] = 0x00CC;
pkey->key1[7] = 0x00A0;
pkey->key2[0] = 0x0061;
pkey->key2[1] = 0x0062;
pkey->key2[2] = 0x00F3;
pkey->key2[3] = 0x0064;
pkey->key2[4] = 0x0035;
pkey->key2[5] = 0x0016;
pkey->key2[6] = 0x00C7;
pkey->key2[7] = 0x00A8;
pkey->key3[0] = 0x002C;
pkey->key3[1] = 0x0030;
pkey->key3[2] = 0x0042;
pkey->key3[3] = 0x0050;
pkey->key3[4] = 0x006E;
pkey->key3[5] = 0x0076;
pkey->key3[6] = 0x008C;
pkey->key3[7] = 0x0090;

/* Initialize pConfig structure for ENCRYPTION */
pConfig->Flags = TRI_DES_ENCRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&C_Text);

p3Des = triDesCreate (pConfig);
...
res = triDesEncrypt (p3Des, plaintext, 13); /* Encrypt 13 bytes of data */
...
res = triDesEncrypt (p3Des, &plaintext[13], 121); /* Encrypt 121 bytes of data */
...
res = triDesEncrypt (p3Des, &plaintext[134], 300); /* Encrypt 300 bytes of data */
res = triDesDestroy (p3Des); /* Destroys 3DES instance for decryption */
......

/* Initialize pConfig structure for DECRYPTION */
pConfig->Flags = TRI_DES_DECRYPT;
pConfig->pKey = pkey;
pConfig->Callback.pCallback = Callback;
pConfig->Callback.pCallbackArg = (WriteOutput *) (&P_Text);

...
p3Des = triDesCreate (pConfig); /* Create instance for decryption */
...
res = triDesDecrypt (p3Des, C_Text.buffer, 13); /* Decrypt 13 bytes */
...
res = triDesDecrypt (p3Des, &C_Text.buffer[13], 300); /* Decrypt 300 bytes */
...
res = triDesDecrypt (p3Des, &C_Text.buffer[313], 199); /* Decrypt 199 bytes */
res = triDesDestroy (p3Des); /* Destroys 3DES instance for decryption */

......
......
Chapter 4
Building the 3DES Library

4.1 Building the 3DES Library

The 3DES library combines all of the components described in the previous sections into one library: 3des.lib. To build this library, a MetroWerks’ CodeWarrior project, 3des.mcp, is provided. This project and all the necessary components to build the 3DES library are located in the ...\uos\security\3des directory of the SDK directory structure.

There are two methods to execute a system library project build: dependency build and direct build.

4.1.1 Dependency Build

Dependency build is the easiest approach and requires no additional work on the user’s part. If you add the 3DES library project, 3des.mcp, to your application project, as shown in Figure 4-1, the 3DES library will automatically build when the application is built.
4.1.2 Direct Build

Direct build allows you to create a 3DES library independently of any other build. Follow these steps for a direct build:

**Step 1.** Open project *3des.mcp*, as shown in Figure 4-2.

![Figure 4-2. 3des.mcp Project](image)

**Step 2.** Execute the build by pressing function key [F7] or by choosing *Make* from the Project menu; see Figure 4-3.

![Figure 4-3. Execute Make](image)

At this point, if the build has been successful, a *3des.lib* library file is created in ...
\n\n\n\n...\n\n\n...\nos\security\3des\Debug directory.
Chapter 5
Linking Applications with the 3DES Library

5.1 3DES Library
The 3DES library contains 3DES code and Application Programming Interfaces (APIs), which provide an interface between the user application and the 3DES modules. To invoke 3DES encryption/decryption, APIs must be called in this order:

- triDesCreate (......);
- triDesInit (......);
- triDesEncrypt (......); or triDesDecrypt (......); as required
- triDesControl (......); or triDesDestroy (......);

5.1.1 Library Sections
The linker command file, linker.cmd, used in the test application, is shown in Code Example 5-1.

Code Example 5-1. linker.cmd File

# Linker.cmd file for DSP56824EVM External RAM
# using both internal and external data memory (EX = 0)
# and using external program memory (Mode = 3)

MEMORY {
    .pvec (RWX) : ORIGIN = 0x0000, LENGTH = 0x002C  # interrupt vector table ( 22 * 2 )
    .pram (RWX) : ORIGIN = 0x002C, LENGTH = 0xFFD4  # external program memory
    .avail (RW) : ORIGIN = 0x0000, LENGTH = 0x0030  # available
    .cwregs (RW) : ORIGIN = 0x0030, LENGTH = 0x0010  # C temp registers in CodeWarrior
    .im1 (RW) : ORIGIN = 0x0040, LENGTH = 0x07C0  # data 1
    .rom (R) : ORIGIN = 0x0800, LENGTH = 0x0800  # internal data ROM
}
FORCE_ACTIVE {FconfigInterruptVector}

SECTIONS {

  # Data (X) Memory Layout
  
  # Internal Memory Partitions (for mem.h partitions)
  _NUM_IM_PARTITIONS = 2;  # .im1 and .im2

  # External Memory Partition (for mem.h partitions)
  _NUM_EM_PARTITIONS = 1;   # .em

  .main_application_vector :
  { 
    # .text sections
    
    # vector.c MUST be placed into .pvec, otherwise the Interrupt Vector
    # configInterruptVector will not be located at the correct address,
    # P:0x0000

    vector.c (.text)
  } > .pvec

  .main_application_code :
  { 
    # .text sections
    
    * (.text)
    * (rtlib.text)
    * (fp_engine.text)
    * (user.text)
  } > .pram

  .main_application_data :
  { 
    # Define variables for C initialization code
    
}
F_Xdata_start_addr_in_ROM = ADDR(.rom) + SIZEOF(.rom) / 2;
F_StackAddr = ADDR(.stack);
F_StackEndAddr = ADDR(.stack) + SIZEOF(.stack) / 2 - 1;

F_Xdata_start_addr_in_RAM = .;

#
# Memory layout data for SDK INCLUDE_MEMORY (mem.h) support
#
FmemEXbit = .;
WRITEH(_EX_BIT);
FmemNumIMpartitions = .;
WRITEH(_NUM_IM_PARTITIONS);
FmemNumEMpartitions = .;
WRITEH(_NUM_EM_PARTITIONS);
FmemIMpartitionList = .;
WRITEH(ADDR(.im1));
WRITEH(SIZEOF(.im1) / 2);
WRITEH(ADDR(.im2));
WRITEH(SIZEOF(.im2) / 2);
FmemEMpartitionList = .;
WRITEH(ADDR(.em));
WRITEH(SIZEOF(.em) /2);

# .data sections

* (.data)
* (fp_state.data)
* (rtlib.data)

F_Xdata_ROMtoRAM_length = 0;

F_bss_start_addr = .;
_BSS_ADDR = .;

* (rtlib.bss.lo)
* (.bss)

F_bss_length = . - _BSS_ADDR;  # Copy DATA

} > .data

FArchIO = ADDR(.onchip2);
}
Chapter 6
3DES Applications

6.1 Test and Demo Applications

To verify the Triple DES algorithm, test and demo applications have been developed. Refer to the Targeting Motorola DSP568xx Platform Manual for the DSP you are using to see if the test and demo applications are available for your target.
Chapter 7
License

7.1 Limited Use License Agreement

LIMITED USE LICENSE AGREEMENT

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THIS SOFTWARE. BY USING OR COPYING THE SOFTWARE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.

The software in either source code form ("Source") or object code form ("Object") (cumulatively hereinafter "Software") is provided under a license agreement ("Agreement") as described herein. Any use of the Software including copying, modifying, or installing the Software so that it is usable by or accessible by a central processing unit constitutes acceptance of the terms of the Agreement by the person or persons making such use or, if employed, the employer thereof ("Licensee") and if employed, the person(s) making such use hereby warrants that they have the authority of their employer to enter this license agreement. If Licensee does not agree with and accept the terms of this Agreement, Licensee must return or destroy any media containing the Software or materials related thereto, and destroy all copies of the Software.

The Software is licensed to Licensee by Motorola Incorporated ("Motorola") for use under the terms of this Agreement. Motorola retains ownership of the Software. Motorola grants only the rights specifically granted in this Agreement and grants no other rights. Title to the Software, all copies thereof and all rights therein, including all rights in any intellectual property including patents, copyrights, and trade secrets applicable thereto, shall remain vested in Motorola.

For the Source, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to use, copy, and make derivatives of the Source solely in a development system environment in order to produce object code solely for operating on a Motorola semiconductor device having a central processing unit ("Derivative Object").

For the Object and Derivative Object, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to copy, use, and distribute the Object and the Derivative Object solely for operating on a Motorola semiconductor device having a central processing unit.

Licensee agrees to: (a) not use, modify, or copy the Software except as expressly provided herein, (b) not distribute, disclose, transfer, sell, assign, rent, lease, or otherwise make available the Software, any derivatives thereof, or this license to a third party except as expressly provided herein, (c) not remove obliterate, or otherwise defeat any copyright, trademark, patent or proprietary notices, related to the Software (d) not in any form export, re-export, resell, ship or divert or cause to be exported, re-exported, resold, shipped, or diverted, directly or indirectly, the Software or a direct product thereof to any country which the United States government or any agency thereof at the time of export or re-export requires an export license or other government approval without first obtaining such license or approval.
THE SOFTWARE IS PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MOTOROLA BE LIABLE FOR ANY LIABILITY OR DAMAGES OF ANY KIND INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT OR INCIDENTAL OR CONSEQUENTIAL OR PUNITIVE DAMAGES OR LOST PROFITS OR LOSS OF USE ARISING FROM USE OF THE SOFTWARE OR THE PRODUCT REGARDLESS OF THE FORM OF ACTION OR THEORY OF LIABILITY (INCLUDING WITHOUT LIMITATION, ACTION IN CONTRACT, NEGLIGENCE, OR PRODUCT LIABILITY) EVEN IF MOTOROLA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS DISCLAIMER OF WARRANTY EXTENDS TO LICENSEE OR USERS OF PRODUCTS AND IS IN LIEU OF ALL WARRANTIES WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE.

Motorola does not represent or warrant that the Software is free of infringement of any third party patents, copyrights, trade secrets, or other intellectual property rights or that Motorola has the right to grant the licenses contained herein. Motorola does not represent or warrant that the Software is free of defect, or that it meets any particular requirements or need of the Licensee, or that it conforms to any documentation, or that it meets any standards.

Motorola shall not be responsible to maintain the Software, provide upgrades to the Software, or provide any field service of the Software. Motorola reserves the right to make changes to the Software without further notice to Licensee.

The Software is not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Software could create a situation where personal injury or death may occur. Should Licensee purchase or use the Software for any such unintended or unauthorized application, Licensee shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the Software.

The term of this Agreement is for as long as Licensee uses the Software for its intended purpose and is not in default of any provisions of this Agreement. Motorola may terminate this Agreement if Licensee is in default of any of the terms and conditions of this Agreement.

This Agreement shall be governed by and construed in accordance with the laws of the State of Arizona and can only be modified in a writing signed by both parties. Licensee agrees to jurisdiction and venue in the State of Arizona.

By using, modifying, installing, compiling, or copying the Software, Licensee acknowledges that this Agreement has been read and understood and agrees to be bound by its terms and conditions. Licensee agrees that this Agreement is the complete and exclusive statement of the agreement between Licensee and Motorola and supersedes any earlier proposal or prior arrangement, whether oral or written, and any other communications relative to the subject matter of this Agreement.
Index

Numerics
3DES x

D
DES x
DSP x
DSP56800 Family Manual xi
DSP56824 User’s Manual xi

E
Embedded SDK Programmer’s Guide xi

F
FFT x
FIR x

I
I/O xi
I/O Services 3-1
IDE xi
IIR xi

L
LSB xi

M
MAC xi
MIPS xi
MSB xi

O
OMR xi
OnCE xi

P
PC xi

S
SDK xi
SP xi
SPI xi
SR xi
SRC xi