Center for Embedded Computer Systems
University of California, Irvine

System Specification of a DES Cipher Chip

Weiwei Chen, Rainer Dömer

Technical Report CECS-08-01
January 28, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{weiweic, doemer}@uci.edu
http://www.cecs.uci.edu/

System Specification of a DES Cipher Chip

Weiwei Chen, Rainer Dömer

Technical Report CECS-08-01
January 28, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{weiweic, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

The development complexity of embedded system design is growing rapidly due to its great computational and high performance demands. System level design becomes more and more important in order to meet these competitive requirements. In this report, we build up a system specification of a DES cipher system in a System Level Design Language (SLDL), SpecC. The model is successfully specified and simulated at the system level.

Contents

List of Figures

List of Tables

List of Acronyms

DES Data Encryption Standard. A cryptographic algorithm used to protect sensitive data.

SLDL System Level Design Language.

System Specification of a DES Cipher Chip

Weiwei Chen, Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

{weiweic, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

The development complexity of embedded system design is growing rapidly due to its great computational and high performance demands. System level design becomes more and more important in order to meet these competitive requirements. In this report, we build up a system specification of a DES cipher system in a System Level Design Language (SLDL), SpecC. The model is successfully specified and simulated at the system level.

1 Introduction

This report describes the SpecC system specification of an information security chip for Data Encryption Standard. Data Encryption Standard (DES) [1] is a cryptographic algorithm which is used to protect sensitive data. It uniquely defines the mathematical steps to transform data into a cryptographic cipher and also to transform the cipher back to the original form. A chip based on the DES cipher algorithm could be built to achieve high encryption / decryption performance. In this report, we will describe the system specification of such a cipher chip in System Level Design Language, SpecC.

The report is organized as follows: The brief introduction to DES algorithm and the DES cipher chip is made in Section 1. The system specification is described in Section 2. Simulation result is presented in Section 3. Conclusions are made in Section 4.

2 DES Cipher System

The input for the encryption procedure of DES requires a key consisting of 64-bit binary digits and the 64-bit data to be encrypted. The input for the decryption procedure of DES also requires a 64-bit key as well as the cipher to be decrypted. Data can be recovered from cipher only by using exactly the same key used to encipher it [1].

In this report, we build the system specification of a DES cipher chip in SpecC. The chip has two functions: DES encipher and DES decipher.

There are three behavior modules in the top ***testbench*** module: They are ***des***, the system model of the DES chip; ***stimulus***, the one who provides the test vectors; and ***monitor***, the one receives the DES cipher results and checks the correctness of the ***des*** function module. Figure 1 shows the top level hierarchy of this system.

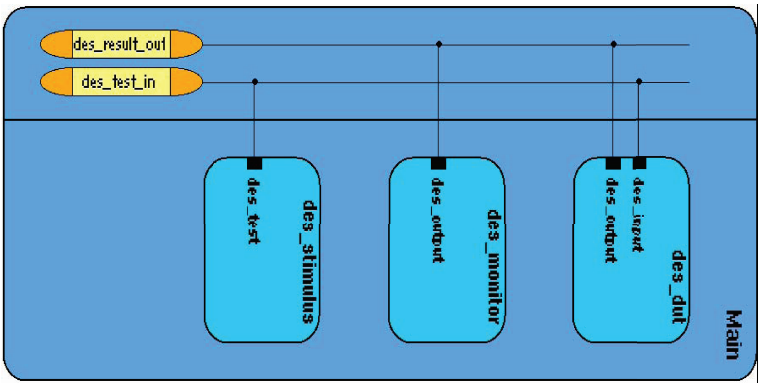


Figure 1: Top Level Hierarchy of the DES Cipher System

The ***des*** acts as the main function unit of the DES cipher chip. It combines the cipher key generator, the decryption module and the encryption module together. Table 1 lists the submodules in the DES cipher chip.

Table 1: DES Cipher Chip Module Description

	Module Name	Functionality
1	des_SetKey	setting the basic cipher subkey set
2	des_Get_Key	getting the cipher subkey set
3	des_Swap_Key	swapping the cipher keys in a subkey set
4	des_SetKey_enc	generating the key for encryption
5	des_SetKey_dec	generating the key for decryption
6	des_Crypt_ECB	executing the cipher algorithms

2.1 des_SetKey_enc

The module which generates the key for data encryption is shown in Figure 2a. The input of this module is user defined 64-bit binary digits key. The output of this module is a subkey set consists of 32 32-bit binary digits. The encryption key is generated by ***des_SetKey*** module which generates the subkey set according to DES algorithm. In Figure 2a, the instance name of the *des_SetKey* module is ***set_enc_key*** and the instance name of the *des_Get_Key* is ***get_key***.

2.2 des_SetKey_dec

The module which generates the key for data decryption is shown in Figure 2b. The input of this module is user defined 64-bit binary digits key. The output of this module is a subkey set consists of 32 32-bit binary digits. The decryption key is first generated by ***des_SetKey*** module which generates the subkey set according to DES algorithm. Then the subkey set is symmetrically swapped two by two in the ***des_Swap_Key*** module. In Figure 2b, the instance name of the *des_SetKey* module is ***set_dec_key*** and the instance name of the *des_Swap_Key* is ***swap_key***.

2.3 des_Crypt_ECB

The module which executes the DES cipher algorithm is shown in Figure 2c. The input of this module is the 64-bit data to be ciphered and the corresponding 32 32-bits subkey set. The output of this module is the 64-bit ciphered data.

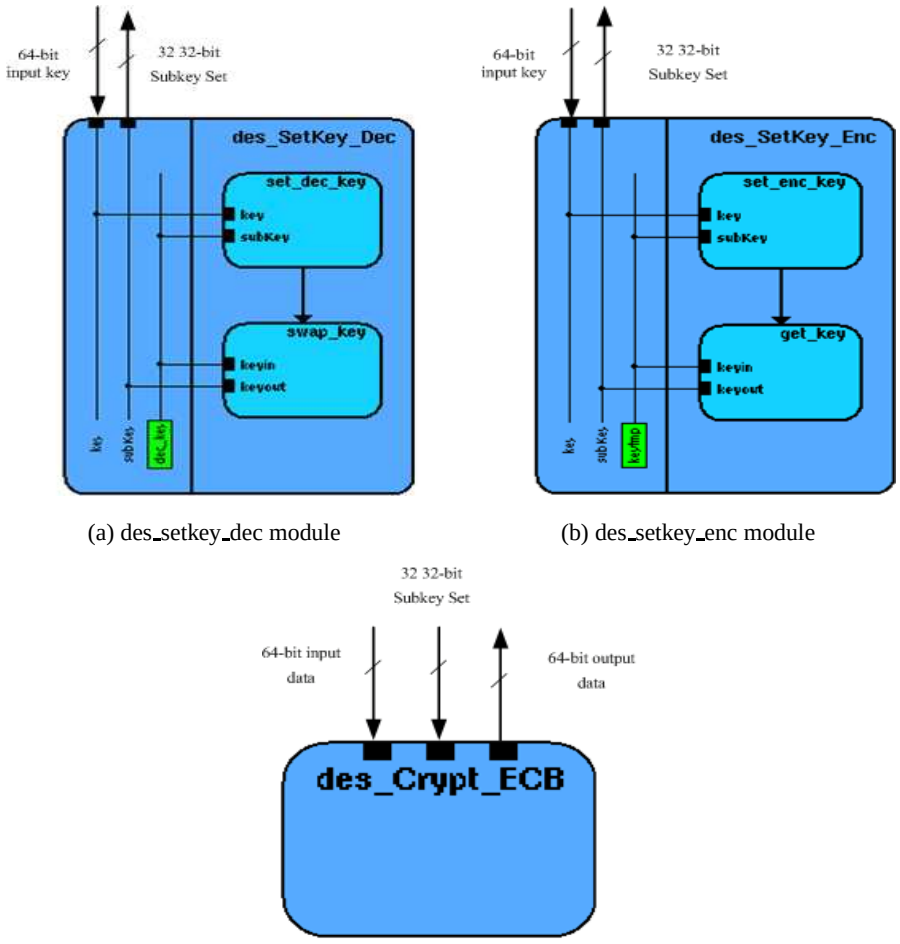
In the top ***testbench*** level, we use two double handshake channels for the communication between the ***des*** function module and the ***stimulus***, ***monitor*** modules. Shared variables are used among the sub-function modules in ***des*** for the sharing of the cipher keys and data as well. Figure 3 shows the full hierarchy of the system. The names of the submodules in Figure 3 are represented by their corresponding instance names. Section A.1 lists the instantiation hierarchy of behaviors and channels contained in the whole system.

3 System Simulation

We provide three test modes in this cipher chip system model: **user defined encryption**, **user defined decryption** and **self-test**. Both the user defined encryption and decryption mode allow the user to provide a certain 64-bit key and the corresponding 64-bit plaintext for encryption or 64-bit cryptographic cipher for decryption. The self-test mode provides 3 sets of keys and 7 sets of plaintext to first encrypt the plaintext, then decrypt the encrypted result, and finally compares the decrypted data with the original plaintext. All of the 21 test cases passed successfully.

4 Conclusion

This is a small project for system modeling in SpecC. We select a simple cipher algorithm, find its reference C code and then convert it into SpecC to build a system model. The difference between



(a) des_setkey_dec module

(b) des_setkey_enc module

(c) des_crypt_ecb module

Figure 2: Main submodules of *des* module.

the C reference code and the SpecC model is that we should have the concept of hardware which connected different parallel running modules together when building the system model rather than passing variables between different sequentially executed procedures in C programming. Moreover, the communication between different modules is also very important in SpecC system specification.

References

[1] National Bureau of Standards. FlPS PUB 46: *Data Encryption Standard*, October 1999.

[2] http://xyssl.org/code/download/xyssl-0.8.zip

[3] Rainer Dömer, Andreas Gerstlauer, Daniel D. Gajski. *SpecC Language Reference Manual, Version 2.0* December, 2002.

[4] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. *SpecC: Specification Language and Design Methodology.* Kluwer, 2000.

[5] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC.* Kluwer, 2001.

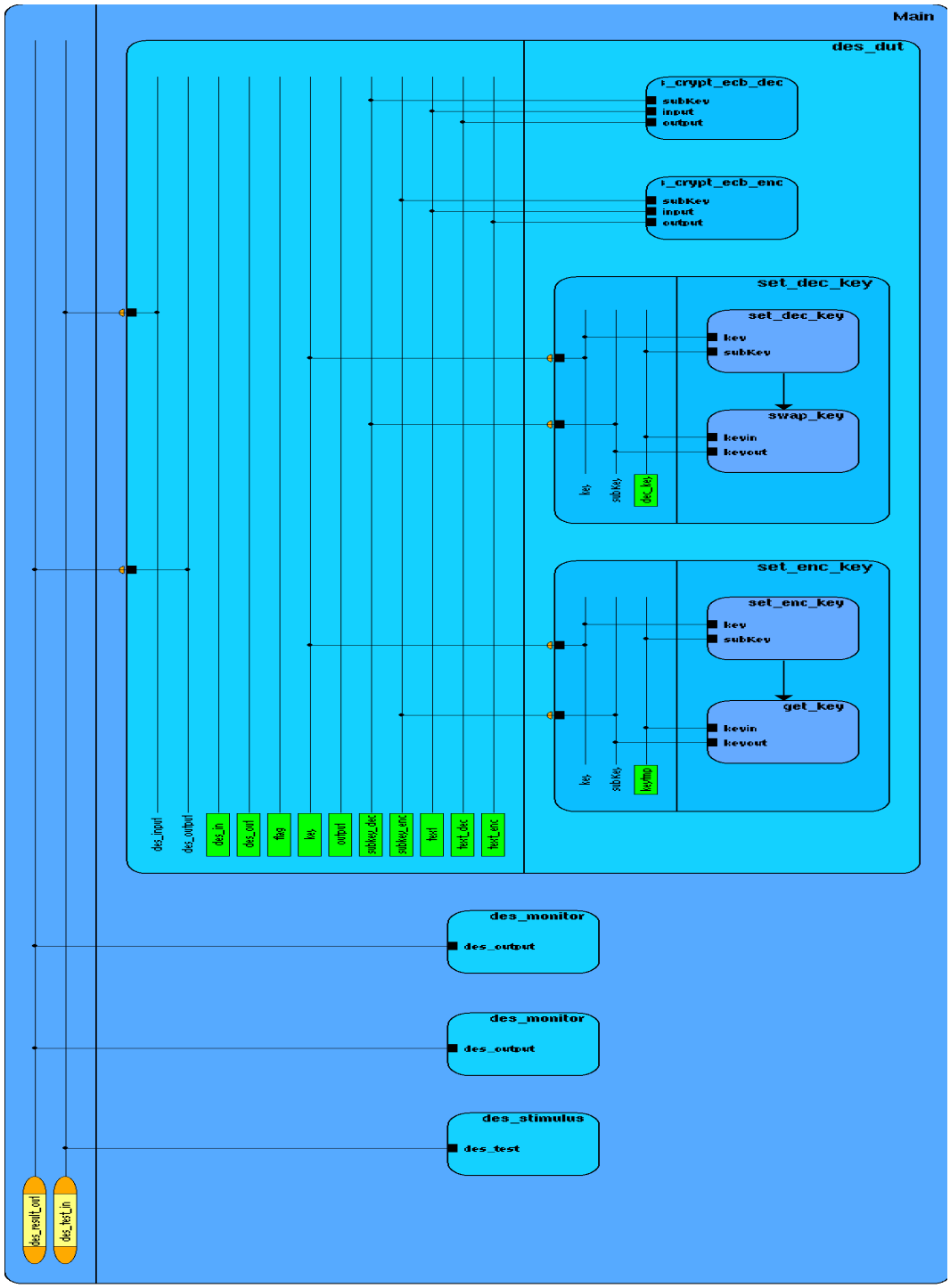


Figure 3: Hierarchy of the DES Cipher System

A Appendix

A.1 System Instantiation Hierarchy of Behaviors and Channels

```
Main
|------ des_dut
|       |------ des_crypt_ecb_dec
|       |------ des_crypt_ecb_enc
|       |------ set_dec_key
|       |       |------ set_dec_key
|       |       \------ swap_key
|       \------ set_enc_key
|               |------ get_key
|               \------ set_enc_key
|------ des_monitor
|------ des_stimulus
|------ des_result_out
\------ des_test_in
```

Note: The graphical list of the instantiation hierarchy above is generated by sir_tree, a tool of the SpecC SIR tool **set**.

A.2 Source Code

A.2.1 testbench.sc

```
/* ------------------------------------------------------------------------
 * testbench.sc - testbench module for Data Encryption Standard (DES) Algorithm
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */
import "des";
import "stimulus";
import "monitor";
import "c_double_handshake";

behavior Main(void)
{
  c_double_handshake       des_test_in;
  c_double_handshake       des_result_out;

  des      des_dut(des_test_in, des_result_out);
  stimulus  des_stimulus(des_test_in);
  monitor    des_monitor(des_result_out);

  int main(void)
  {
    par
    {
      des_dut.main();
      des_stimulus.main();
      des_monitor.main();
    }
    return(0);
  }
};
```

```
//EOF
```

A.2.2 des.sh

```
/* ------------------------------------------------------------------
 * des.h - Header File of Data Encryption Standard (DES) Algorithm
 * ------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */
#ifndef DES_H
#define DES_H

typedef struct des_input
{
  unsigned char key[8];
  unsigned char buffer[8];
  bit[1:0] flag;
} DES_input;

typedef struct des_output
{
  unsigned char obuffer[8]; // the original buffer
  unsigned char buffer[8];
  bit[1:0] flag;
} DES_output;


#endif

//EOF
```

A.2.3 des.sc

```
/* ------------------------------------------------------------------
 * des.sc - Data Encryption Standard (DES) Algorithm
 * ------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */
#include "des.sh"

import "des_SetKey_Enc";
import "des_SetKey_Dec";
import "des_Crypt_ECB";
import "i_receiver";
import "i_sender";

behavior des(i_receiver des_input,
       i_sender des_output)
{
  DES_input des_in;
  DES_output des_out;

  unsigned char key[8];
  unsigned char text[8];
  bit[1:0] flag;    // 00 encrypt, 01 decrypt, 10 enc-dec
```

```
  unsigned long subkey_enc[32];
  unsigned long subkey_dec[32];
  unsigned char text_enc[8];
  unsigned char text_dec[8];
  unsigned char output[8];

  des_SetKey_Enc  set_enc_key(key, subkey_enc);
  des_SetKey_Dec  set_dec_key(key, subkey_dec);
  des_Crypt_ECB   des_crypt_ecb_enc(subkey_enc, text, text_enc);
  des_Crypt_ECB   des_crypt_ecb_dec(subkey_dec, text, text_dec);

  void main(void)
  {
    while(1)
    {
      des_input.receive(&des_in, sizeof(des_in));
      key = des_in.key;
      text = des_in.buffer;
      flag = des_in.flag;

      des_out.obuffer = text;

      if(flag == 00b)       //DES encryption
      {
        set_enc_key.main();
        des_crypt_ecb_enc.main();

        des_out.flag = flag;
        des_out.buffer = text_enc;
        des_output.send(&des_out, sizeof(des_out));
      }
      else if(flag == 01b)        //DES decryption
      {
        set_dec_key.main();
        des_crypt_ecb_dec.main();

        des_out.flag = flag;
        des_out.buffer = text_dec;
        des_output.send(&des_out, sizeof(des_out));
      }
      else if(flag == 10b)    //DES enc-dec
      {
        par
        {
          set_enc_key.main();
          set_dec_key.main();
        }
        des_crypt_ecb_enc.main();

        text = text_enc;

        des_crypt_ecb_dec.main();

        des_out.flag = flag;
        des_out.buffer = text_dec;
        des_output.send(&des_out, sizeof(des_out));
      }
    }
  }
};
```

```
//EOF
```

A.2.4 des_SetKey.sc

```
/* ------------------------------------------------------------------------
 * des_SetKey.sc - SubKey set calculation module for DES
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */
#include <stdio.h>

/*
 * PC1: left and right halves bit-swap
 */
static const unsigned long LHs[16] =
{
    0x00000000, 0x00000001, 0x00000100, 0x00000101,
    0x00010000, 0x00010001, 0x00010100, 0x00010101,
    0x01000000, 0x01000001, 0x01000100, 0x01000101,
    0x01010000, 0x01010001, 0x01010100, 0x01010101
};

static const unsigned long RHs[16] =
{
    0x00000000, 0x01000000, 0x00010000, 0x01010000,
    0x00000100, 0x01000100, 0x00010100, 0x01010100,
    0x00000001, 0x01000001, 0x00010001, 0x01010001,
    0x00000101, 0x01000101, 0x00010101, 0x01010101,
};

#ifndef GET_ULONG_BE
#define GET_ULONG_BE(n,b,i)                             \
{                                                       \
    (n) = ( (unsigned long) (b)[(i)    ] << 24 )        \
        | ( (unsigned long) (b)[(i) + 1] << 16 )        \
        | ( (unsigned long) (b)[(i) + 2] <<  8 )        \
        | ( (unsigned long) (b)[(i) + 3]       );       \
}
#endif

#ifndef PUT_ULONG_BE
#define PUT_ULONG_BE(n,b,i)                             \
{                                                       \
    (b)[(i)    ] = (unsigned char) ( (n) >> 24 );       \
    (b)[(i) + 1] = (unsigned char) ( (n) >> 16 );       \
    (b)[(i) + 2] = (unsigned char) ( (n) >>  8 );       \
    (b)[(i) + 3] = (unsigned char) ( (n)       );       \
}
#endif

behavior des_SetKey(in unsigned char key[8],
          out unsigned long subKey[32])
{
  void main(void)
  {
    int i, j;
    unsigned long X, Y, T;
```

```
unsigned long SK[32];

//while(true)
//{
//  wait key;
    GET_ULONG_BE( X, key, 0 );  // left group
    GET_ULONG_BE( Y, key, 4 );  // right group

    /*
     * Permuted Choice 1
     */
    T =  ((Y >>  4) ^ X) & 0x0F0F0F0F;  X ^= T; Y ^= (T <<  4);
    T =  ((Y      ) ^ X) & 0x10101010;  X ^= T; Y ^= (T      );

    X =   (LHs[ (X      ) & 0xF] << 3) | (LHs[ (X >>  8) & 0xF ] << 2)
        | (LHs[ (X >> 16) & 0xF] << 1) | (LHs[ (X >> 24) & 0xF ]     )
        | (LHs[ (X >>  5) & 0xF] << 7) | (LHs[ (X >> 13) & 0xF ] << 6)
        | (LHs[ (X >> 21) & 0xF] << 5) | (LHs[ (X >> 29) & 0xF ] << 4);

    Y =   (RHs[ (Y >>  1) & 0xF] << 3) | (RHs[ (Y >>  9) & 0xF ] << 2)
        | (RHs[ (Y >> 17) & 0xF] << 1) | (RHs[ (Y >> 25) & 0xF ]     )
        | (RHs[ (Y >>  4) & 0xF] << 7) | (RHs[ (Y >> 12) & 0xF ] << 6)
        | (RHs[ (Y >> 20) & 0xF] << 5) | (RHs[ (Y >> 28) & 0xF ] << 4);

    X &= 0x0FFFFFFF;
    Y &= 0x0FFFFFFF;

    /*
     * calculate subkeys
     */
    for( i = 0, j = 0; i < 16; i++ )
    {
        if( i < 2 || i == 8 || i == 15 )
        {
            X = ((X <<  1) | (X >> 27)) & 0x0FFFFFFF;
            Y = ((Y <<  1) | (Y >> 27)) & 0x0FFFFFFF;
        }
        else
        {
            X = ((X <<  2) | (X >> 26)) & 0x0FFFFFFF;
            Y = ((Y <<  2) | (Y >> 26)) & 0x0FFFFFFF;
        }

        SK[j ++] =   ((X <<  4) & 0x24000000) | ((X << 28) & 0x10000000)
                   | ((X << 14) & 0x08000000) | ((X << 18) & 0x02080000)
                   | ((X <<  6) & 0x01000000) | ((X <<  9) & 0x00200000)
                   | ((X >>  1) & 0x00100000) | ((X << 10) & 0x00040000)
                   | ((X <<  2) & 0x00020000) | ((X >> 10) & 0x00010000)
                   | ((Y >> 13) & 0x00002000) | ((Y >>  4) & 0x00001000)
                   | ((Y <<  6) & 0x00000800) | ((Y >>  1) & 0x00000400)
                   | ((Y >> 14) & 0x00000200) | ((Y      ) & 0x00000100)
                   | ((Y >>  5) & 0x00000020) | ((Y >> 10) & 0x00000010)
                   | ((Y >>  3) & 0x00000008) | ((Y >> 18) & 0x00000004)
                   | ((Y >> 26) & 0x00000002) | ((Y >> 24) & 0x00000001);

        SK[j ++] =   ((X << 15) & 0x20000000) | ((X << 17) & 0x10000000)
                   | ((X << 10) & 0x08000000) | ((X << 22) & 0x04000000)
                   | ((X >>  2) & 0x02000000) | ((X <<  1) & 0x01000000)
                   | ((X << 16) & 0x00200000) | ((X << 11) & 0x00100000)
                   | ((X <<  3) & 0x00080000) | ((X >>  6) & 0x00040000)
```

```
                    | ((X << 15) & 0x00020000) | ((X >>  4) & 0x00010000)
                    | ((Y >>  2) & 0x00002000) | ((Y <<  8) & 0x00001000)
                    | ((Y >> 14) & 0x00000808) | ((Y >>  9) & 0x00000400)
                    | ((Y      ) & 0x00000200) | ((Y <<  7) & 0x00000100)
                    | ((Y >>  7) & 0x00000020) | ((Y >>  3) & 0x00000011)
                    | ((Y <<  2) & 0x00000004) | ((Y >> 21) & 0x00000002);
        }
      subKey = SK;
    //}
  }
};

//EOF
```

A.2.5 des_SetKey_Enc.sc

```
/* ------------------------------------------------------------------------
 * des_SetKey_Enc.sc - SubKey set calculation module for DES encryption
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */

import "des_SetKey";

behavior des_Get_Key(in unsigned long keyin[32],
            out unsigned long keyout[32])
{
  unsigned long key_tmp[32];

  void main(void)
  {
    //while(true)
    //{
    //  wait keyin;
      key_tmp = keyin;
      keyout = key_tmp;
    //}
  }
};

behavior des_SetKey_Enc(in unsigned char key[8],
            out unsigned long subKey[32])
{
  unsigned long keytmp[32];

  des_SetKey set_enc_key(key, keytmp);
  des_Get_Key get_key(keytmp, subKey);

  void main(void)
  {
    //par
    //{
      set_enc_key;
      get_key;
    //}
  }
};
```

```
//EOF
```

A.2.6 des_SetKey_Dec.sc

```
/* ------------------------------------------------------------------------
 * des_SetKey_Dec.sc - SubKey set calculation module for DES decryption
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */

import "des_SetKey";

#define SWAP(a,b,t) {t = a; a = b; b = t;}

behavior des_Swap_Key(in unsigned long keyin[32],
            out unsigned long keyout[32])
{
  unsigned long key_tmp[32];

  void main(void)
  {
    int i;
    unsigned long tmp;

    //while(true)
    //{
    //  wait keyin;
      key_tmp = keyin;

      for( i = 0; i < 16; i += 2 )
      {
        SWAP(key_tmp[i], key_tmp[30 - i], tmp );
        SWAP(key_tmp[i + 1], key_tmp[31 - i], tmp );
      }
      keyout = key_tmp;
    //}
  }
};

behavior des_SetKey_Dec(in unsigned char key[8],
            out unsigned long subKey[32])
{
  unsigned long dec_key[32];

  des_SetKey set_dec_key(key, dec_key);
  des_Swap_Key swap_key(dec_key, subKey);

  void main(void)
  {
  //  par
  //  {
      set_dec_key;
      swap_key;
  //  }
  }
};

//EOF
```

A.2.7 des_Crypt_ECB.sc

```
/* ------------------------------------------------------------------------
 * des_Crypt_ECB.sc - DES ECB Cryption
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */

#include <stdio.h>

static const unsigned long SB1[64] =
{
    0x01010400, 0x00000000, 0x00010000, 0x01010404,
    0x01010004, 0x00010404, 0x00000004, 0x00010000,
    0x00000400, 0x01010400, 0x01010404, 0x00000400,
    0x01000404, 0x01010004, 0x01000000, 0x00000004,
    0x00000404, 0x01000400, 0x01000400, 0x00010400,
    0x00010400, 0x01010000, 0x01010000, 0x01000404,
    0x00010004, 0x01000004, 0x01000004, 0x00010004,
    0x00000000, 0x00000404, 0x00010404, 0x01000000,
    0x00010000, 0x01010404, 0x00000004, 0x01010000,
    0x01010400, 0x01000000, 0x01000000, 0x00000400,
    0x01010004, 0x00010000, 0x00010400, 0x01000004,
    0x00000400, 0x00000004, 0x01000404, 0x00010404,
    0x01010404, 0x00010004, 0x01010000, 0x01000404,
    0x01000004, 0x00000404, 0x00010404, 0x01010400,
    0x00000404, 0x01000400, 0x01000400, 0x00000000,
    0x00010004, 0x00010400, 0x00000000, 0x01010004
};

static const unsigned long SB2[64] =
{
    0x80108020, 0x80008000, 0x00008000, 0x00108020,
    0x00100000, 0x00000020, 0x80100020, 0x80008020,
    0x80000020, 0x80108020, 0x80108000, 0x80000000,
    0x80008000, 0x00100000, 0x00000020, 0x80100020,
    0x00108000, 0x00100020, 0x80008020, 0x00000000,
    0x80000000, 0x00008000, 0x00108020, 0x80100000,
    0x00100020, 0x80000020, 0x00000000, 0x00108000,
    0x00008020, 0x80108000, 0x80100000, 0x00008020,
    0x00000000, 0x00108020, 0x80100020, 0x00100000,
    0x80008020, 0x80100000, 0x80108000, 0x00008000,
    0x80100000, 0x80008000, 0x00000020, 0x80108020,
    0x00108020, 0x00000020, 0x00008000, 0x80000000,
    0x00008020, 0x80108000, 0x00100000, 0x80000020,
    0x00100020, 0x80008020, 0x80000020, 0x00100020,
    0x00108000, 0x00000000, 0x80008000, 0x00008020,
    0x80000000, 0x80100020, 0x80108020, 0x00108000
};

static const unsigned long SB3[64] =
{
    0x00000208, 0x08020200, 0x00000000, 0x08020008,
    0x08000200, 0x00000000, 0x00020208, 0x08000200,
    0x00020008, 0x08000008, 0x08000008, 0x00020000,
    0x08020208, 0x00020008, 0x08020000, 0x00000208,
    0x08000000, 0x00000008, 0x08020200, 0x00000200,
    0x00020200, 0x08020000, 0x08020008, 0x00020208,
```

```
    0x08000208, 0x00020200, 0x00020000, 0x08000208,
    0x00000008, 0x08020208, 0x00000200, 0x08000000,
    0x08020200, 0x08000000, 0x00020008, 0x00000208,
    0x00020000, 0x08020200, 0x08000200, 0x00000000,
    0x00000200, 0x00020008, 0x08020208, 0x08000200,
    0x08000008, 0x00000200, 0x00000000, 0x08020008,
    0x08000208, 0x00020000, 0x08000000, 0x08020208,
    0x00000008, 0x00020208, 0x00020200, 0x08000008,
    0x08020000, 0x08000208, 0x00000208, 0x08020000,
    0x00020208, 0x00000008, 0x08020008, 0x00020200
};

static const unsigned long SB4[64] =
{
    0x00802001, 0x00002081, 0x00002081, 0x00000080,
    0x00802080, 0x00800081, 0x00800001, 0x00002001,
    0x00000000, 0x00802000, 0x00802000, 0x00802081,
    0x00000081, 0x00000000, 0x00800080, 0x00800001,
    0x00000001, 0x00002000, 0x00800000, 0x00802001,
    0x00000080, 0x00800000, 0x00002001, 0x00002080,
    0x00800081, 0x00000001, 0x00002080, 0x00800080,
    0x00002000, 0x00802080, 0x00802081, 0x00000081,
    0x00800080, 0x00800001, 0x00802000, 0x00802081,
    0x00000081, 0x00000000, 0x00000000, 0x00802000,
    0x00002080, 0x00800080, 0x00800081, 0x00000001,
    0x00802001, 0x00002081, 0x00002081, 0x00000080,
    0x00802081, 0x00000081, 0x00000001, 0x00002000,
    0x00800001, 0x00002001, 0x00802080, 0x00800081,
    0x00002001, 0x00002080, 0x00800000, 0x00802001,
    0x00000080, 0x00800000, 0x00002000, 0x00802080
};

static const unsigned long SB5[64] =
{
    0x00000100, 0x02080100, 0x02080000, 0x42000100,
    0x00080000, 0x00000100, 0x40000000, 0x02080000,
    0x40080100, 0x00080000, 0x02000100, 0x40080100,
    0x42000100, 0x42080000, 0x00080100, 0x40000000,
    0x02000000, 0x40080000, 0x40080000, 0x00000000,
    0x40000100, 0x42080100, 0x42080100, 0x02000100,
    0x42080000, 0x40000100, 0x00000000, 0x42000000,
    0x02080100, 0x02000000, 0x42000000, 0x00080100,
    0x00080000, 0x42000100, 0x00000100, 0x02000000,
    0x40000000, 0x02080000, 0x42000100, 0x40080100,
    0x02000100, 0x40000000, 0x42080000, 0x02080100,
    0x40080100, 0x00000100, 0x02000000, 0x42080000,
    0x42080100, 0x00080100, 0x42000000, 0x42080100,
    0x02080000, 0x00000000, 0x40080000, 0x42000000,
    0x00080100, 0x02000100, 0x40000100, 0x00080000,
    0x00000000, 0x40080000, 0x02080100, 0x40000100
};

static const unsigned long SB6[64] =
{
    0x20000010, 0x20400000, 0x00004000, 0x20404010,
    0x20400000, 0x00000010, 0x20404010, 0x00400000,
    0x20004000, 0x00404010, 0x00400000, 0x20000010,
    0x00400010, 0x20004000, 0x20000000, 0x00004010,
    0x00000000, 0x00400010, 0x20004010, 0x00004000,
    0x00404000, 0x20004010, 0x00000010, 0x20400010,
```

```
    0x20400010, 0x00000000, 0x00404010, 0x20404000,
    0x00004010, 0x00404000, 0x20404000, 0x20000000,
    0x20004000, 0x00000010, 0x20400010, 0x00404000,
    0x20404010, 0x00400000, 0x00004010, 0x20000010,
    0x00400000, 0x20004000, 0x20000000, 0x00004010,
    0x20000010, 0x20404010, 0x00404000, 0x20400000,
    0x00404010, 0x20404000, 0x00000000, 0x20400010,
    0x00000010, 0x00004000, 0x20400000, 0x00404010,
    0x00004000, 0x00400010, 0x20004010, 0x00000000,
    0x20404000, 0x20000000, 0x00400010, 0x20004010
};

static const unsigned long SB7[64] =
{
    0x00200000, 0x04200002, 0x04000802, 0x00000000,
    0x00000800, 0x04000802, 0x00200802, 0x04200800,
    0x04200802, 0x00200000, 0x00000000, 0x04000002,
    0x00000002, 0x04000000, 0x04200002, 0x00000802,
    0x04000800, 0x00200802, 0x00200002, 0x04000800,
    0x04000002, 0x04200000, 0x04200800, 0x00200002,
    0x04200000, 0x00000800, 0x00000802, 0x04200802,
    0x00200800, 0x00000002, 0x04000000, 0x00200800,
    0x04000000, 0x00200800, 0x00200000, 0x04000802,
    0x04000802, 0x04200002, 0x04200002, 0x00000002,
    0x00200002, 0x04000000, 0x04000800, 0x00200000,
    0x04200800, 0x00000802, 0x00200802, 0x04200800,
    0x00000802, 0x04000002, 0x04200802, 0x04200000,
    0x00200800, 0x00000000, 0x00000002, 0x04200802,
    0x00000000, 0x00200802, 0x04200000, 0x00000800,
    0x04000002, 0x04000800, 0x00000800, 0x00200002
};

static const unsigned long SB8[64] =
{
    0x10001040, 0x00001000, 0x00040000, 0x10041040,
    0x10000000, 0x10001040, 0x00000040, 0x10000000,
    0x00040040, 0x10040000, 0x10041040, 0x00041000,
    0x10041000, 0x00041040, 0x00001000, 0x00000040,
    0x10040000, 0x10000040, 0x10001000, 0x00001040,
    0x00041000, 0x00040040, 0x10040040, 0x10041000,
    0x00001040, 0x00000000, 0x00000000, 0x10040040,
    0x10000040, 0x10001000, 0x00041040, 0x00040000,
    0x00041040, 0x00040000, 0x10041000, 0x00001000,
    0x00000040, 0x10040040, 0x00001000, 0x00041040,
    0x10001000, 0x00000040, 0x10000040, 0x10040000,
    0x10040040, 0x10000000, 0x00040000, 0x10001040,
    0x00000000, 0x10041040, 0x00040040, 0x10000040,
    0x10040000, 0x10001000, 0x10001040, 0x00000000,
    0x10041040, 0x00041000, 0x00041000, 0x00001040,
    0x00001040, 0x00040040, 0x10000000, 0x10041000
};

#ifndef GET_ULONG_BE
#define GET_ULONG_BE(n,b,i)                             \
{                                                       \
    (n) = ( (unsigned long) (b)[(i)    ] << 24 )        \
        | ( (unsigned long) (b)[(i) + 1] << 16 )        \
        | ( (unsigned long) (b)[(i) + 2] <<  8 )        \
        | ( (unsigned long) (b)[(i) + 3]       );       \
}
```

```
#endif

#ifndef PUT_ULONG_BE
#define PUT_ULONG_BE(n,b,i)                                     \
{                                                               \
    (b)[(i)    ] = (unsigned char) ( (n) >> 24 );       \
    (b)[(i) + 1] = (unsigned char) ( (n) >> 16 );       \
    (b)[(i) + 2] = (unsigned char) ( (n) >>  8 );       \
    (b)[(i) + 3] = (unsigned char) ( (n)       );       \
}
#endif

/*
 * Initial Permutation macro
 */
#define DES_IP(X,Y)                                             \
{                                                               \
    T = ((X >>  4) ^ Y) & 0x0F0F0F0F; Y ^= T; X ^= (T <<  4);   \
    T = ((X >> 16) ^ Y) & 0x0000FFFF; Y ^= T; X ^= (T << 16);   \
    T = ((Y >>  2) ^ X) & 0x33333333; X ^= T; Y ^= (T <<  2);   \
    T = ((Y >>  8) ^ X) & 0x00FF00FF; X ^= T; Y ^= (T <<  8);   \
    Y = ((Y << 1) | (Y >> 31)) & 0xFFFFFFFF;                    \
    T = (X ^ Y) & 0xAAAAAAAA; Y ^= T; X ^= T;                   \
    X = ((X << 1) | (X >> 31)) & 0xFFFFFFFF;                    \
}

/*
 * Final Permutation macro
 */
#define DES_FP(X,Y)                                             \
{                                                               \
    X = ((X << 31) | (X >> 1)) & 0xFFFFFFFF;                    \
    T = (X ^ Y) & 0xAAAAAAAA; X ^= T; Y ^= T;                   \
    Y = ((Y << 31) | (Y >> 1)) & 0xFFFFFFFF;                    \
    T = ((Y >>  8) ^ X) & 0x00FF00FF; X ^= T; Y ^= (T <<  8);   \
    T = ((Y >>  2) ^ X) & 0x33333333; X ^= T; Y ^= (T <<  2);   \
    T = ((X >> 16) ^ Y) & 0x0000FFFF; Y ^= T; X ^= (T << 16);   \
    T = ((X >>  4) ^ Y) & 0x0F0F0F0F; Y ^= T; X ^= (T <<  4);   \
}

/*
 * DES round macro
 */

#define DES_ROUND(X,Y,i)                                  \
{                                                         \
    T = SK[i] ^ X;                                        \
    Y ^= SB8[ (T      ) & 0x3F ] ^                        \
         SB6[ (T >>  8) & 0x3F ] ^                        \
         SB4[ (T >> 16) & 0x3F ] ^                        \
         SB2[ (T >> 24) & 0x3F ];                         \
                                                          \
    T = SK[i + 1] ^ ((X << 28) | (X >> 4));          \
    Y ^= SB7[ (T      ) & 0x3F ] ^                        \
         SB5[ (T >>  8) & 0x3F ] ^                        \
         SB3[ (T >> 16) & 0x3F ] ^                        \
         SB1[ (T >> 24) & 0x3F ];                         \
}

behavior des_Crypt_ECB(in unsigned long subKey[32],
```

```
        in unsigned char input[8],
        out unsigned char output[8])
{
  void main(void)
  {
    int i;
    unsigned long X, Y, T, SK[32];

    unsigned char outdisp[8];
    X=0; Y=0; T=0;
    //while(1)
    //{
    //
    //   wait input;
        SK = subKey;

        GET_ULONG_BE( X, input, 0 );
        GET_ULONG_BE( Y, input, 4 );

        DES_IP( X, Y );

        for( i = 0; i < 32; i += 4 )
        {
            DES_ROUND( Y, X, i);
            DES_ROUND( X, Y, (i + 2));
        }

        DES_FP( Y, X );

      PUT_ULONG_BE( Y, outdisp, 0 );
        PUT_ULONG_BE( X, outdisp, 4 );

      output = outdisp;

    //}
  }
};


//EOF
```

A.2.8 stimulus.sc

```
/* ------------------------------------------------------------------------
 * stimulus.sc - stimulus module for Data Encryption Standard (DES) Algorithm
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "des.sh"

//import "des_Enc_Dec";
import "des";
import "i_sender";

#define MAX_LINE 1024
```

```
/*
 * DES/3DES test vectors (source: NIST, tripledes-vectors.zip)
 */
static const unsigned char DES_keys[3][8] =
{
    { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF },
    { 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF, 0x01 },
  { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88 }
    //{ 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF, 0x01, 0x23 }
};

static const unsigned char DES_test[7][8] =
{
  { 0x4E, 0x6F, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74 },
    { 0x6A, 0x2A, 0x19, 0xF4, 0x1E, 0xCA, 0x85, 0x4B },
    { 0x03, 0xE6, 0x9F, 0x5B, 0xFA, 0x58, 0xEB, 0x42 },
    { 0xDD, 0x17, 0xE8, 0xB8, 0xB4, 0x37, 0xD2, 0x32 },
  { 0xCD, 0xD6, 0x4F, 0x2F, 0x94, 0x27, 0xC1, 0x5D },
    { 0x69, 0x96, 0xC8, 0xFA, 0x47, 0xA2, 0xAB, 0xEB },
  { 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34 }
    //{ 0x83, 0x25, 0x39, 0x76, 0x44, 0x09, 0x1A, 0x0A }
};

static const unsigned char DES_init_zero[8] =
{
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
};

behavior stimulus(i_sender des_test)
{
  unsigned char line[MAX_LINE];
  unsigned char tmp;

  unsigned char getValue(unsigned char ch)
  {
    if(ch >= 0x30 && ch <= 0x39)
      return (ch - 0x30);
    else
      return (ch - 0x61 + 0xa);
  }

  int getBuffer(unsigned char* buffer)
  {
    int i = 0;
    scanf("%s %s %s %s %s %s %s %s", line, line + 4, line + 8,
    line + 12, line + 16, line + 20, line + 24, line + 28);
    fflush(stdin);
    if(strlen((char *)line) != 32)
    {
      printf("Error input! \n");
      return 1;
    }
    else
    {
      for(i = 0; i < 32; i +=4)
      {
        if((line[i] != '0') || (line[i + 1] != 'x')
        || (line[i + 2] < 0x30 || (line[i + 2] > 0x39 && line[i + 2]< 0x61) || (line[i +
            2]> 0x66))
```

```
      || (line[i + 3] < 0x30 || (line[i + 3] > 0x39 && line[i + 3]< 0x61) || (line[i +
          3]> 0x66)))
      {
        printf("Error input! \n");
        fflush(stdin);
        return 1;
      }
    }

    for(i = 0; i < 8; i ++)
    {
      tmp = getValue(line[i * 4 + 2]);
      tmp <<= 4;
      tmp |= getValue(line[i * 4 + 3]);
      buffer[i] = tmp;
    }

    printf("input buffer: 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x
        \n",
        buffer[0], buffer[1], buffer[2], buffer[3],
        buffer[4], buffer[5], buffer[6], buffer[7]);
    return 0;
  }
  return 0;
}

void main(void)
{
  int i, j, choice;

  DES_input  des_out;
  unsigned char key[8];

  key = DES_init_zero;
  printf("*****************************************************************\n");
  printf("Welcome to DES (Data Encryption Standard) Cipher Chip Simulation\n");

  while(1)
  {
    printf("Please make a simulation choice: \n");
    printf("1. encrypt data \n");
    printf("2. decrypt data \n");
    printf("3. self-test \n");
    printf("0. exit \n");
    printf("Your choice:");

    scanf("%d", &choice);
    fflush(stdin);

    switch(choice)
    {
    case 0:   //  exit simulation
      printf("Cheers,Goodbye!  \n");
      printf("*****************************************************************\n");
      exit(0);
      break;
    case 1:   // DES encryption
      printf("Please input the encryption key: \n");
      printf("Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB--->MSB)\n");
      printf("encryption key:");
```

```
        if(getBuffer(des_out.key))
        {
          continue;
        }
        printf("Please input the encryption data: \n");
        printf("Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB—>MSB)\n");
        printf("encryption data:");
        if(getBuffer(des_out.buffer))
        {
          continue;
        }
        des_out.flag = 00b;
        des_test.send(&des_out, sizeof(des_out));
        waitfor 1000;
        break;
      case 2:   // DES decryption
        printf("Please input the decryption key: \n");
        printf("Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB—>MSB)\n");
        printf("decryption key: ");
        if(getBuffer(des_out.key))
        {
          continue;
        }
        printf("Please input the decryption data: \n");
        printf("Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB—>MSB)\n");
        printf("decryption data: ");
        if(getBuffer(des_out.buffer))
        {
          continue;
        }
        des_out.flag = 01b;
        des_test.send(&des_out, sizeof(des_out));
        waitfor 1000;
        break;
      case 3:    // DES Self-test
        for(i = 0; i < 3; i ++)
        {
          waitfor 1000;
          key = DES_keys[i];

          for(j = 0; j < 7; j ++)
          {
            des_out.buffer = DES_test[j];
            des_out.key = key;
            des_out.flag = 10b;
            des_test.send(&des_out, sizeof(des_out));
            waitfor 100;
          }
        }
        break;
      default:
        break;
      }
    }
  }
};

//EOF
```

A.2.9 monitor.sc

```
/* ------------------------------------------------------------------------
 * monitor.sc - monitor module for Data Encryption Standard (DES) Algorithm
 * ------------------------------------------------------------------------
 *
 * 07/10/28  W.Chen
 * weiweic@uci.edu
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "des.sh"

import "i_receiver";

behavior monitor(i_receiver des_output)
{
  void main(void)
  {
    int count = 0;
    DES_output des_out;

    while(1)
    {
      des_output.receive(&des_out, sizeof(des_out));

      if(des_out.flag == 00b) // DES encryption
      {
        printf("encryt test...... \n");
        printf("input plaintext:       0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x
            %02x, 0x%02x \n",
          des_out.obuffer[0], des_out.obuffer[1], des_out.obuffer[2],
          des_out.obuffer[3], des_out.obuffer[4], des_out.obuffer[5],
          des_out.obuffer[6], des_out.obuffer[7]);

        printf("output encrypted text: 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x
            %02x, 0x%02x \n",
          des_out.buffer[0], des_out.buffer[1], des_out.buffer[2],
          des_out.buffer[3], des_out.buffer[4], des_out.buffer[5],
          des_out.buffer[6], des_out.buffer[7]);
        printf("case %d passed! \n\n", count);
      }
      else if(des_out.flag == 01b)
      {
        printf("decryt test...... \n");
        printf("input encrypted text:  0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x
            %02x, 0x%02x \n",
          des_out.obuffer[0], des_out.obuffer[1], des_out.obuffer[2],
          des_out.obuffer[3], des_out.obuffer[4], des_out.obuffer[5],
          des_out.obuffer[6], des_out.obuffer[7]);

        printf("output decrypted text: 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x
            %02x, 0x%02x \n",
          des_out.buffer[0], des_out.buffer[1], des_out.buffer[2], des_out.buffer[3],
          des_out.buffer[4], des_out.buffer[5], des_out.buffer[6], des_out.buffer[7]);
        printf("case %d passed! \n\n", count);
      }
      else if(des_out.flag == 10b)
      {
        printf("encryt-decrypt test...... \n");
```

```
        printf("input plaintext:      0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02
            x, 0x%02x \n",
          des_out.obuffer[0], des_out.obuffer[1], des_out.obuffer[2],
          des_out.obuffer[3], des_out.obuffer[4], des_out.obuffer[5],
          des_out.obuffer[6], des_out.obuffer[7]);

        printf("output enc-dec text: 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02
            x, 0x%02x \n",
          des_out.buffer[0], des_out.buffer[1], des_out.buffer[2],
          des_out.buffer[3], des_out.buffer[4], des_out.buffer[5],
          des_out.buffer[6], des_out.buffer[7]);

        if(!memcmp(des_out.obuffer, des_out.buffer, 8 ))
        {
          printf("case %d passed! \n\n", count);
        }
        else
        {
          printf("case %d failed! \n\n", count);
        }
      }
      count ++;
    }
  }
};


//EOF
```

A.2.10 Makefile

```
# ----------------------------------------------------------------------
# Makefile for  Data Encryption Standard (DES) Algorithm
# ----------------------------------------------------------------------
#
# Modifications:
#
# W. Chen 07/10/28  first version
#weiweic@uci.edu


SCC = scc
DESIGN = des_test
SCCOPT  = -vvv -ww -sl
SCC2SIR = -sc2sir $(SCCOPT)

all: des_test

des_test: testbench.sc des.sir stimulus.sir monitor.sir
  $(SCC) testbench $(SCCOPT) -g -o $(DESIGN)

des_SetKey.sir: des_SetKey.sc
  $(SCC) des_SetKey $(SCC2SIR)

des_SetKey_Enc.sir: des_SetKey_Enc.sc des_SetKey.sir
  $(SCC) des_SetKey_Enc $(SCC2SIR)

des_SetKey_Dec.sir: des_SetKey_Dec.sc des_SetKey.sir
  $(SCC) des_SetKey_Dec $(SCC2SIR)
```

```
des_Crypt_ECB.sir: des_Crypt_ECB.sc
  $(SCC) des_Crypt_ECB $(SCC2SIR)

#des_Enc_Dec.sir: des_Enc_Dec.sc des_SetKey_Enc.sir des_SetKey_Dec.sir des_Crypt_ECB.sir
# $(SCC) des_Enc_Dec $(SCC2SIR)

des.sir: des.sc des_SetKey_Enc.sir des_SetKey_Dec.sir des_Crypt_ECB.sir des.sh
  $(SCC) des $(SCC2SIR)

stimulus.sir: stimulus.sc des.sh
  $(SCC) stimulus $(SCC2SIR)

monitor.sir: monitor.sc des.sh
  $(SCC) monitor $(SCC2SIR)


#test:
# @echo "For testing, please run"
# @echo "    ./des_test"
# @echo "and wait for 'xterm' windows to open."

clean:
  -rm -f *~ *.o *.cc *.h ecs
  -rm -f *.si
  -rm -f *.sir
  -rm -f *.out
  -rm -f $(DESIGN)

# EOF
```

A.3 Simulation Results

```
************************************************************
Welcome to DES (Data Encryption Standard) Cipher Chip Simulation
Please make a simulation choice:
1. encrypt data
2. decrypt data
3. self-test
0. exit
Your choice:1
Please input the encryption key:
Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB-->MSB)
encryption key:0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
input buffer:0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
Please input the encryption data:
Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB-->MSB)
encryption data:0xaa 0xbb 0xcc 0xdd 0xee 0xff 0x12 0x34
input buffer: 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
encryt test......
input plaintext:       0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
output encrypted text: 0x3d, 0x4e, 0x07, 0xf3, 0x06, 0x41, 0x87, 0x60
case 0 passed!

Please make a simulation choice:
1. encrypt data
2. decrypt data
3. self-test
0. exit
Your choice:2
Please input the decryption key:
```

```
Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB—>MSB)
decryption key: 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
input buffer: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
Please input the decryption data:
Format: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 (LSB—>MSB)
decryption data: 0x3d 0x4e 0x07 0xf3 0x06 0x41 0x87 0x60
input buffer: 0x3d, 0x4e, 0x07, 0xf3, 0x06, 0x41, 0x87, 0x60
decryt test......
input encrypted text:  0x3d, 0x4e, 0x07, 0xf3, 0x06, 0x41, 0x87, 0x60
output decrypted text: 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
case 1 passed!

Please make a simulation choice:
1. encrypt data
2. decrypt data
3. self-test
0. exit
Your choice:3
encryt-decrypt test......
input plaintext:      0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74
output enc-dec text: 0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74
case 2 passed!

encryt-decrypt test......
input plaintext:      0x6a, 0x2a, 0x19, 0xf4, 0x1e, 0xca, 0x85, 0x4b
output enc-dec text: 0x6a, 0x2a, 0x19, 0xf4, 0x1e, 0xca, 0x85, 0x4b
case 3 passed!

encryt-decrypt test......
input plaintext:      0x03, 0xe6, 0x9f, 0x5b, 0xfa, 0x58, 0xeb, 0x42
output enc-dec text: 0x03, 0xe6, 0x9f, 0x5b, 0xfa, 0x58, 0xeb, 0x42
case 4 passed!

encryt-decrypt test......
input plaintext:      0xdd, 0x17, 0xe8, 0xb8, 0xb4, 0x37, 0xd2, 0x32
output enc-dec text: 0xdd, 0x17, 0xe8, 0xb8, 0xb4, 0x37, 0xd2, 0x32
case 5 passed!

encryt-decrypt test......
input plaintext:      0xcd, 0xd6, 0x4f, 0x2f, 0x94, 0x27, 0xc1, 0x5d
output enc-dec text: 0xcd, 0xd6, 0x4f, 0x2f, 0x94, 0x27, 0xc1, 0x5d
case 6 passed!

encryt-decrypt test......
input plaintext:      0x69, 0x96, 0xc8, 0xfa, 0x47, 0xa2, 0xab, 0xeb
output enc-dec text: 0x69, 0x96, 0xc8, 0xfa, 0x47, 0xa2, 0xab, 0xeb
case 7 passed!

encryt-decrypt test......
input plaintext:      0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
output enc-dec text: 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
case 8 passed!

encryt-decrypt test......
input plaintext:      0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74
output enc-dec text: 0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74
case 9 passed!

encryt-decrypt test......
input plaintext:      0x6a, 0x2a, 0x19, 0xf4, 0x1e, 0xca, 0x85, 0x4b
```

```
output enc-dec text: 0x6a, 0x2a, 0x19, 0xf4, 0x1e, 0xca, 0x85, 0x4b
case 10 passed!

encryt-decrypt test......
input plaintext:     0x03, 0xe6, 0x9f, 0x5b, 0xfa, 0x58, 0xeb, 0x42
output enc-dec text: 0x03, 0xe6, 0x9f, 0x5b, 0xfa, 0x58, 0xeb, 0x42
case 11 passed!

encryt-decrypt test......
input plaintext:     0xdd, 0x17, 0xe8, 0xb8, 0xb4, 0x37, 0xd2, 0x32
output enc-dec text: 0xdd, 0x17, 0xe8, 0xb8, 0xb4, 0x37, 0xd2, 0x32
case 12 passed!

encryt-decrypt test......
input plaintext:     0xcd, 0xd6, 0x4f, 0x2f, 0x94, 0x27, 0xc1, 0x5d
output enc-dec text: 0xcd, 0xd6, 0x4f, 0x2f, 0x94, 0x27, 0xc1, 0x5d
case 13 passed!

encryt-decrypt test......
input plaintext:     0x69, 0x96, 0xc8, 0xfa, 0x47, 0xa2, 0xab, 0xeb
output enc-dec text: 0x69, 0x96, 0xc8, 0xfa, 0x47, 0xa2, 0xab, 0xeb
case 14 passed!

encryt-decrypt test......
input plaintext:     0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
output enc-dec text: 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
case 15 passed!

encryt-decrypt test......
input plaintext:     0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74
output enc-dec text: 0x4e, 0x6f, 0x77, 0x20, 0x69, 0x73, 0x20, 0x74
case 16 passed!

encryt-decrypt test......
input plaintext:     0x6a, 0x2a, 0x19, 0xf4, 0x1e, 0xca, 0x85, 0x4b
output enc-dec text: 0x6a, 0x2a, 0x19, 0xf4, 0x1e, 0xca, 0x85, 0x4b
case 17 passed!

encryt-decrypt test......
input plaintext:     0x03, 0xe6, 0x9f, 0x5b, 0xfa, 0x58, 0xeb, 0x42
output enc-dec text: 0x03, 0xe6, 0x9f, 0x5b, 0xfa, 0x58, 0xeb, 0x42
case 18 passed!

encryt-decrypt test......
input plaintext:     0xdd, 0x17, 0xe8, 0xb8, 0xb4, 0x37, 0xd2, 0x32
output enc-dec text: 0xdd, 0x17, 0xe8, 0xb8, 0xb4, 0x37, 0xd2, 0x32
case 19 passed!

encryt-decrypt test......
input plaintext:     0xcd, 0xd6, 0x4f, 0x2f, 0x94, 0x27, 0xc1, 0x5d
output enc-dec text: 0xcd, 0xd6, 0x4f, 0x2f, 0x94, 0x27, 0xc1, 0x5d
case 20 passed!

encryt-decrypt test......
input plaintext:     0x69, 0x96, 0xc8, 0xfa, 0x47, 0xa2, 0xab, 0xeb
output enc-dec text: 0x69, 0x96, 0xc8, 0xfa, 0x47, 0xa2, 0xab, 0xeb
case 21 passed!

encryt-decrypt test......
input plaintext:     0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
```

```
output enc-dec text: 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x12, 0x34
case 22 passed!

Please make a simulation choice:
1. encrypt data
2. decrypt data
3. self-test
0. exit
Your choice:0
Cheers,Goodbye!
************************************************************
```