Software AES

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	Implement Advanced Encryption Standard by software.
BSP Version	NUC230/240 Series BSP v3.01.002
Hardware	NuTiny–EVB–NUC240–LQFP100 V1.0

1 Function Description

1.1 Introduction

This sample code demonstrates how to implement software AES (Advanced Encryption Standard) and how AES encryption/decryption works.

1.2 Principle

AES is an algorithm that operates on a 4x4 Byte matrix and consists of four main steps. The first step is to XOR the data with the key, defined as Add round key. The second step is to convert each byte of the data into a substitutes bytes through the S-Box (substitute function), defined as Substitutes bytes. The third step is the action of shifting each row of the matrix, defined as Shift rows. The fourth step is to perform polynomial multiplication for each straight line of the matrix and a polynomial, defined as Mix columns. AES encryption and decryption, the first need to expand the key, the expansion of the action is defined as Key Expansion; then use the expanded key to perform the encryption and decryption algorithm, which is the above four steps.
The process of AES encryption is shown in Figure 1-1. Suppose the encryption process has N Rounds. First, the key is expanded. After the key is expanded, the Add Round key is executed in the first Round; then Substitutes bytes, Shift rows, Mix columns, and Add round key are sequentially executed from the second Round to the (N-1) Round. The last Round executes Substitutes bytes, Shift rows, and Add round key in sequence, which completes the AES encryption process.

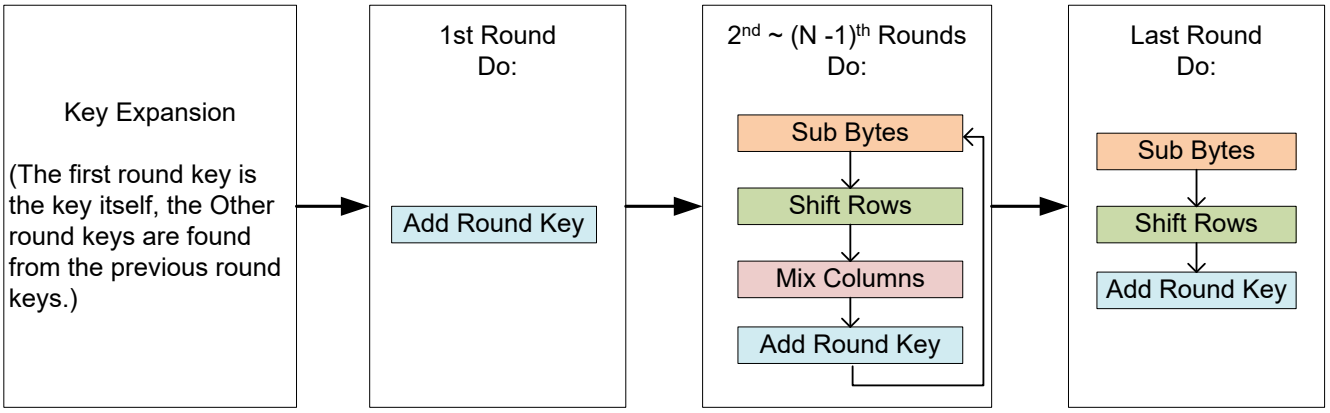

Figure 1-1 AES Encryption

The process of AES decryption is shown in Figure 1-2. Compared with the encrypted Substitutes bytes, Shift rows, and Mix columns algorithms, the algorithms of Invers substitutes bytes, Inverse shift rows, and Inverse mix columns are implemented for use in the decryption process. In the process of decryption, the hypothetical decryption process has N Rounds. First, do the expansion of the key. After the key is expanded, the Add Round key is executed in the first Round, then from the (N-1) Round to the second Round, the Inverse shift rows, the

Inverse substitutes bytes, the Add round key, and the Inverse mix columns are sequentially executed. The last round executes Inverse shift rows, Inverse substitutes bytes, and Add round key in sequence, which completes the AES decryption process.

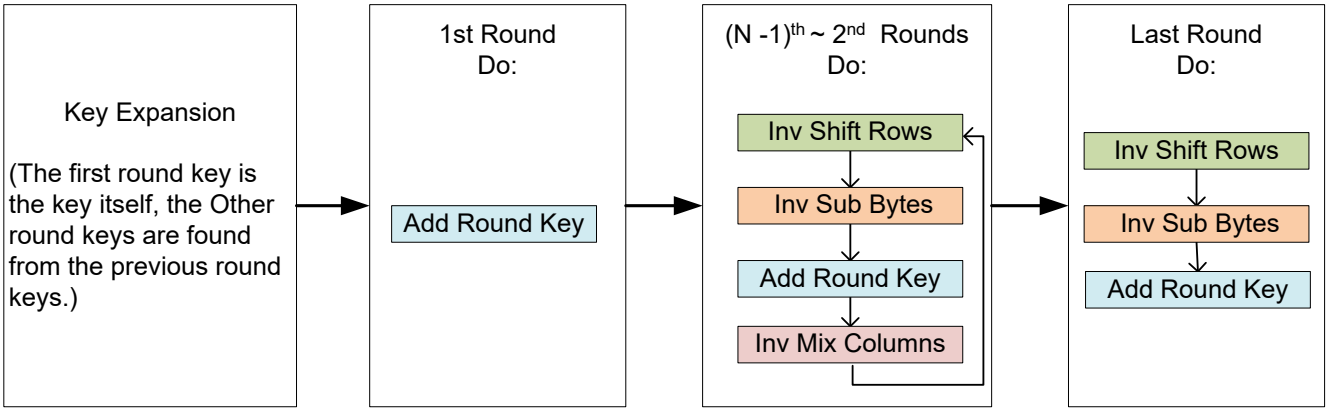

Figure 1-2 AES Decryption

1.3 Demo Result

The execution result can be output on UART, baud rate 115200, as shown in Figure 1-3.

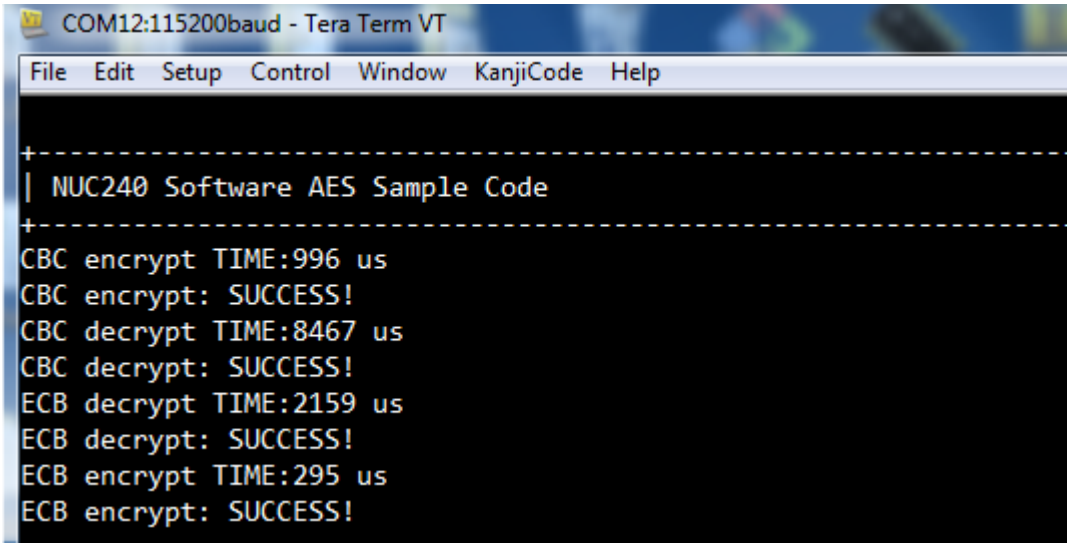

Figure 1-3 Execution Result on UART

2 Code Description

The function below is the process of key expansion：

```
/* This function produces NB(Nr+1) round keys. The round keys are used in each round to
decrypt the states. */
static void KeyExpansion(void)
{
    uint32_t i, j, k;
    uint8_t au8Tempa[4]; /* Used for the column/row operations */

    /* The first round key is the key itself. */
    for (i = 0; i < NK; ++i)
    {
        s_au8RoundKey[(i * 4) + 0] = s_pu8Key[(i * 4) + 0];
        s_au8RoundKey[(i * 4) + 1] = s_pu8Key[(i * 4) + 1];
        s_au8RoundKey[(i * 4) + 2] = s_pu8Key[(i * 4) + 2];
        s_au8RoundKey[(i * 4) + 3] = s_pu8Key[(i * 4) + 3];
    }

    /* All other round keys are found from the previous round keys. */
    for (; (i < (NB * (NR + 1))); ++i)
    {
        for (j = 0; j < 4; ++j)
        {
            au8Tempa[j] = s_au8RoundKey[(i - 1) * 4 + j];
        }
        if (i % NK == 0)
        {
            /* This function rotates the 4 bytes in a word to the left once. */
            /* [a0,a1,a2,a3] becomes [a1,a2,a3,a0]                            */

            /* Function RotWord() */
            {
                k = au8Tempa[0];
                au8Tempa[0] = au8Tempa[1];
                au8Tempa[1] = au8Tempa[2];
                au8Tempa[2] = au8Tempa[3];
                au8Tempa[3] = k;
            }
```

```
            /* SubWord() is a function that takes a four-byte input word and          */
            /* applies the S-box to each of the four bytes to produce an output word. */

            /* Function Subword()                                                     */
            {
                au8Tempa[0] = GetSBoxValue(au8Tempa[0]);
                au8Tempa[1] = GetSBoxValue(au8Tempa[1]);
                au8Tempa[2] = GetSBoxValue(au8Tempa[2]);
                au8Tempa[3] = GetSBoxValue(au8Tempa[3]);
            }

            au8Tempa[0] = au8Tempa[0] ^ s_au8Rcon[i / NK];
        }
        else if (NK > 6 && i % NK == 4)
        {
            /* Function Subword() */
            {
                au8Tempa[0] = GetSBoxValue(au8Tempa[0]);
                au8Tempa[1] = GetSBoxValue(au8Tempa[1]);
                au8Tempa[2] = GetSBoxValue(au8Tempa[2]);
                au8Tempa[3] = GetSBoxValue(au8Tempa[3]);
            }
        }
        s_au8RoundKey[i * 4 + 0] = s_au8RoundKey[(i - NK) * 4 + 0] ^ au8Tempa[0];
        s_au8RoundKey[i * 4 + 1] = s_au8RoundKey[(i - NK) * 4 + 1] ^ au8Tempa[1];
        s_au8RoundKey[i * 4 + 2] = s_au8RoundKey[(i - NK) * 4 + 2] ^ au8Tempa[2];
        s_au8RoundKey[i * 4 + 3] = s_au8RoundKey[(i - NK) * 4 + 3] ^ au8Tempa[3];
    }
}
```

The function below is the process of Add round key：

```
/* This function adds the round key to state.                 */
/* The round key is added to the state by an XOR function.    */
static void AddRoundKey(uint8_t round)
{
    uint8_t i, j;
    for (i = 0; i<4; ++i)
    {
        for (j = 0; j < 4; ++j)
        {
            (*s_pu8State)[i][j] ^= s_au8RoundKey[round * NB * 4 + i * NB + j];
```

```
        }
    }
}
```

The function below is the process of encrypting and decrypting Substitute bytes：

```
/* The SubBytes Function Substitutes the values in the */
/* state matrix with values in an S-box.               */
static void SubBytes(void)
{
    uint8_t i, j;
    for (i = 0; i < 4; ++i)
    {
        for (j = 0; j < 4; ++j)
        {
            (*s_pu8State)[j][i] = GetSBoxValue((*s_pu8State)[j][i]);
        }
    }
}
/* The SubBytes Function Substitutes the values in the   */
/* state matrix with values in an S-box.                 */
static void InvSubBytes(void)
{
    uint8_t i, j;
    for (i = 0; i<4; ++i)
    {
        for (j = 0; j<4; ++j)
        {
            (*s_pu8State)[j][i] = GetSBoxInvert((*s_pu8State)[j][i]);
        }
    }
}
```

The function below is the process of encrypting and decrypting Shift rows：

```
/* The ShiftRows() function shifts the rows in the state to the left.   */
/* Each row is shifted with different offset.                           */
/* Offset = Row number. So the first row is not shifted.                */
static void ShiftRows(void)
{
    uint8_t u8Temp;

    /* Rotate first row 1 columns to left */
    u8Temp = (*s_pu8State)[0][1];
```

```
    (*s_pu8State)[0][1] = (*s_pu8State)[1][1];
    (*s_pu8State)[1][1] = (*s_pu8State)[2][1];
    (*s_pu8State)[2][1] = (*s_pu8State)[3][1];
    (*s_pu8State)[3][1] = u8Temp;

    /* Rotate second row 2 columns to left */
    u8Temp = (*s_pu8State)[0][2];
    (*s_pu8State)[0][2] = (*s_pu8State)[2][2];
    (*s_pu8State)[2][2] = u8Temp;

    u8Temp = (*s_pu8State)[1][2];
    (*s_pu8State)[1][2] = (*s_pu8State)[3][2];
    (*s_pu8State)[3][2] = u8Temp;

    /* Rotate third row 3 columns to left */
    u8Temp = (*s_pu8State)[0][3];
    (*s_pu8State)[0][3] = (*s_pu8State)[3][3];
    (*s_pu8State)[3][3] = (*s_pu8State)[2][3];
    (*s_pu8State)[2][3] = (*s_pu8State)[1][3];
    (*s_pu8State)[1][3] = u8Temp;
}

static void InvShiftRows(void)
{
    uint8_t u8Temp;

    /* Rotate first row 1 columns to right */
    u8Temp = (*s_pu8State)[3][1];
    (*s_pu8State)[3][1] = (*s_pu8State)[2][1];
    (*s_pu8State)[2][1] = (*s_pu8State)[1][1];
    (*s_pu8State)[1][1] = (*s_pu8State)[0][1];
    (*s_pu8State)[0][1] = u8Temp;

    /*  Rotate second row 2 columns to right */
    u8Temp = (*s_pu8State)[0][2];
    (*s_pu8State)[0][2] = (*s_pu8State)[2][2];
    (*s_pu8State)[2][2] = u8Temp;

    u8Temp = (*s_pu8State)[1][2];
    (*s_pu8State)[1][2] = (*s_pu8State)[3][2];
    (*s_pu8State)[3][2] = u8Temp;
```

```
    /* Rotate third row 3 columns to right */
    u8Temp = (*s_pu8State)[0][3];
    (*s_pu8State)[0][3] = (*s_pu8State)[1][3];
    (*s_pu8State)[1][3] = (*s_pu8State)[2][3];
    (*s_pu8State)[2][3] = (*s_pu8State)[3][3];
    (*s_pu8State)[3][3] = u8Temp;
}
```

The function below is the process of encrypting and decrypting Mix columns：

```
/* MixColumns function mixes the columns of the state matrix */
static void MixColumns(void)
{
    uint8_t i;
    uint8_t u8Tmp, u8Tm, u8t;
    for (i = 0; i < 4; ++i)
    {
        u8t = (*s_pu8State)[i][0];
        u8Tmp = (*s_pu8State)[i][0] ^ (*s_pu8State)[i][1] ^ (*s_pu8State)[i][2] ^
(*s_pu8State)[i][3];
        u8Tm = (*s_pu8State)[i][0] ^ (*s_pu8State)[i][1];
        u8Tm = Xtime(u8Tm);
        (*s_pu8State)[i][0] ^= u8Tm ^ u8Tmp;
        u8Tm = (*s_pu8State)[i][1] ^ (*s_pu8State)[i][2];
        u8Tm = Xtime(u8Tm);
        (*s_pu8State)[i][1] ^= u8Tm ^ u8Tmp;
        u8Tm = (*s_pu8State)[i][2] ^ (*s_pu8State)[i][3];
        u8Tm = Xtime(u8Tm);
        (*s_pu8State)[i][2] ^= u8Tm ^ u8Tmp;
        u8Tm = (*s_pu8State)[i][3] ^ u8t;
        u8Tm = Xtime(u8Tm);
        (*s_pu8State)[i][3] ^= u8Tm ^ u8Tmp;
    }
}
/* MixColumns function mixes the columns of the state matrix.                 */
/* The method used to multiply may be difficult to understand for the inexperienced. */
/* Please use the references to gain more information.                        */
static void InvMixColumns(void)
{
    int i;
    uint8_t u8a, u8b, u8c, u8d;
    for (i = 0; i<4; ++i)
```

```
    {
        u8a = (*s_pu8State)[i][0];
        u8b = (*s_pu8State)[i][1];
        u8c = (*s_pu8State)[i][2];
        u8d = (*s_pu8State)[i][3];

        (*s_pu8State)[i][0] = Multiply(u8a, 0x0e) ^ Multiply(u8b, 0x0b) ^ Multiply(u8c,
0x0d) ^ Multiply(u8d, 0x09);
        (*s_pu8State)[i][1] = Multiply(u8a, 0x09) ^ Multiply(u8b, 0x0e) ^ Multiply(u8c,
0x0b) ^ Multiply(u8d, 0x0d);
        (*s_pu8State)[i][2] = Multiply(u8a, 0x0d) ^ Multiply(u8b, 0x09) ^ Multiply(u8c,
0x0e) ^ Multiply(u8d, 0x0b);
        (*s_pu8State)[i][3] = Multiply(u8a, 0x0b) ^ Multiply(u8b, 0x0d) ^ Multiply(u8c,
0x09) ^ Multiply(u8d, 0x0e);
    }
}
```

The function below is the AES encryption process：

```
/* Cipher is the main function that encrypts the PlainText. */
static void Cipher(void)
{
    uint8_t u8Round = 0;

    /* Add the First round key to the state before starting the rounds. */
    AddRoundKey(0);

    /* There will be Nr rounds.                          */
    /* The first Nr-1 rounds are identical.              */
    /* These Nr-1 rounds are executed in the loop below.  */
    for (u8Round = 1; u8Round < NR; ++u8Round)
    {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(u8Round);
    }

    /* The last round is given below.                        */
    /* The MixColumns function is not here in the last round. */
    SubBytes();
    ShiftRows();
    AddRoundKey(NR);
}
```

The function below is the AES decryption process：

```
static void InvCipher(void)
{
    uint8_t u8Round = 0;

    /* Add the First round key to the state before starting the rounds. */
    AddRoundKey(NR);

    /* There will be Nr rounds.                          */
    /* The first Nr-1 rounds are identical.              */
    /* These Nr-1 rounds are executed in the loop below. */
    for (u8Round = NR - 1; u8Round>0; u8Round--)
    {
        InvShiftRows();
        InvSubBytes();
        AddRoundKey(u8Round);
        InvMixColumns();
    }

    /* The last round is given below.                        */
    /* The MixColumns function is not here in the last round. */
    InvShiftRows();
    InvSubBytes();
    AddRoundKey(0);
}
```

3 Software and Hardware Environment

Software Environment

- BSP version
  - NUC230/240 Series BSP CMSIS v3.01.002
- IDE version
  - Keil uVersion 5.28

Hardware Environment

- Circuit components
  - NuTiny–EVB–NUC240–LQFP100 V1.0
- Diagram
  - Connect UART TX (PB.1) pin to PC UART RX for display the execution result of example code on PC.

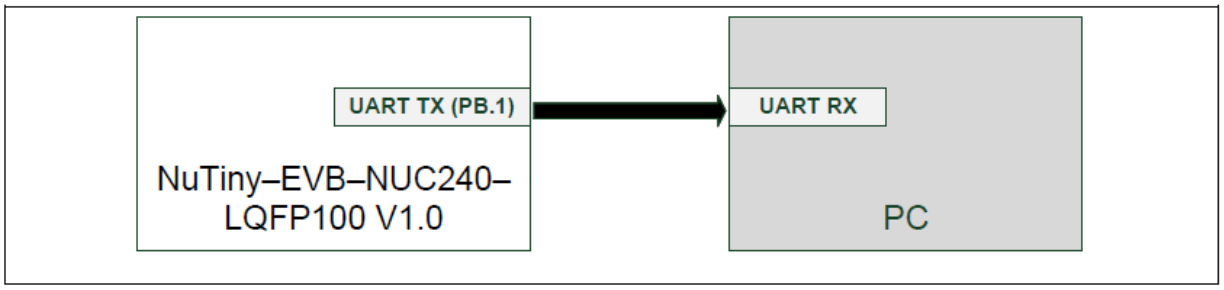

Figure 3-1 NUC240 UART TX and PC UART RX Connection

4 Directory Information

- EC_NUC240_Software_AES_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click Software_AES.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Sep. 9, 2019	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.