Escola Politécnica da USP

Departamento de Engenharia de Computação e Sistemas Digitais

PCS3722 – Arquitetura e Organização de Computadores

Profª Cíntia Borges Margi

Projeto - Fase 2

Implementacão de AES em SystemC

Bruno Susko Marcellini

Eric Rodrigues Pires

Mateus Nakajo de Mendonça

Tiago Koji Castro Shibata

Data: 09/07/2017

1. Introdução

Neste relatório serão apresentados as atividades executadas durante o desenvolvimento da parte 2 do projeto, detalhando inclusive etapas que não obtiveram êxito total em sua execução.

Os códigos desenvolvidos em SystemC e descritos nas etapas posteriores podem ser encontrados no seguinte site: https://github.com/tiagoshibata/AES-FPGA/tree/master/include/systemc

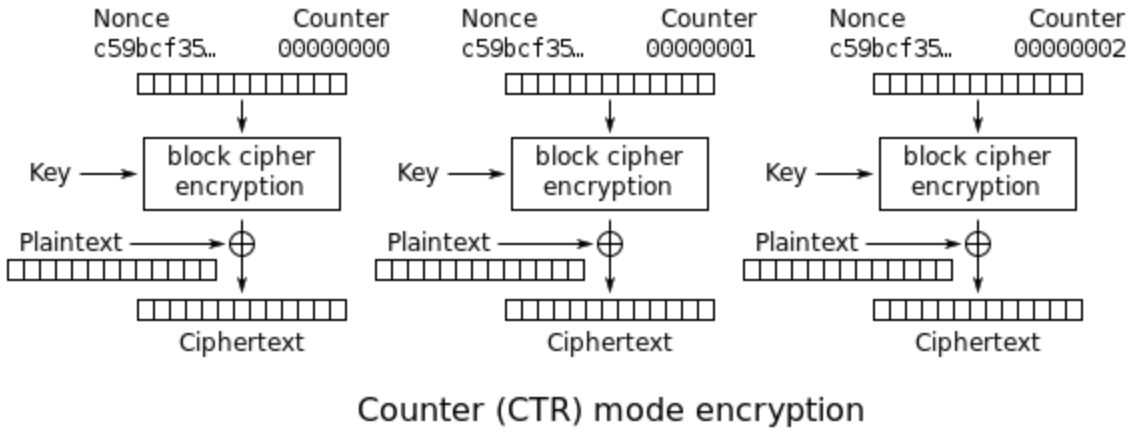
2. Atividades

O trabalho foi dividido em três etapas distintas:

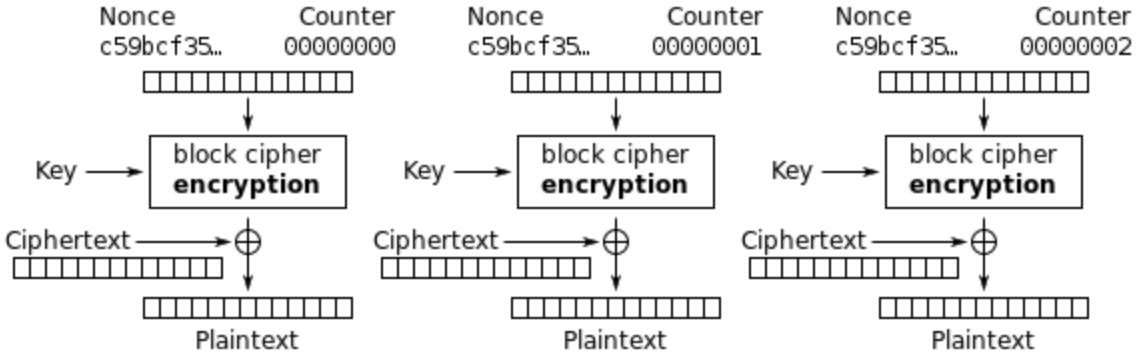
- Estudo do código em C do "golden" para entender melhor as rotinas realizadas em software.
- Aprofundamento do algoritmo para realização de código de hardware em SystemC dos componentes.
- Testes e simulação do módulo em SystemC. Em seguida, comparação do desempenho do AES em software e hardware pelo simulador.

2.1 Estudo da implementação "Golden"

Na primeira parte do relatório, selecionamos o algoritmo de Advanced Encryption Standard para implementação em hardware. Mais especificamente, escolhemos a variante do Rjindael no modo Counter, visto que poderia ser utilizado para criptografia e descriptografia sem alteração nos componentes – é utilizado um único bloco para ambas as operações, como vê-se nas figuras.



Counter (CTR) mode encryption



Counter (CTR) mode decryption

Como havíamos analisado anteriormente, o algoritmo de AES possui 4 etapas para criptografia de blocos, espalhadas por 10 rodadas (no caso de chave de 128 bits). Porém, estas etapas foram simplificadas na biblioteca mbed.TLS utilizada como "Golden". Sabendo-se que os valores podem ser pré-calculados, esta biblioteca faz uso de tabelas com os valores de referência de RoundKeys e rodadas para os diversos usos. A função principal de criptografia pode ser visualizada abaixo.

```
void mbedtls_aes_encrypt( mbedtls_aes_context *ctx,
                          const unsigned char input[16],
                          unsigned char output[16] )
{
    int i;
```

```
    uint32_t *RK, X0, X1, X2, X3, Y0, Y1, Y2, Y3;

    RK = ctx->buf; /* ctx->rk; */

    GET_UINT32_LE( X0, input,  0 ); X0 ^= *RK++;
    GET_UINT32_LE( X1, input,  4 ); X1 ^= *RK++;
    GET_UINT32_LE( X2, input,  8 ); X2 ^= *RK++;
    GET_UINT32_LE( X3, input, 12 ); X3 ^= *RK++;

    for( i = ( 10 /* ctx->nr */ >> 1 ) - 1; i > 0; i-- )
    {
        AES_FROUND( Y0, Y1, Y2, Y3, X0, X1, X2, X3 );
        AES_FROUND( X0, X1, X2, X3, Y0, Y1, Y2, Y3 );
    }

    AES_FROUND( Y0, Y1, Y2, Y3, X0, X1, X2, X3 );

    X0 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y0       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y1 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y2 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y3 >> 24 ) & 0xFF ] << 24 );

    X1 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y1       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y2 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y3 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y0 >> 24 ) & 0xFF ] << 24 );

    X2 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y2       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y3 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y0 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y1 >> 24 ) & 0xFF ] << 24 );

    X3 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y3       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y0 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y1 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y2 >> 24 ) & 0xFF ] << 24 );

    PUT_UINT32_LE( X0, output,  0 );
    PUT_UINT32_LE( X1, output,  4 );
    PUT_UINT32_LE( X2, output,  8 );
    PUT_UINT32_LE( X3, output, 12 );
}
```

As Round Keys de cada etapa são obtidas do array `RK = ctx->buf`, e são utilizadas de 4 em 4. As rodadas são realizadas em duas partes, por meio da macro AES_FROUND abaixo:

```
#define AES_FROUND(X0,X1,X2,X3,Y0,Y1,Y2,Y3)     \
{                                               \
    X0 = *RK++ ^ FT0[ ( Y0       ) & 0xFF ] ^   \
                 FT1[ ( Y1 >>  8 ) & 0xFF ] ^   \
                 FT2[ ( Y2 >> 16 ) & 0xFF ] ^   \
                 FT3[ ( Y3 >> 24 ) & 0xFF ];    \
                                                \
    X1 = *RK++ ^ FT0[ ( Y1       ) & 0xFF ] ^   \
                 FT1[ ( Y2 >>  8 ) & 0xFF ] ^   \
```

```
                FT2[ ( Y3 >> 16 ) & 0xFF ] ^    \
                FT3[ ( Y0 >> 24 ) & 0xFF ];     \
                                                \
    X2 = *RK++ ^ FT0[ ( Y2       ) & 0xFF ] ^   \
                FT1[ ( Y3 >>  8 ) & 0xFF ] ^    \
                FT2[ ( Y0 >> 16 ) & 0xFF ] ^    \
                FT3[ ( Y1 >> 24 ) & 0xFF ];     \
                                                \
    X3 = *RK++ ^ FT0[ ( Y3       ) & 0xFF ] ^   \
                FT1[ ( Y0 >>  8 ) & 0xFF ] ^    \
                FT2[ ( Y1 >> 16 ) & 0xFF ] ^    \
                FT3[ ( Y2 >> 24 ) & 0xFF ];     \
}
```

Verifica-se que as etapas foram substituídas por consultas às tabelas FT0-3 (funções de transformação equivalentes para cada byte possível) e FSb (função SubBytes), com XORs adequados às RoundKeys e demais valores.

O valor das RoundKeys, que varia segundo a chave principal, é pré-calculado e armazenado no contexto atual antes da execução do AES:

```
void mbedtls_aes_setkey_enc( mbedtls_aes_context *ctx, const unsigned char *key /*,
                    unsigned int keybits */ )
{
    unsigned int i;
    uint32_t *RK;

    /* ctx->rk = */ RK = ctx->buf;

    for( i = 0; i < ( 128 /* keybits */ >> 5 ); i++ )
    {
        GET_UINT32_LE( RK[i], key, i << 2 );
    }

    for( i = 0; i < 10; i++, RK += 4 )
        {
                RK[4]  = RK[0] ^ RCON[i] ^
                ( (uint32_t) FSb[ ( RK[3] >>  8 ) & 0xFF ]       ) ^
                ( (uint32_t) FSb[ ( RK[3] >> 16 ) & 0xFF ] <<  8 ) ^
                ( (uint32_t) FSb[ ( RK[3] >> 24 ) & 0xFF ] << 16 ) ^
                ( (uint32_t) FSb[ ( RK[3]       ) & 0xFF ] << 24 );

                RK[5]  = RK[1] ^ RK[4];
                RK[6]  = RK[2] ^ RK[5];
                RK[7]  = RK[3] ^ RK[6];
        }
}
```

Estes valores são calculados pela mesma tabela FSb e mais outra tabela, RCON (tabela round constants). Portanto, utilizando apenas tabelas de consulta (LUTs) e operadores bit-a-bit (XOR, AND, shift), e iterando-se sobre as RoundKeys previamente calculadas, é possível implementar o AES para qualquer modo. As RoundKeys são salvas na variável de contexto, para utilização ao longo de diversas funções do AES.

No caso do modo Counter escolhido, há alguns detalhes com o modo pelo qual a função `mbedtls_aes_encrypt()` é chamada. Primeiro, a entrada será o Nonce Counter, que será incrementado

após cada passo da execução. Depois, a cifra de saída sofrerá um XOR com cada caractere da mensagem da entrada (plaintext ou cyphertext) para se obter a mensagem codificada ou decodificada, respectivamente, até que todos tenham sido alterados:

```
int mbedtls_aes_crypt_ctr( mbedtls_aes_context *ctx,
                       size_t length,
                       size_t *nc_off,
                       unsigned char nonce_counter[16],
                       unsigned char stream_block[16],
                       const unsigned char *input,
                       unsigned char *output )
{
    int c, i;
    size_t n = *nc_off;

    while( length-- )
    {
        if( n == 0 ) {
            mbedtls_aes_encrypt( ctx, nonce_counter, stream_block );

            for( i = 16; i > 0; i-- )
                if( ++nonce_counter[i - 1] != 0 )
                    break;
        }
        c = *input++;
        *output++ = (unsigned char)( c ^ stream_block[n] );

        n = ( n + 1 ) & 0x0F;
    }

    *nc_off = n;

    return( 0 );
}
```

Na etapa de benchmark do último relatório, utilizamos uma função `mbedtls_aes_self_test()` para realizar os testes de verificação do código no modo CTR. Ela simplesmente realiza a definição de chave (por meio da criação das RoundKeys) e criptografa ou descriptografa utilizando a função AES com entrada nonce_counter incremental, cuja saída sofre XOR com o plaintext.

Identificamos, portanto, as seguintes entradas e saídas:

- ➔ Key: Chave criptográfica compartilhada. Responsável por gerar as RoundKeys do AES.
- ➔ Nonce Counter: Valor utilizado como entrada do AES. É incrementado para cada 128 bits da string de entrada.
- ➔ Input: Texto de 16 caracteres plaintext/ciphertext que será codificado/decodificado pelo AES, respectivamente.
- ➔ Output: Resultado ciphertext/plaintext de 16 caracteres da codificação/decodificação do AES-CTR, respectivamente.

E notamos a utilização das seguintes funções em árvore:

- ★ Geração de RoundKeys (in Key, out RoundKeys) – Gera as RKs de acordo com uma única chave de 128 bits.
- ★ AES-CTR (in RoundKeys, in NonceCounter, in Input, out Output) – Transforma o texto de entrada de acordo com as RKs e o contador.
  - ☆ AES (in Input(NonceCounter), in RoundKeys, out Output(Ciphertext) – Transforma os bytes de input do contador de acordo com as RKs.
    - ✦ AES_FROUND (in Input, in RoundKeys, out RoundOutput) – Transforma os bytes em todas as rodadas exceto a última.
    - ✦ AES_FSb (in RoundInput, in RoundKeys, out Output) – Transforma os bytes na última rodada.

Com o hardware feito, o ideal a seguir seria realizar um testbench, possivelmente imitando o comportamento do código de teste original com encryptions e decryptions consecutivos com os valores de teste definidos na memória em `aes_test_ctr_nonce_counter`, `aes_test_ctr_key`, `aes_test_ctr_pt`, e `aes_test_ctr_ct`, a ver:

```
static const unsigned char aes_test_ctr_key[3][16] =
{
    { 0xAE, 0x68, 0x52, 0xF8, 0x12, 0x10, 0x67, 0xCC,
      0x4B, 0xF7, 0xA5, 0x76, 0x55, 0x77, 0xF3, 0x9E },
    { 0x7E, 0x24, 0x06, 0x78, 0x17, 0xFA, 0xE0, 0xD7,
      0x43, 0xD6, 0xCE, 0x1F, 0x32, 0x53, 0x91, 0x63 },
    { 0x76, 0x91, 0xBE, 0x03, 0x5E, 0x50, 0x20, 0xA8,
      0xAC, 0x6E, 0x61, 0x85, 0x29, 0xF9, 0xA0, 0xDC }
};

static const unsigned char aes_test_ctr_nonce_counter[3][16] =
{
    { 0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
    { 0x00, 0x6C, 0xB6, 0xDB, 0xC0, 0x54, 0x3B, 0x59,
      0xDA, 0x48, 0xD9, 0x0B, 0x00, 0x00, 0x00, 0x01 },
    { 0x00, 0xE0, 0x01, 0x7B, 0x27, 0x77, 0x7F, 0x3F,
      0x4A, 0x17, 0x86, 0xF0, 0x00, 0x00, 0x00, 0x01 }
};

static const unsigned char aes_test_ctr_pt[3][48] =
{
    { 0x53, 0x69, 0x6E, 0x67, 0x6C, 0x65, 0x20, 0x62,
      0x6C, 0x6F, 0x63, 0x6B, 0x20, 0x6D, 0x73, 0x67 },

    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F },

    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
      0x20, 0x21, 0x22, 0x23 }
};

static const unsigned char aes_test_ctr_ct[3][48] =
{
    { 0xE4, 0x09, 0x5D, 0x4F, 0xB7, 0xA7, 0xB3, 0x79,
      0x2D, 0x61, 0x75, 0xA3, 0x26, 0x13, 0x11, 0xB8 },
```

```
    { 0x51, 0x04, 0xA1, 0x06, 0x16, 0x8A, 0x72, 0xD9,
      0x79, 0x0D, 0x41, 0xEE, 0x8E, 0xDA, 0xD3, 0x88,
      0xEB, 0x2E, 0x1E, 0xFC, 0x46, 0xDA, 0x57, 0xC8,
      0xFC, 0xE6, 0x30, 0xDF, 0x91, 0x41, 0xBE, 0x28 },
    { 0xC1, 0xCF, 0x48, 0xA8, 0x9F, 0x2F, 0xFD, 0xD9,
      0xCF, 0x46, 0x52, 0xE9, 0xEF, 0xDB, 0x72, 0xD7,
      0x45, 0x40, 0xA4, 0x2B, 0xDE, 0x6D, 0x78, 0x36,
      0xD5, 0x9A, 0x5C, 0xEA, 0xAE, 0xF3, 0x10, 0x53,
      0x25, 0xB2, 0x07, 0x2F }
};

...

/*
 * Checkup routine
 */
int mbedtls_aes_self_test( int verbose )
{
    int ret = 0, i, j, u, v;
    unsigned char key[32];
    unsigned char buf[64];
    size_t offset;
    int len;
    unsigned char nonce_counter[16];
    unsigned char stream_block[16];
    mbedtls_aes_context ctx;

    memset( key, 0, 32 );
    mbedtls_aes_init( &ctx );

    /*
     * CTR mode
     */
    for( i = 0; i < 6; i++ )
    {
        u = i >> 1;
        v = i  & 1;

        if( verbose != 0 )
            mbedtls_printf( "  AES-CTR-128 (%s): ",
                            ( v == MBEDTLS_AES_DECRYPT ) ? "dec" : "enc" );

        memcpy( nonce_counter, aes_test_ctr_nonce_counter[u], 16 );
        memcpy( key, aes_test_ctr_key[u], 16 );

        offset = 0;
        mbedtls_aes_setkey_enc( &ctx, key /*, 128 */ );

        if( v == MBEDTLS_AES_DECRYPT )
        {
            len = aes_test_ctr_len[u];
            memcpy( buf, aes_test_ctr_ct[u], len );

            mbedtls_aes_crypt_ctr( &ctx, len, &offset, nonce_counter, stream_block,
                           buf, buf );

            if( memcmp( buf, aes_test_ctr_pt[u], len ) != 0 )
            {
                if( verbose != 0 )
                    mbedtls_printf( "failed\n" );

                ret = 1;
                goto exit;
            }
        }
```

```
        else
        {
            len = aes_test_ctr_len[u];
            memcpy( buf, aes_test_ctr_pt[u], len );

            mbedtls_aes_crypt_ctr( &ctx, len, &offset, nonce_counter, stream_block,
                           buf, buf );

            if( memcmp( buf, aes_test_ctr_ct[u], len ) != 0 )
            {
                if( verbose != 0 )
                    mbedtls_printf( "failed\n" );

                ret = 1;
                goto exit;
            }
        }

        if( verbose != 0 )
            mbedtls_printf( "passed\n" );
    }

    if( verbose != 0 )
        mbedtls_printf( "\n" );

    ret = 0;

exit:
    mbedtls_aes_free( &ctx );

    return( ret );
}
```

2.2 Desenvolvimento de componentes em SystemC

Como foi comentado anteriormente, as funções AES_FROUND e AES_FSb utilizam apenas OU-exclusivo (XOR) e tabelas de consulta, podem ser implementadas com extrema facilidade em hardware. Para isso, criamos dois componentes assíncronos, respectivamente, que realizam essas funções com as tabelas adequadas de cada:

aes_fround.h

```
#include <systemc>

#define FT \
\
    V(A5,63,63,C6), V(84,7C,7C,F8), V(99,77,77,EE), V(8D,7B,7B,F6), \
    V(0D,F2,F2,FF), V(BD,6B,6B,D6), V(B1,6F,6F,DE), V(54,C5,C5,91), \
    V(50,30,30,60), V(03,01,01,02), V(A9,67,67,CE), V(7D,2B,2B,56), \
    V(19,FE,FE,E7), V(62,D7,D7,B5), V(E6,AB,AB,4D), V(9A,76,76,EC), \
    V(45,CA,CA,8F), V(9D,82,82,1F), V(40,C9,C9,89), V(87,7D,7D,FA), \
    V(15,FA,FA,EF), V(EB,59,59,B2), V(C9,47,47,8E), V(0B,F0,F0,FB), \
    V(EC,AD,AD,41), V(67,D4,D4,B3), V(FD,A2,A2,5F), V(EA,AF,AF,45), \
    V(BF,9C,9C,23), V(F7,A4,A4,53), V(96,72,72,E4), V(5B,C0,C0,9B), \
    V(C2,B7,B7,75), V(1C,FD,FD,E1), V(AE,93,93,3D), V(6A,26,26,4C), \
    V(5A,36,36,6C), V(41,3F,3F,7E), V(02,F7,F7,F5), V(4F,CC,CC,83), \
    V(5C,34,34,68), V(F4,A5,A5,51), V(34,E5,E5,D1), V(08,F1,F1,F9), \
    V(93,71,71,E2), V(73,D8,D8,AB), V(53,31,31,62), V(3F,15,15,2A), \
    V(0C,04,04,08), V(52,C7,C7,95), V(65,23,23,46), V(5E,C3,C3,9D), \
```

```
    V(28,18,18,30), V(A1,96,96,37), V(0F,05,05,0A), V(B5,9A,9A,2F), \
    V(09,07,07,0E), V(36,12,12,24), V(9B,80,80,1B), V(3D,E2,E2,DF), \
    V(26,EB,EB,CD), V(69,27,27,4E), V(CD,B2,B2,7F), V(9F,75,75,EA), \
    V(1B,09,09,12), V(9E,83,83,1D), V(74,2C,2C,58), V(2E,1A,1A,34), \
    V(2D,1B,1B,36), V(B2,6E,6E,DC), V(EE,5A,5A,B4), V(FB,A0,A0,5B), \
    V(F6,52,52,A4), V(4D,3B,3B,76), V(61,D6,D6,B7), V(CE,B3,B3,7D), \
    V(7B,29,29,52), V(3E,E3,E3,DD), V(71,2F,2F,5E), V(97,84,84,13), \
    V(F5,53,53,A6), V(68,D1,D1,B9), V(00,00,00,00), V(2C,ED,ED,C1), \
    V(60,20,20,40), V(1F,FC,FC,E3), V(C8,B1,B1,79), V(ED,5B,5B,B6), \
    V(BE,6A,6A,D4), V(46,CB,CB,8D), V(D9,BE,BE,67), V(4B,39,39,72), \
    V(DE,4A,4A,94), V(D4,4C,4C,98), V(E8,58,58,B0), V(4A,CF,CF,85), \
    V(6B,D0,D0,BB), V(2A,EF,EF,C5), V(E5,AA,AA,4F), V(16,FB,FB,ED), \
    V(C5,43,43,86), V(D7,4D,4D,9A), V(55,33,33,66), V(94,85,85,11), \
    V(CF,45,45,8A), V(10,F9,F9,E9), V(06,02,02,04), V(81,7F,7F,FE), \
    V(F0,50,50,A0), V(44,3C,3C,78), V(BA,9F,9F,25), V(E3,A8,A8,4B), \
    V(F3,51,51,A2), V(FE,A3,A3,5D), V(C0,40,40,80), V(8A,8F,8F,05), \
    V(AD,92,92,3F), V(BC,9D,9D,21), V(48,38,38,70), V(04,F5,F5,F1), \
    V(DF,BC,BC,63), V(C1,B6,B6,77), V(75,DA,DA,AF), V(63,21,21,42), \
    V(30,10,10,20), V(1A,FF,FF,E5), V(0E,F3,F3,FD), V(6D,D2,D2,BF), \
    V(4C,CD,CD,81), V(14,0C,0C,18), V(35,13,13,26), V(2F,EC,EC,C3), \
    V(E1,5F,5F,BE), V(A2,97,97,35), V(CC,44,44,88), V(39,17,17,2E), \
    V(57,C4,C4,93), V(F2,A7,A7,55), V(82,7E,7E,FC), V(47,3D,3D,7A), \
    V(AC,64,64,C8), V(E7,5D,5D,BA), V(2B,19,19,32), V(95,73,73,E6), \
    V(A0,60,60,C0), V(98,81,81,19), V(D1,4F,4F,9E), V(7F,DC,DC,A3), \
    V(66,22,22,44), V(7E,2A,2A,54), V(AB,90,90,3B), V(83,88,88,0B), \
    V(CA,46,46,8C), V(29,EE,EE,C7), V(D3,B8,B8,6B), V(3C,14,14,28), \
    V(79,DE,DE,A7), V(E2,5E,5E,BC), V(1D,0B,0B,16), V(76,DB,DB,AD), \
    V(3B,E0,E0,DB), V(56,32,32,64), V(4E,3A,3A,74), V(1E,0A,0A,14), \
    V(DB,49,49,92), V(0A,06,06,0C), V(6C,24,24,48), V(E4,5C,5C,B8), \
    V(5D,C2,C2,9F), V(6E,D3,D3,BD), V(EF,AC,AC,43), V(A6,62,62,C4), \
    V(A8,91,91,39), V(A4,95,95,31), V(37,E4,E4,D3), V(8B,79,79,F2), \
    V(32,E7,E7,D5), V(43,C8,C8,8B), V(59,37,37,6E), V(B7,6D,6D,DA), \
    V(8C,8D,8D,01), V(64,D5,D5,B1), V(D2,4E,4E,9C), V(E0,A9,A9,49), \
    V(B4,6C,6C,D8), V(FA,56,56,AC), V(07,F4,F4,F3), V(25,EA,EA,CF), \
    V(AF,65,65,CA), V(8E,7A,7A,F4), V(E9,AE,AE,47), V(18,08,08,10), \
    V(D5,BA,BA,6F), V(88,78,78,F0), V(6F,25,25,4A), V(72,2E,2E,5C), \
    V(24,1C,1C,38), V(F1,A6,A6,57), V(C7,B4,B4,73), V(51,C6,C6,97), \
    V(23,E8,E8,CB), V(7C,DD,DD,A1), V(9C,74,74,E8), V(21,1F,1F,3E), \
    V(DD,4B,4B,96), V(DC,BD,BD,61), V(86,8B,8B,0D), V(85,8A,8A,0F), \
    V(90,70,70,E0), V(42,3E,3E,7C), V(C4,B5,B5,71), V(AA,66,66,CC), \
    V(D8,48,48,90), V(05,03,03,06), V(01,F6,F6,F7), V(12,0E,0E,1C), \
    V(A3,61,61,C2), V(5F,35,35,6A), V(F9,57,57,AE), V(D0,B9,B9,69), \
    V(91,86,86,17), V(58,C1,C1,99), V(27,1D,1D,3A), V(B9,9E,9E,27), \
    V(38,E1,E1,D9), V(13,F8,F8,EB), V(B3,98,98,2B), V(33,11,11,22), \
    V(BB,69,69,D2), V(70,D9,D9,A9), V(89,8E,8E,07), V(A7,94,94,33), \
    V(B6,9B,9B,2D), V(22,1E,1E,3C), V(92,87,87,15), V(20,E9,E9,C9), \
    V(49,CE,CE,87), V(FF,55,55,AA), V(78,28,28,50), V(7A,DF,DF,A5), \
    V(8F,8C,8C,03), V(F8,A1,A1,59), V(80,89,89,09), V(17,0D,0D,1A), \
    V(DA,BF,BF,65), V(31,E6,E6,D7), V(C6,42,42,84), V(B8,68,68,D0), \
    V(C3,41,41,82), V(B0,99,99,29), V(77,2D,2D,5A), V(11,0F,0F,1E), \
    V(CB,B0,B0,7B), V(FC,54,54,A8), V(D6,BB,BB,6D), V(3A,16,16,2C)

#define V(a,b,c,d) 0x##a##b##c##d
static const uint32_t FT0[256] = { FT };
#undef V

#define V(a,b,c,d) 0x##b##c##d##a
static const uint32_t FT1[256] = { FT };
#undef V

#define V(a,b,c,d) 0x##c##d##a##b
static const uint32_t FT2[256] = { FT };
#undef V

#define V(a,b,c,d) 0x##d##a##b##c
```

```
static const uint32_t FT3[256] = { FT };
#undef V

#undef FT

SC_MODULE(AES_FROUND)
{
  sc_core::sc_in<uint32_t> y0, y1, y2, y3, rk0, rk1, rk2, rk3;
  sc_core::sc_out<uint32_t> x0, x1, x2, x3;

  void do_fround()
  {
    x0.write(rk0.read() ^ FT0[ ( y0.read()       ) & 0xFF ] ^
                          FT1[ ( y1.read() >>  8 ) & 0xFF ] ^
                          FT2[ ( y2.read() >> 16 ) & 0xFF ] ^
                          FT3[ ( y3.read() >> 24 ) & 0xFF ]);

    x1.write(rk1.read() ^ FT0[ ( y1.read()       ) & 0xFF ] ^
                          FT1[ ( y2.read() >>  8 ) & 0xFF ] ^
                          FT2[ ( y3.read() >> 16 ) & 0xFF ] ^
                          FT3[ ( y0.read() >> 24 ) & 0xFF ]);

    x2.write(rk2.read() ^ FT0[ ( y2.read()       ) & 0xFF ] ^
                          FT1[ ( y3.read() >>  8 ) & 0xFF ] ^
                          FT2[ ( y0.read() >> 16 ) & 0xFF ] ^
                          FT3[ ( y1.read() >> 24 ) & 0xFF ]);

    x3.write(rk3.read() ^ FT0[ ( y3.read()       ) & 0xFF ] ^
                          FT1[ ( y0.read() >>  8 ) & 0xFF ] ^
                          FT2[ ( y1.read() >> 16 ) & 0xFF ] ^
                          FT3[ ( y2.read() >> 24 ) & 0xFF ]);
  }

  SC_CTOR(AES_FROUND)
  {
    SC_METHOD(do_fround);
    sensitive << y0 << y1 << y2 << y3 << rk0 << rk1 << rk2 << rk3;
  }
};
```

aes_fsb.h

```
#include <systemc>

static const uint32_t FSb[256] =
{
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
    0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
    0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A,
    0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,
    0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B,
    0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
    0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,
    0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
```

```
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17,
    0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88,
    0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,
    0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9,
    0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6,
    0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
    0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94,
    0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68,
    0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
};

SC_MODULE(AES_FSb)
{
  sc_core::sc_in<uint32_t> y0, y1, y2, y3, rk0, rk1, rk2, rk3;
  sc_core::sc_out<uint32_t> x0, x1, x2, x3;

  void do_fsb()
  {
    x0.write(rk0.read() ^ ( FSb[ ( y0.read()       ) & 0xFF ]       ) ^
                          ( FSb[ ( y1.read() >>  8 ) & 0xFF ] << 8  ) ^
                          ( FSb[ ( y2.read() >> 16 ) & 0xFF ] << 16 ) ^
                          ( FSb[ ( y3.read() >> 24 ) & 0xFF ] << 24 ));

    x1.write(rk1.read() ^ ( FSb[ ( y1.read()       ) & 0xFF ]       ) ^
                          ( FSb[ ( y2.read() >>  8 ) & 0xFF ] << 8  ) ^
                          ( FSb[ ( y3.read() >> 16 ) & 0xFF ] << 16 ) ^
                          ( FSb[ ( y0.read() >> 24 ) & 0xFF ] << 24 ));

    x2.write(rk2.read() ^ ( FSb[ ( y2.read()       ) & 0xFF ]       ) ^
                          ( FSb[ ( y3.read() >>  8 ) & 0xFF ] << 8  ) ^
                          ( FSb[ ( y0.read() >> 16 ) & 0xFF ] << 16 ) ^
                          ( FSb[ ( y1.read() >> 24 ) & 0xFF ] << 24 ));

    x3.write(rk3.read() ^ ( FSb[ ( y3.read()       ) & 0xFF ]       ) ^
                          ( FSb[ ( y0.read() >>  8 ) & 0xFF ] << 8  ) ^
                          ( FSb[ ( y1.read() >> 16 ) & 0xFF ] << 16 ) ^
                          ( FSb[ ( y2.read() >> 24 ) & 0xFF ] << 24 ));
  }

  SC_CTOR(AES_FSb)
  {
    SC_METHOD(do_fsb);
    sensitive << y0 << y1 << y2 << y3 << rk0 << rk1 << rk2 << rk3;
  }
};
```

As tabelas foram copiadas do código original, e serão usadas como lookup tables (LUTs) pelas rodadas do algoritmo Rjindael. A lista de sensitividade de cada uma equivale a todas as entradas.

Outro módulo que será interessante, devido às limitações de hardware em relação a software quanto a variáveis, é a atualização do Nonce Counter. Atualmente, em `mbedtls_aes_crypt_ctr()`, ele incrementa a variável em esquema Big Endian de 1, iterando da direita para a esquerda sobre os caracteres

apropriadamente. Pode ser reiniciado com um valor inicial também. Ambas as ações são síncronas i.e. em borda de subida do clock.

aes_nonce_counter.h

```
#include <systemc>

SC_MODULE(AES_Nonce_Counter)
{
  sc_core::sc_in<bool> clock, clear, increment;
  sc_core::sc_in<unsigned char> original_nc[16];
  sc_core::sc_out<unsigned char> current_nc[16];

  void do_nc()
  {
    if (clear) {
      for (int i = 0; i < 16; i++) {
        current_nc[i] = original_nc[i];
      }
    } else if (increment) {
      for (int i = 16; i > 0; i--) {
        if (++current_nc[i-1] != 0) break;
      }
    }
  };

  SC_CTOR(AES_Nonce_Counter)
  {
    SC_METHOD(do_nc);
    sensitive << clock.pos();
  }
};
```

O cálculo das RoundKeys a partir da chave compartilhada será feito por outro componente. Porém, como requer diversas consultas a LUTs e escritas a uma memória de RKs (que funciona como RAM), será utilizada uma máquina de estados. Além disso, a leitura da RAM com as RoundKeys, que será realizada nas rodadas do AES, será feita de modo assíncrono. No código abaixo, é possível ver as funções de ASM (lógica de próximo estado e fluxo de dados Moore) e a função separada de leitura da RAM, em um arquivo de cabeçalho com definições gerais e um arquivo com o código das funções:

aes_roundkey.h

```
#include <systemc>

enum aes_rk_state {
  aes_rk_st_wait, aes_rk_st_read, aes_rk_st_generate_begin, aes_rk_st_generate_end,
aes_rk_st_end
};

SC_MODULE(AES_RoundKey)
{
  sc_core::sc_in<bool> clock, start, clear;
  sc_core::sc_in<uint32_t> rk_addr;
  sc_core::sc_vector<sc_core::sc_in<unsigned char>> key;
  sc_core::sc_out<uint32_t> rk0, rk1, rk2, rk3;
  sc_core::sc_out<bool> done;
```

```
  // State logic
  sc_core::sc_signal<aes_rk_state> sreg;
  sc_core::sc_signal<aes_rk_state> snext;

  // Internal values
  sc_core::sc_signal<uint32_t> rk[44];
  sc_core::sc_signal<uint32_t> round;

  void get_next_state();
  void set_state();
  void read_rk();

  SC_CTOR(AES_RoundKey) : key("AES_RoundKey_key", 16)
  {
    round = 0;

    SC_METHOD(get_next_state);
      sensitive << sreg << start << clear << round;
       SC_METHOD(set_state);
      sensitive << clock.pos();
    SC_METHOD(read_rk);
      sensitive << rk_addr;
  }
};
```

aes_roundkey.cc

```
include "aes_roundkey.h"

#define GET_UINT32_LE(n,b,i)                            \
{                                                       \
    (n) = ( (uint32_t) (b)[(i)    ]       )             \
        | ( (uint32_t) (b)[(i) + 1] <<  8 )             \
        | ( (uint32_t) (b)[(i) + 2] << 16 )             \
        | ( (uint32_t) (b)[(i) + 3] << 24 );            \
}

static const uint32_t RCON[10] =
{
    0x00000001, 0x00000002, 0x00000004, 0x00000008,
    0x00000010, 0x00000020, 0x00000040, 0x00000080,
    0x0000001B, 0x00000036
};

static const uint32_t FSb[256] =
{
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
    0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
    0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A,
    0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,
    0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B,
    0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
    0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,
    0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
```

```
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17,
    0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88,
    0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,
    0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9,
    0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6,
    0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
    0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94,
    0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68,
    0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
};

void AES_RoundKey::get_next_state()
{
        if (clear) {
                snext = aes_rk_st_wait;
        } else switch (sreg) {
                case aes_rk_st_wait:
                        if (start) {
                                snext = aes_rk_st_read;
                        } else {
                                snext = aes_rk_st_wait;
                        }
                        break;
                case aes_rk_st_read:
                        snext = aes_rk_st_generate_begin;
                        break;
    case aes_rk_st_generate_begin:
      snext = aes_rk_st_generate_end;
       break;
    case aes_rk_st_generate_end:
      // set_state runs up to 9, since it takes 1 cycle for the snext
      //  update to be seen
      if (round < 9) {
        snext = aes_rk_st_generate_begin;
      } else {
                  snext = aes_rk_st_end;
      }
                break;
                case aes_rk_st_end:
                        snext = aes_rk_st_wait;
                        break;
        }
}

void AES_RoundKey::set_state()
{
        sreg = snext;

        switch (sreg) {
                case aes_rk_st_wait:
            round = 0;
                        done = 0;
                        break;
                case aes_rk_st_read:
                        round = 0;
                        GET_UINT32_LE(rk[0], key, 0);
                        GET_UINT32_LE(rk[1], key, 4);
                        GET_UINT32_LE(rk[2], key, 8);
```

```
                    GET_UINT32_LE(rk[3], key, 12);
                    break;
                case aes_rk_st_generate_begin:
                    rk[4 + (round << 2)]  = rk[0 + (round << 2)] ^ RCON[round] ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)] >>  8 ) & 0xFF ]       ) ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)] >> 16 ) & 0xFF ] <<  8 ) ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)] >> 24 ) & 0xFF ] << 16 ) ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)]       ) & 0xFF ] << 24 );
                    break;
    case aes_rk_st_generate_end:
                rk[5 + (round << 2)]  = rk[1 + (round << 2)] ^ rk[4 + (round << 2)];
                rk[6 + (round << 2)]  = rk[1 + (round << 2)] ^ rk[2 + (round << 2)] ^ rk[4 +
(round << 2)];
                rk[7 + (round << 2)]  = rk[1 + (round << 2)] ^ rk[2 + (round << 2)] ^ rk[3 +
(round << 2)] ^ rk[4 + (round << 2)];
                round = round + 1;
                break;
                case aes_rk_st_end:
                    done = 1;
                    break;
        }
}

void AES_RoundKey::read_rk()
{
    rk0 = rk[(rk_addr    ) % 44];
    rk1 = rk[(rk_addr + 1) % 44];
    rk2 = rk[(rk_addr + 2) % 44];
    rk3 = rk[(rk_addr + 3) % 44];
}
```

O código de criptografia AES também foi utilizado como máquina de estados, já que utiliza loops para os cálculos da cifra. Ele deve ser limpo com clear assíncrono antes de seu uso com start. A parte de acesso às RKs foi especificada como API, e deve ser conectada em um módulo de nível superior. As etapas FROUND e FSb utilizaram os dois componentes `aes_fround.h` e `aes_fsb.h` criados anteriormente. A unidade de controle e o fluxo de dados também tentaram seguir o padrão da implementação "Golden", visto que só são usados os módulos FRound e FSb, incrementações no endereço de RoundKeys e XOR. Além disso, para evitar possíveis timing hazards, utilizamos um ciclo de escrita nos módulos de rodada e um ciclo de leitura; assim, com período de clock adequado, podemos obter os valores sem nos preocuparmos com leituras erradas.

aes_encrypt.h

```
#include <systemc>

enum aes_rk_state {
  aes_rk_st_wait, aes_rk_st_read, aes_rk_st_generate_begin, aes_rk_st_generate_end,
aes_rk_st_end
};

SC_MODULE(AES_RoundKey)
{
  sc_core::sc_in<bool> clock, start, clear;
  sc_core::sc_in<uint32_t> rk_addr;
  sc_core::sc_vector<sc_core::sc_in<unsigned char>> key;
  sc_core::sc_out<uint32_t> rk0, rk1, rk2, rk3;
  sc_core::sc_out<bool> done;
```

```
  // State logic
  sc_core::sc_signal<aes_rk_state> sreg;
  sc_core::sc_signal<aes_rk_state> snext;

  // Internal values
  sc_core::sc_signal<uint32_t> rk[44];
  sc_core::sc_signal<uint32_t> round;

  void get_next_state();
  void set_state();
  void read_rk();

  SC_CTOR(AES_RoundKey) : key("AES_RoundKey_key", 16)
  {
    round = 0;

    SC_METHOD(get_next_state);
      sensitive << sreg << start << clear << round;
       SC_METHOD(set_state);
      sensitive << clock.pos();
    SC_METHOD(read_rk);
      sensitive << rk_addr;
  }
};
```

aes_encrypt.cc

```
#include "aes_roundkey.h"

#define GET_UINT32_LE(n,b,i)                            \
{                                                       \
    (n) = ( (uint32_t) (b)[(i)    ]       )             \
        | ( (uint32_t) (b)[(i) + 1] <<  8 )             \
        | ( (uint32_t) (b)[(i) + 2] << 16 )             \
        | ( (uint32_t) (b)[(i) + 3] << 24 );            \
}

static const uint32_t RCON[10] =
{
    0x00000001, 0x00000002, 0x00000004, 0x00000008,
    0x00000010, 0x00000020, 0x00000040, 0x00000080,
    0x0000001B, 0x00000036
};

static const uint32_t FSb[256] =
{
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
    0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
    0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A,
    0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,
    0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B,
    0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
    0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,
```

```
    0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17,
    0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88,
    0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,
    0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9,
    0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6,
    0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
    0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94,
    0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68,
    0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
};

void AES_RoundKey::get_next_state()
{
        if (clear) {
                snext = aes_rk_st_wait;
        } else switch (sreg) {
                case aes_rk_st_wait:
                        if (start) {
                                snext = aes_rk_st_read;
                        } else {
                                snext = aes_rk_st_wait;
                        }
                        break;
                case aes_rk_st_read:
                        snext = aes_rk_st_generate_begin;
                        break;
    case aes_rk_st_generate_begin:
      snext = aes_rk_st_generate_end;
       break;
    case aes_rk_st_generate_end:
      // set_state runs up to 9, since it takes 1 cycle for the snext
      //  update to be seen
      if (round < 9) {
        snext = aes_rk_st_generate_begin;
      } else {
                  snext = aes_rk_st_end;
      }
                break;
                case aes_rk_st_end:
                        snext = aes_rk_st_wait;
                        break;
        }
}

void AES_RoundKey::set_state()
{
        sreg = snext;

        switch (sreg) {
                case aes_rk_st_wait:
             round = 0;
                        done = 0;
                        break;
                case aes_rk_st_read:
                        round = 0;
                        GET_UINT32_LE(rk[0], key, 0);
                        GET_UINT32_LE(rk[1], key, 4);
```

```
                    GET_UINT32_LE(rk[2], key, 8);
                    GET_UINT32_LE(rk[3], key, 12);
                    break;
                case aes_rk_st_generate_begin:
                    rk[4 + (round << 2)]  = rk[0 + (round << 2)] ^ RCON[round] ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)] >>  8 ) & 0xFF ]       ) ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)] >> 16 ) & 0xFF ] <<  8 ) ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)] >> 24 ) & 0xFF ] << 16 ) ^
                    ( (uint32_t) FSb[ ( rk[3 + (round << 2)]       ) & 0xFF ] << 24 );
                    break;
    case aes_rk_st_generate_end:
                rk[5 + (round << 2)]  = rk[1 + (round << 2)] ^ rk[4 + (round << 2)];
                rk[6 + (round << 2)]  = rk[1 + (round << 2)] ^ rk[2 + (round << 2)] ^ rk[4 +
(round << 2)];
                rk[7 + (round << 2)]  = rk[1 + (round << 2)] ^ rk[2 + (round << 2)] ^ rk[3 +
(round << 2)] ^ rk[4 + (round << 2)];
                round = round + 1;
                break;
                case aes_rk_st_end:
                    done = 1;
                    break;
        }
}

void AES_RoundKey::read_rk()
{
    rk0 = rk[(rk_addr    ) % 44];
    rk1 = rk[(rk_addr + 1) % 44];
    rk2 = rk[(rk_addr + 2) % 44];
    rk3 = rk[(rk_addr + 3) % 44];
}
```

Por fim, para utilizar o módulo de AES no formato Counter, criamos a hierarquia top-level do projeto, `aes_ctr.h`. Ela utiliza o `aes_encrypt.h` para realizar as rodadas de criptografia e `aes_nonce_counter.h` para incrementar o contador de entrada do bloco de criptografia. Além disso, diferentemente do algoritmo original, o calculador de RoundKeys `aes_roundkey.h` também foi inserido dentro deste módulo, por 2 razões:

1. Para utilizar uma nova chave, é necessário usar o sinal clear. Senão, a chave utilizada da última vez será utilizada novamente. Isso segue o comportamento esperado do modo AES-CTR, além de que evita o recálculo das RoundKeys, que não variam entre cifrragens de entradas plaintext. Isso foi feito por lógica de estados com loop incompleto (i.e. não retorna ao estado de cálculo das RoundKeys).
2. Torna o sistema mais completo e com menos sinais I/O, necessitando apenas: das entradas de texto input, Nonce Counter inicial, chave compartilhada e dos sinais de controle para início e reset; e das saídas done e do texto cifrado output.

A ASM implementada foi mais simples, apenas aguardando os blocos individuais realizarem as suas tarefas. As conexões de RK entre AES_RoundKey e AES_Encrypt foram feitas adequadamente, assim como as conexões de entrada entre AES_Nonce_Counter e AES_Encrypt. A saída simplesmente faz o XOR esperado.

aes_ctr.h

```
include <systemc>
#include "aes_roundkey.h"
#include "aes_encrypt.h"
#include "aes_nonce_counter.h"

enum aes_ctr_state {
  aes_ctr_st_wait, aes_ctr_st_rk_start, aes_ctr_st_rk_wait, aes_ctr_st_enc_start,
aes_ctr_st_enc_wait, aes_ctr_st_nc_inc, aes_ctr_st_end_wait, aes_ctr_st_end
};

SC_MODULE(AES_CTR)
{
  sc_core::sc_in<bool> clock, start, clear;
  // Input:
  // - Encryption: plaintext
  // - Decryption: ciphertext
  // Nonce Counter will be incremented by 1 inside this module for each encryption
  sc_core::sc_vector<sc_core::sc_in<unsigned char>> input, nonce_counter, key;
  sc_core::sc_vector<sc_core::sc_out<unsigned char>> output;
  sc_core::sc_out<bool> done;

  // State logic
  sc_core::sc_signal<aes_ctr_state> sreg;
  sc_core::sc_signal<aes_ctr_state> snext;

  // Internal values
  sc_core::sc_signal<uint32_t> rk0, rk1, rk2, rk3, rk_addr;
  sc_core::sc_vector<sc_core::sc_signal<unsigned char>> cipher, curr_nc;

  // Modules
  AES_RoundKey *roundkey;
  sc_core::sc_signal<bool> start_rk, done_rk;
  AES_Encrypt *aes_encrypt;
  sc_core::sc_signal<bool> start_enc, done_enc;
  AES_Nonce_Counter *nonce_ctr;
  sc_core::sc_signal<bool> inc_nc;

  void get_next_state();
  void set_state();

  SC_CTOR(AES_CTR) : input("AES_CTR_input", 16), nonce_counter("AES_CTR_nonce_counter", 16),
      key("AES_CTR_key", 16), output("AES_CTR_output", 16),
      cipher("AES_CTR_cipher", 16), curr_nc("AES_CTR_curr_nc", 16)
  {
    roundkey = new AES_RoundKey("roundkey");
    roundkey->rk0(rk0); roundkey->rk1(rk1); roundkey->rk2(rk2); roundkey->rk3(rk3);
roundkey->rk_addr(rk_addr);
    roundkey->clock(clock); roundkey->start(start_rk); roundkey->clear(clear);
roundkey->done(done_rk);
    roundkey->key(key);

    aes_encrypt = new AES_Encrypt("encrypt");
    aes_encrypt->rk0(rk0); aes_encrypt->rk1(rk1); aes_encrypt->rk2(rk2); aes_encrypt->rk3(rk3);
aes_encrypt->rk_addr(rk_addr);
    aes_encrypt->clock(clock); aes_encrypt->start(start_enc); aes_encrypt->clear(clear);
aes_encrypt->done(done_enc);
    aes_encrypt->input(curr_nc); aes_encrypt->output(cipher);

    nonce_ctr = new AES_Nonce_Counter("nonce_ctr");
    nonce_ctr->clock(clock); nonce_ctr->clear(clear); nonce_ctr->increment(inc_nc);
    nonce_ctr->original_nc(nonce_counter); nonce_ctr->current_nc(curr_nc);

    SC_METHOD(get_next_state);
```

```
      sensitive << sreg << start << clear << done_rk << done_enc;
       SC_METHOD(set_state);
      sensitive << clock.pos();
  }
};
```

aes_ctr.cc

```
#include "aes_ctr.h"

void AES_CTR::get_next_state()
{
        if (clear) {
                snext = aes_ctr_st_wait;
        } else switch (sreg) {
                case aes_ctr_st_wait:
                        if (start) {
                                snext = aes_ctr_st_rk_start;
                        } else {
                                snext = aes_ctr_st_wait;
                        }
                        break;
                case aes_ctr_st_rk_start:
                    snext = aes_ctr_st_rk_wait;
                        break;
        case aes_ctr_st_rk_wait:
            if (done_rk) {
                snext = aes_ctr_st_enc_start;
            } else {
                    snext = aes_ctr_st_rk_wait;
            }
                break;
        case aes_ctr_st_enc_start:
                snext = aes_ctr_st_enc_wait;
                break;
        case aes_ctr_st_enc_wait:
            if (done_enc) {
                snext = aes_ctr_st_nc_inc;
            } else {
                    snext = aes_ctr_st_enc_wait;
            }
                break;
                case aes_ctr_st_nc_inc:
                        snext = aes_ctr_st_end_wait;
                        break;
                case aes_ctr_st_end_wait:
                        if (start) {
                                snext = aes_ctr_st_end_wait; // Wait until start is released
                        } else {
                                snext = aes_ctr_st_end;
                        }
                        break;
                case aes_ctr_st_end:
                        if (start) {
                                snext = aes_ctr_st_enc_start; // No need to recalculate round
keys
                        } else {
                                snext = aes_ctr_st_end;
                        }
                        break;
        }
}
```

```
void AES_CTR::set_state()
{
        if (!clock.posedge())
                return;
        sreg = snext;
        std::cout << "CTR state: " << sreg << "\n";
        switch (sreg) {
                case aes_ctr_st_wait:
            done = 0;
            inc_nc = 0;
            start_rk = 0;
            start_enc = 0;
            break;
         case aes_ctr_st_rk_start:
            start_rk = 1;
            break;
         case aes_ctr_st_rk_wait:
            start_rk = 0;
            break;
         case aes_ctr_st_enc_start:
            start_enc = 1;
            break;
         case aes_ctr_st_enc_wait:
            start_enc = 0;
            break;
         case aes_ctr_st_nc_inc:
            for (int i = 0; i < 16; i++) {
                output[i] = input[i] ^ cipher[i];
            }
            inc_nc = 1;
            break;
               case aes_ctr_st_end_wait:
                done = 1;
                break;
               case aes_ctr_st_end:
            done = 1;
            break;
        }
}
```

Com todos os módulos feitos, implementamos o testbench do sistema da seguinte forma, com os valores de teste do "golden":

main.cc

```
include <systemc>
#include <iostream>

#include "aes_ctr.h"

#define tick() do{  \
  std::cout << "Running 1 cycle\n";  \
  sc_core::sc_start(half_clock_time);  \
  clock = 1;  \
  sc_core::sc_start(half_clock_time);  \
  clock = 0;  \
}while(0)

static const unsigned char aes_test_ctr_nonce_counter[16] =
    { 0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x00,
```

```
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 };

static const unsigned char aes_test_ctr_key[16] =
    { 0xAE, 0x68, 0x52, 0xF8, 0x12, 0x10, 0x67, 0xCC,
      0x4B, 0xF7, 0xA5, 0x76, 0x55, 0x77, 0xF3, 0x9E };

static const unsigned char aes_test_ctr_ct[16] =
    { 0xE4, 0x09, 0x5D, 0x4F, 0xB7, 0xA7, 0xB3, 0x79,
      0x2D, 0x61, 0x75, 0xA3, 0x26, 0x13, 0x11, 0xB8 };

static const unsigned char aes_test_ctr_pt[16] =
    { 0x53, 0x69, 0x6E, 0x67, 0x6C, 0x65, 0x20, 0x62,
      0x6C, 0x6F, 0x63, 0x6B, 0x20, 0x6D, 0x73, 0x67 };

int sc_main(int argc, char* argv[])
{
  std::cout << "Starting simulation: " << sc_core::sc_time_stamp() << "\n";

  sc_core::sc_time half_clock_time(0.5, sc_core::sc_time_unit::SC_MS);
  AES_CTR ctr("AES_CTR");

  sc_core::sc_signal<bool> clock, start, clear;
  sc_core::sc_vector<sc_core::sc_signal<unsigned char>> input("AES_input", 16),
      nonce_counter("AES_nonce_counter", 16), key("AES_key", 16);
  sc_core::sc_vector<sc_core::sc_signal<unsigned char>> output("AES_output", 16);
  sc_core::sc_signal<bool> done;

  for (int i = 0; i < 16; i++) {
    key[i] = aes_test_ctr_key[i];
    nonce_counter[i] = aes_test_ctr_nonce_counter[i];
    input[i] = aes_test_ctr_pt[i];
  }

  ctr.clock(clock);
  ctr.start(start);
  ctr.clear(clear);
  ctr.input(input);
  ctr.nonce_counter(nonce_counter);
  ctr.key(key);
  ctr.output(output);
  ctr.done(done);

  start = 0;
  clear = 1;
  tick();
  clear = 0;
  tick();
  start = 1;
  tick();
  start = 0;
  while (!ctr.done) {
    tick();
  }

  std::cout << "Ended simulation: " << sc_core::sc_time_stamp() << "\n";
  for (int i = 0; i < 16; i++) {
    std::cout << "Position " << i << ": expected " << +aes_test_ctr_ct[i] << ", found " <<
+output[i] << "\n";
    assert(aes_test_ctr_ct[i] == output[i]);
  }

  std::cout << "Congratulations, it worked.\n";
  return 0;
}
```

Porém, devido à falta de recursos de documentação sobre a utilização de *testbenches* em SystemC e no Vivado para tais componentes, não foi possível realizar uma implementação do testbench no Vivado. Falaremos mais sobre esses contratempos nas seções a seguir.

Ao invés disso, conseguimos testar o nosso módulo instalando a biblioteca de SystemC e compilando o nosso código pelo GCC. Após erros iniciais que foram corrigidos por depuração do código, conseguimos verificar o funcionamento do hardware pela simulação com compilador.

2.3 Simulação no Vivado e comparações de desempenho

A parte de simulação e por consequência a de comparação de desempenho não foi concluída com sucesso. A seguir serão listados os procedimentos executados e problemas encontrados durante a realização desta etapa.

2.3.1 Instanciamento e configuração do Microblaze

Primeiramente foi realizado o instanciamento do Microblaze no Vivado 2017.2 conforme o tutorial fornecido pela disciplina. A placa escolhida foi a Nexys 4 DDR, pertencente a família Artix-7, que será futuramente utilizado na implementação física do projeto.

As configurações do Microblaze estão listadas a seguir:

- Memória local: 32KB
- Memória local ECC: Nenhuma
- Configuração de cache: 16KB
- Clock: 100MHz

Também foram adicionados 1 bloco AXI Uartlite para comunicação serial entre o mesmo e o co-processador que seria gerado e 1 bloco de interface de memória.

Foram realizadas todas as conexões corretamente e foram gerados o Bitstream e o hardware exportado para o SDK para a implementação de aplicações no mesmo.

2.3.2 Implementação do AES no Microblaze

Nesta etapa o objetivo era fazer o AES executar totalmente no Microblaze e ter seu desempenho medido através de simulações. No entanto diversas dificuldades foram encontradas nesta etapa.

A transferência e adaptação do código em C original para que fosse executado no Microblaze não funcionou conforme o previsto, apresentando diversos erros de compilação e outros internos do Vivado. Tratando-se de um uso específico do ambiente de desenvolvimento e com a dificuldade de encontrar mais informações para que os problemas fossem resolvidos, impossibilitou a completude desta etapa dentro do prazo proposto e, consequentemente, de todas as etapas dependentes dessa.

2.3.3 Geração do módulo do coprocessador através do Vivado HLS

Esta etapa consistia em utilizar o código gerado em SystemC para que o Vivado HLS sintetizasse o mesmo para exportação e simulações para tomadas de parâmetros de desempenho com o Microblaze. Aqui também foram encontradas inúmeras dificuldades que comprometeram a completude do projeto dentro do tempo proposto.

Uma vez criado o projeto no HLS, foi possível apenas realizar a verificação de sintaxe através do mesmo. Não foi possível uma síntese completa por erros de simulação e a falta de conhecimento do grupo para implementar um *testbench* adequado para o mesmo. Sendo assim, foi possível verificar apenas a coerência do código em SystemC e o funcionamento do código compilado, mas não uma verificação da simulação de FPGA.

3. Conclusão

Nesta etapa do projeto foi possível ter um primeiro contato com SystemC. Foi possível perceber que a linguagem tem suas vantagens em relação ao VHDL, uma vez que é mais alto nível, no entanto ainda é pouco utilizada, com suporte e quantidade de informação disponíveis ainda limitados.

Também foi possível utilizar o Vivado pela primeira vez em um projeto. Foi possível notar que trata-se de um ambiente de desenvolvimento com inúmeras funcionalidades. No entanto, os tutoriais disponíveis não foram suficientes para solucionar todos os problemas encontrados pelo grupo, uma vez que tratavam-se de problemas específicos do projeto. Acreditamos que, com um maior contato com a plataforma e em posse de informações mais diretamente relacionadas com o projeto, além de documentação adequada, será possível realizar as etapas propostas.

4. Bibliografia

- ARMbed. **AES source code**. Disponível em: https://tls.mbed.org/aes-source-code. Acesso em: 13/06/2017.
- National Institue of Standards and Technology. **Recommendation for Block Cipher Modes of Operation**. Disponível em: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf. Acesso em: 14/06/2017.
- Accelera Standards. **SystemC**. Disponível em: http://accellera.org/downloads/standards/systemc. Acesso em: 27/06/2017.
- Doulos. **SystemC Tutorial**. Disponível em: https://www.doulos.com/knowhow/systemc/tutorial/. Acesso em: 28/06/2017.
- Xilinx. **MicroBlaze Soft Processor Core**. Disponível em: https://www.xilinx.com/products/design-tools/microblaze.html. Acesso em: 03/07/2017.
- Xilinx. **Creating a Simple MicroBlaze Design in IP Integrator**. Disponível em: https://www.xilinx.com/video/hardware/creating-a-simple-microblaze-design-in-ip-integrator.html. Acesso em: 03/07/2017.

- Xilinx. **Using Vivado HLS SW Libraries in your C, C++, System-C Code.** Disponível em: https://www.xilinx.com/video/hardware/vivado-hls-sw-libraries-in-your-c-system-c-code.html. Acesso em: 03/07/2017.
- Digilent. **Simulating a Custom IP core using a Zynq processor**. Disponível em: https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-custom-ip-simulation/start. Acesso em: 03/07/2017.
- Digilent. **Nexys 4 DDR - Getting Started with Microblaze.** Disponível em: https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze/start. Acesso em: 03/07/2017.
- CHAPMAN, Ken. **Implementing a Simple PicoBlaze Design in Vivado**. Disponível em: http://www-classes.usc.edu/engr/ee-s/254/ee254l_lab_manual/PicoBlaze/Picoblaze_KCPSM6_Release9_30Sept14/PicoBlaze_Design_in_Vivado.pdf. Acesso em: 06/07/2017.
- Xilinx. **Getting Started with Vivado High-Level Synthesis**. Disponível em: https://www.xilinx.com/video/hardware/getting-started-vivado-high-level-synthesis.html. Acesso em: 06/07/2017.
- HU, Yu Hen. **SystemC and System Level Design Resources**. Disponível em: http://homepages.cae.wisc.edu/~ece734/SystemC/. Acesso em: 08/07/2017.
- Archlinux User Respository. **Systemc**. Disponível em: https://aur.archlinux.org/packages/systemc/. Acesso em: 09/07/2017.

As imagens utilizadas estão em domínio público.