Coprocessador AES (Advanced Encryption Standard)

Bruno Susko Marcellini
Eric Rodrigues Pires
Mateus Nakajo de Mendonça
Tiago Koji Castro Shibata

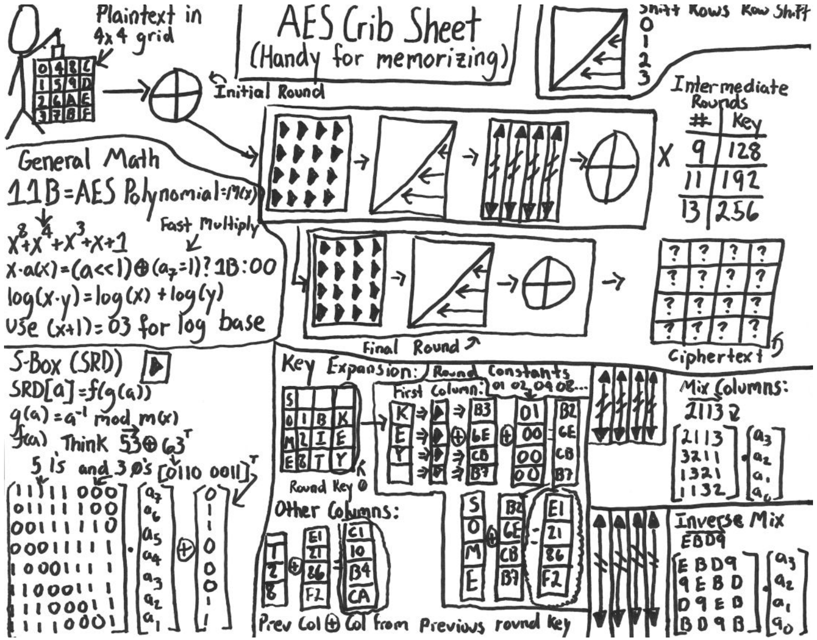
Introdução

- Algoritmo de criptografia baseado em blocos.
- Adotado como padrão pelo governo dos EUA em 2001.
- Variação do Rijndael (*rein-dal*), algoritmo vencedor de um concurso do NIST (National Institute of Standards and Technology).
- Instruções extendidas do x86 (AES-NI):

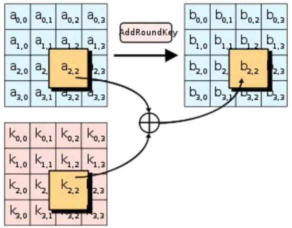
Instruction	Description
AESENC	Perform one round of an AES encryption flow
AESENCLAST	Perform the last round of an AES encryption flow
AESDEC	Perform one round of an AES decryption flow
AESDECLAST	Perform the last round of an AES decryption flow
AESKEYGENASSIST	Assist in AES round key generation
AESIMC	Assist in AES Inverse Mix Columns
PCLMULQDQ	Carryless multiply

Funcionamento

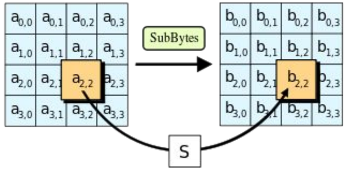
- Realiza operações determinísticas em grupos de bits como se fossem uma unidade, utilizando uma chave simétrica para transformação direta e inversa.
- Cada bloco é um arranjo bidimensional de bytes com 4x4 posições, constituindo 128 bits.
- 4 etapas de operação: AddRoundKey, SubBytes, ShiftRows, MixColumns.



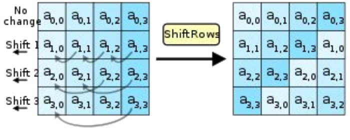
Etapas



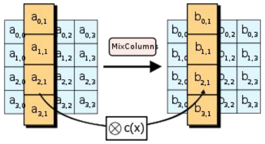
AddRoundKey



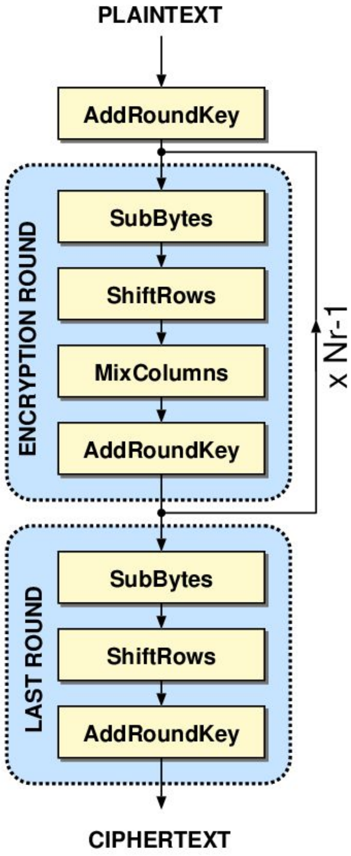
SubBytes



ShiftRows

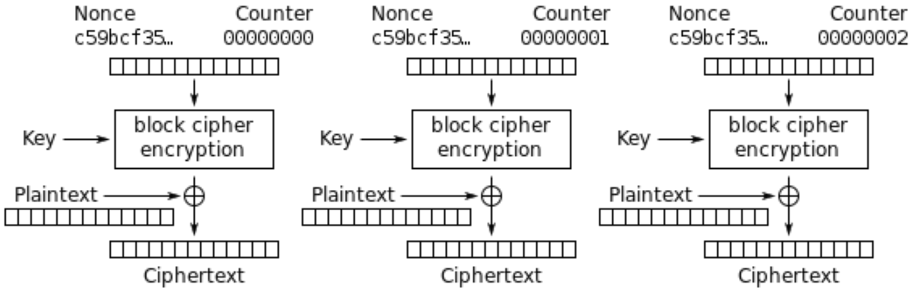


MixColumns

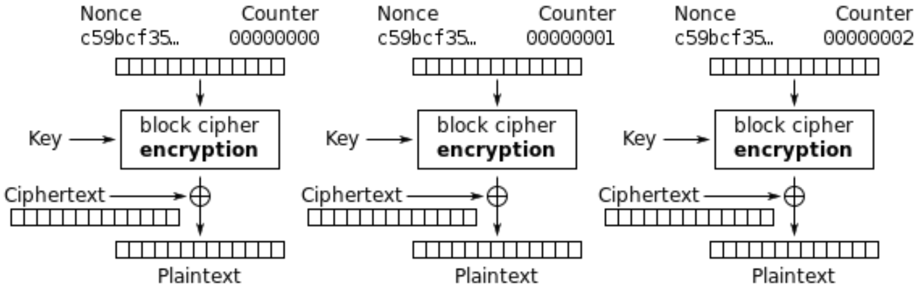


No Golden: Ao invés de cada etapa, usa tabelas pré-calculadas para substituição dos bytes.

Modo CTR

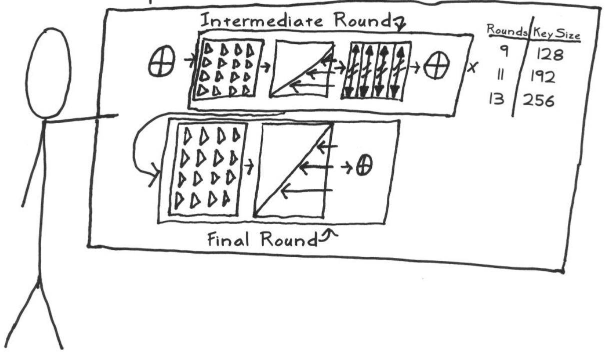


Counter (CTR) mode encryption



Counter (CTR) mode decryption

Golden – Encrypt




```c
void mbedtls_aes_encrypt( uint32_t *RK,
                          const unsigned char input[16],
                          unsigned char output[16] )
{
    uint32_t X0, X1, X2, X3, Y0, Y1, Y2, Y3;
    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 (int i = ( 10 >> 1 ) - 1; i > 0; i--)
    {
        AES_FROUND( RK, Y0, Y1, Y2, Y3, X0, X1, X2, X3 );
        AES_FROUND( RK, X0, X1, X2, X3, Y0, Y1, Y2, Y3 );
    }
    AES_FROUND( RK, Y0, Y1, Y2, Y3, X0, X1, X2, X3 );
    AES_FSb( RK, X0, X1, X2, X3, Y0, Y1, Y2, Y3 ); // Round sem MixColumns
    PUT_UINT32_LE( X0, output,  0 );
    PUT_UINT32_LE( X1, output,  4 );
    PUT_UINT32_LE( X2, output,  8 );
    PUT_UINT32_LE( X3, output, 12 );
}
```

```c
#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 ];    \
}
```

```c
#define AES_FSb(X0,X1,X2,X3,Y0,Y1,Y2,Y3)                                    \
{                                                                           \
    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 );                \
}
```

Golden – Tabelas pré-calculadas

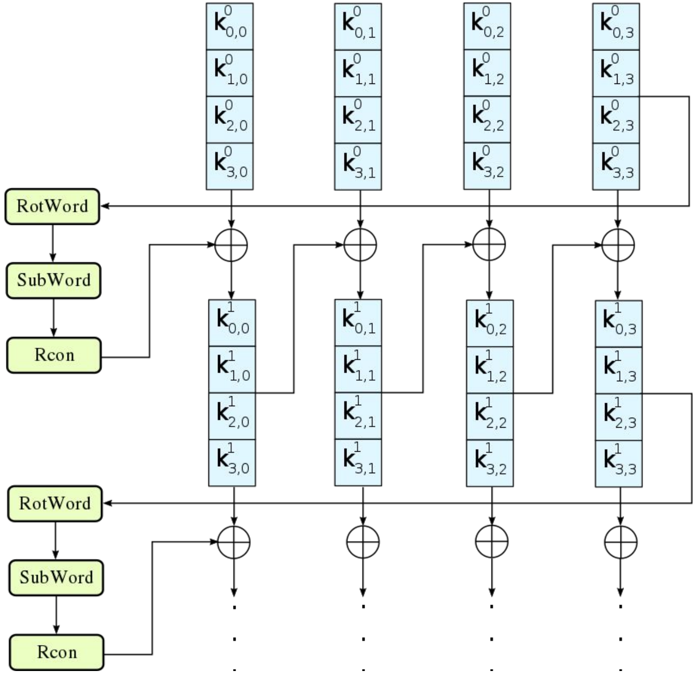
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

```
/*
* Forward tables
*/
#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), \
... total de 64 linhas ...
   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
```

```
/*
* Forward S-box
*/
static const unsigned char 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,
... total de 32 linhas ...
   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
};
```

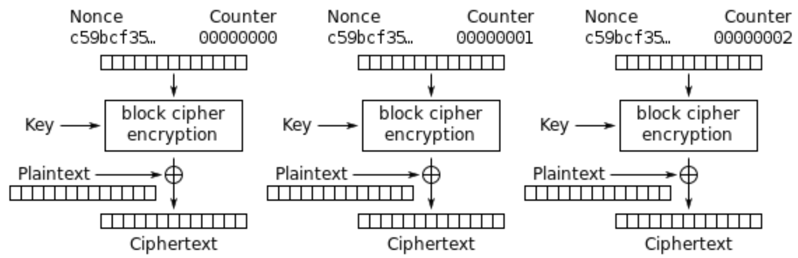
Golden – RoundKeys

```
void mbedtls_aes_setkey_enc( mbedtls_aes_context *ctx, const unsigned char *key) {
   unsigned int i;
   uint32_t *RK = ctx->buf;
   for( i = 0; i < ( 128 >> 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];
   }
}
```

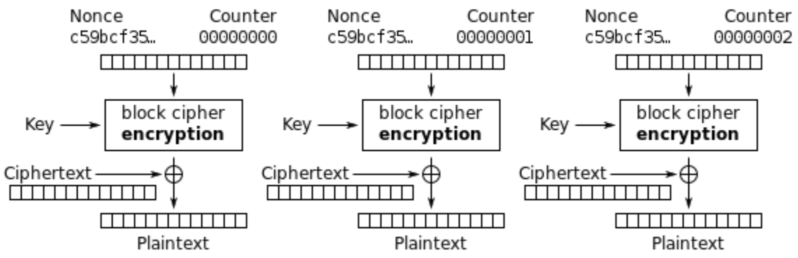


Golden – CTR

```c
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( *RK, input, output );
           mbedtls_aes_encrypt( ctx->buf, 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 );
}
```



Counter (CTR) mode encryption



Counter (CTR) mode decryption

SystemC

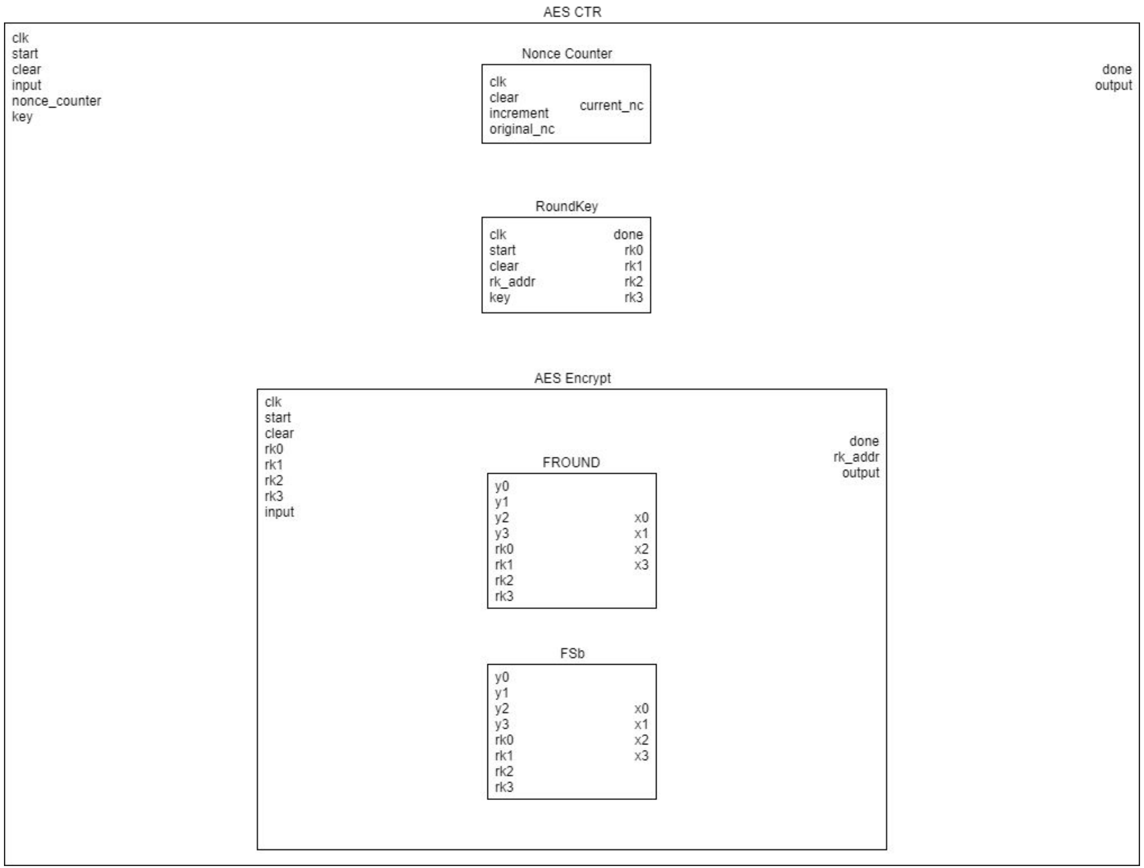
Trocar tabelas pré-calculadas (FROUND, FSb, RoundKey) por RAMs.

Calcular Nonce Counter e Round Keys internamente.

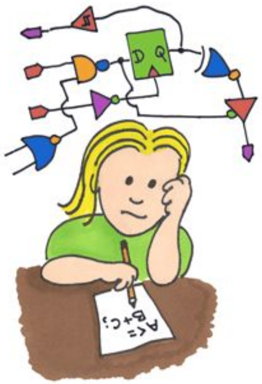
ASMs para loops.

RAMs com acesso assíncrono.

Testbench em C++ (GCC).



VHDL



Feito a partir do código SystemC ➔ Mesma estrutura de componentes.

Usa ROMs ao invés de RAMs para tabelas ➔ Serão sintetizadas como Mux (Quartus).

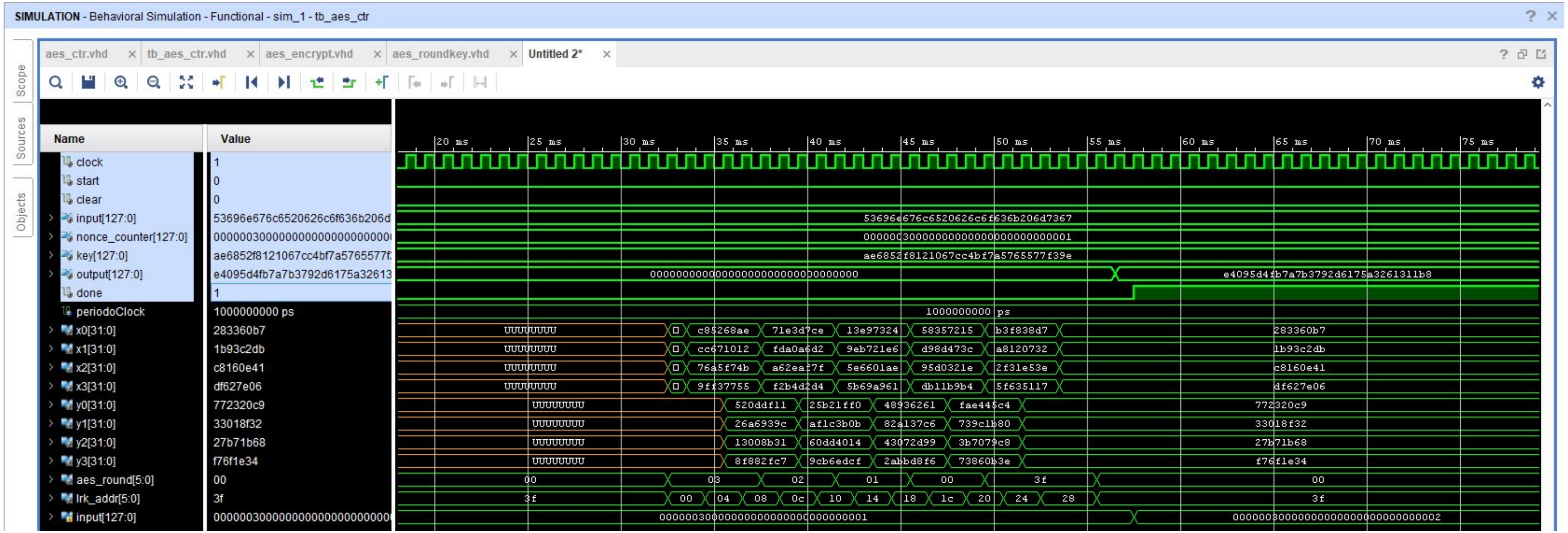
Uma única tabela FT (1024 bytes ao invés de 4 * 256 * 32 bits = 4098 bytes).

Testbench em VHDL (Vivado).

Vantagens em relação ao golden:

- Shifts + bitwise-ANDs viram divisão de barramentos.
- Shifts + bitwise-ORs viram concatenação de barramentos.
- XOR é pouco custoso em hardware (vários por ciclo de clock).
- Tabelas com múltiplos acessos simultâneos de leitura.

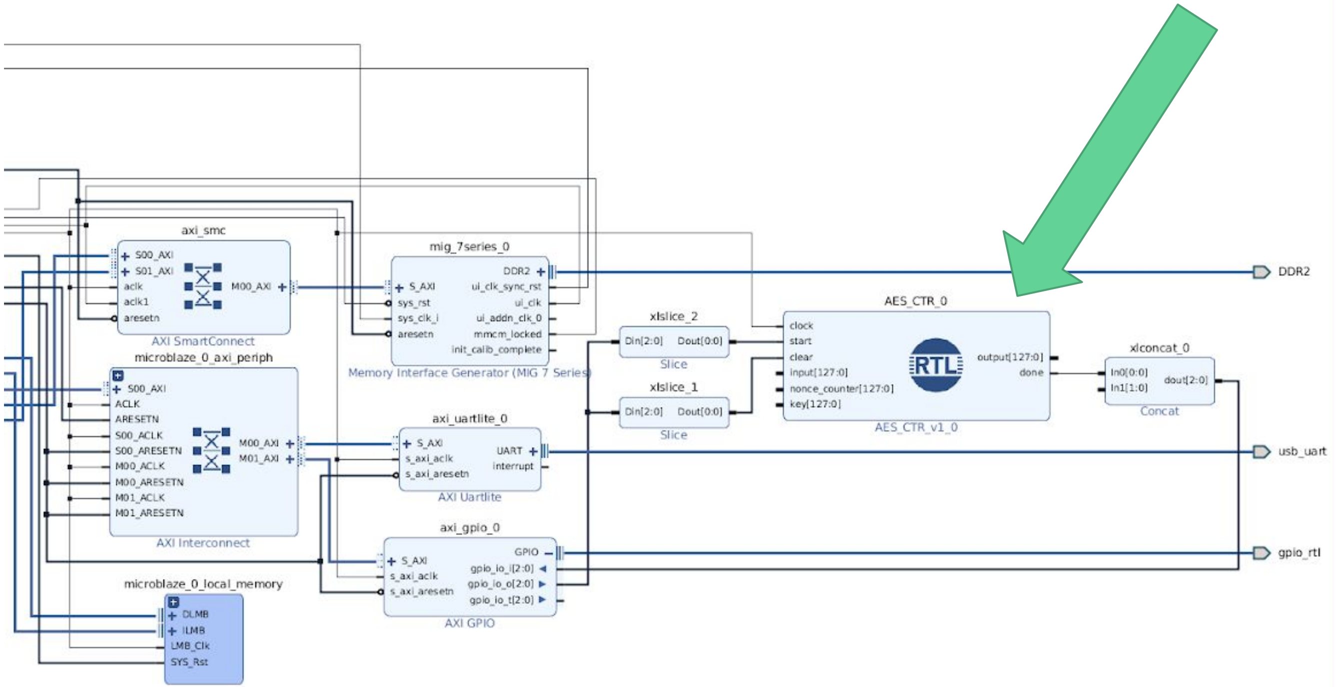
```
vhdl/
├── aes_ctr.vhd
├── aes_encrypt.vhd
├── aes_fround.vhd
├── aes_fsb.vhd
├── aes_nonce_counter.vhd
├── aes_roundkey.vhd
└── tb_aes_ctr.vhd
```



Tcl Console | Messages | Log

```
INFO: [USF-XSim-69] 'elaborate' step finished in '1' seconds
Vivado Simulator 2017.2
Time resolution is 1 ps
run 500 ms
Note: :^)
Time: 57500 us  Iteration: 1  Process: /tb_aes_ctr/tb  File: C:/Users/bruno/Dropbox/Quadri5/Arquitetura/git/AES-FPGA/include/vhdl/tb_aes_ctr.vhd
```

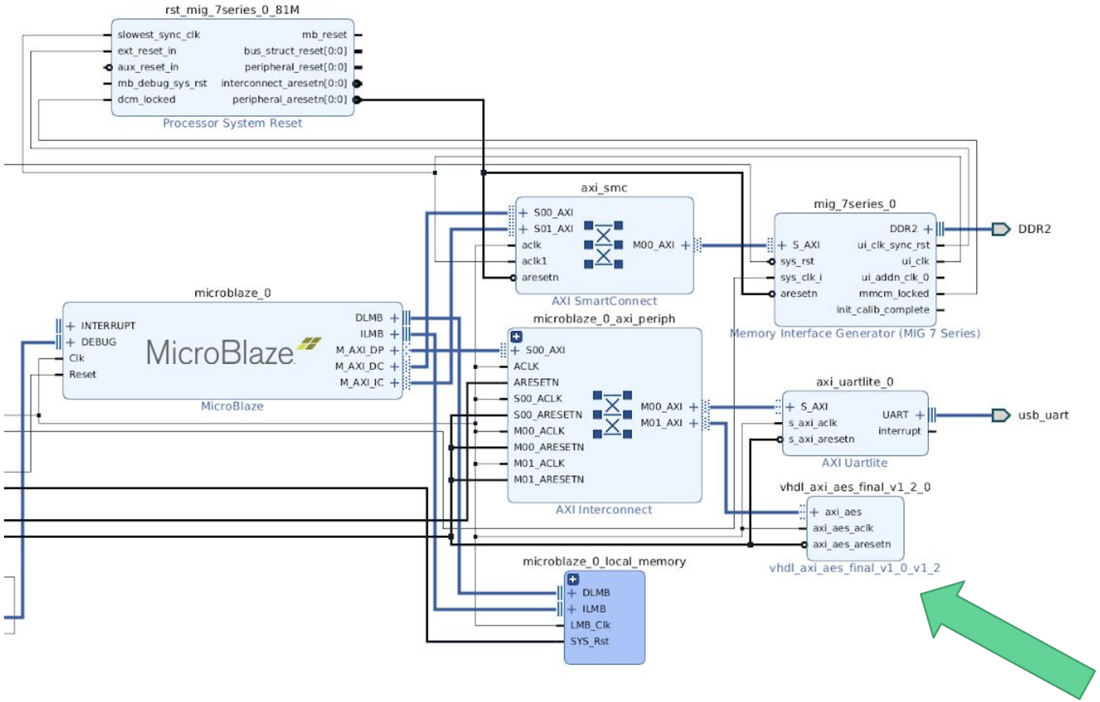
Síntese



... mas GPIO não funcionou!

Próximo passo: DMA

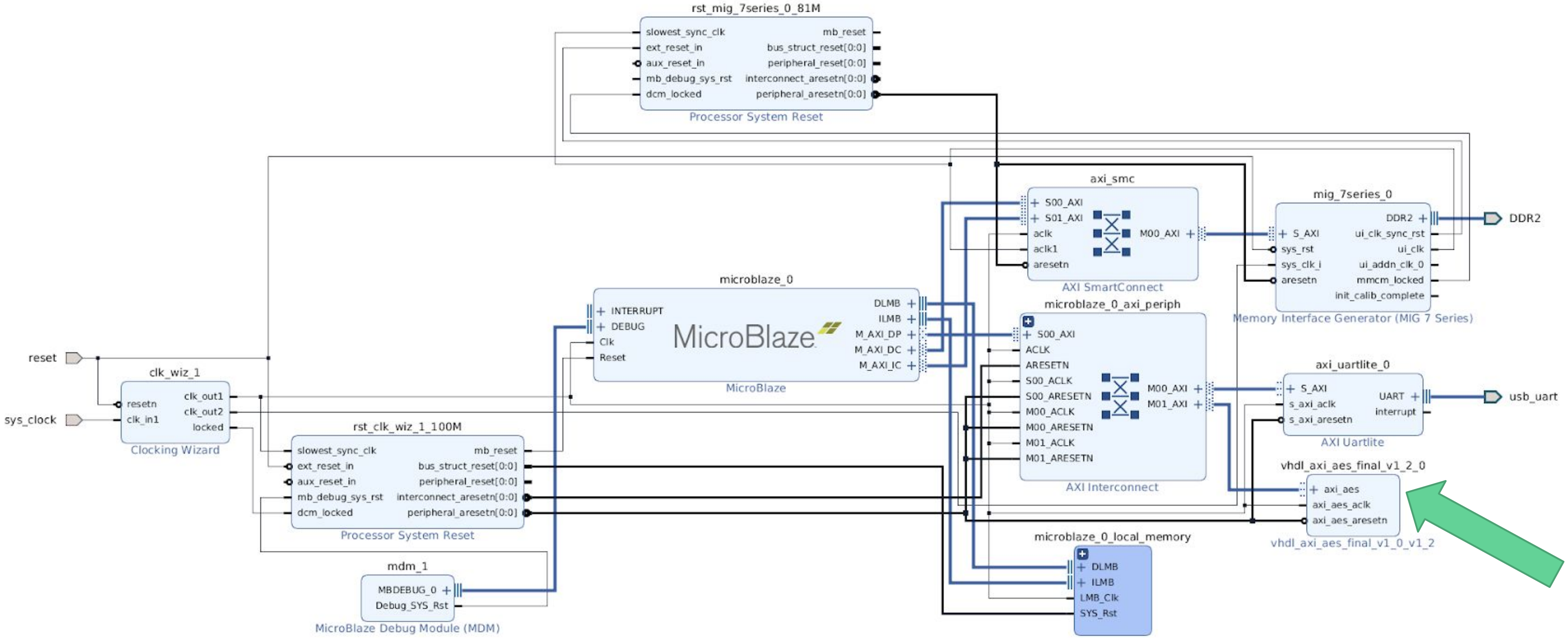
Síntese - TAKE TWO



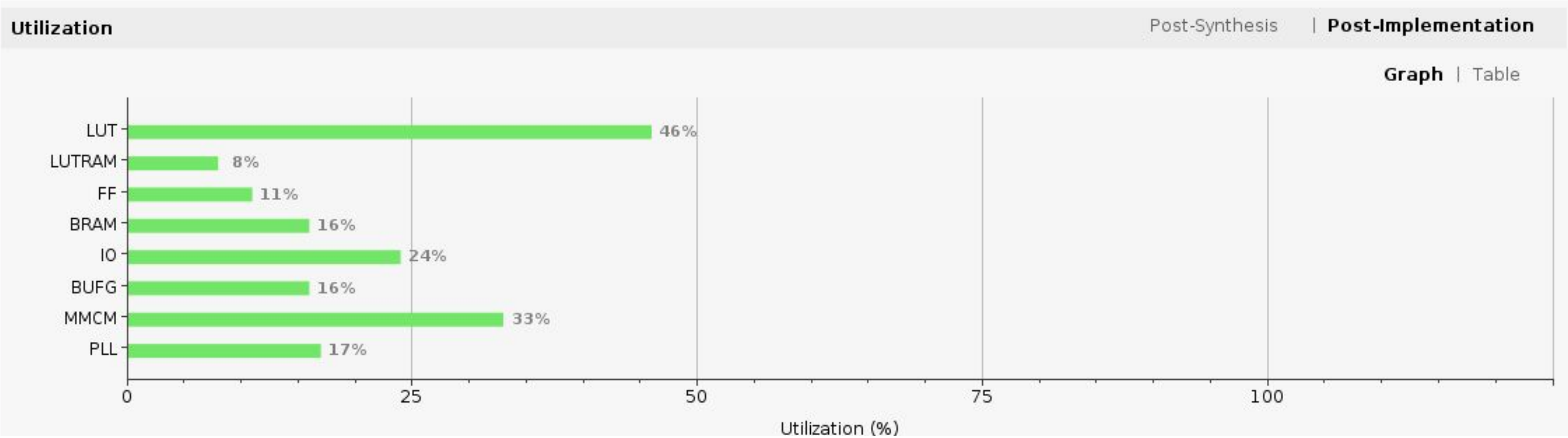
... DMA full mode não funcionou!

Próximo passo: DMA Lite

Síntese - TAKE THREE



Síntese - TAKE THREE



Métricas de desempenho

Quase 3 milhões de vezes mais rápido em hardware!

Implementação	Tempo	Tempo médio
Software	27.809 s (7 execuções)	3.97 s
VHDL	14 872.20 µs (10 000 execuções)	1.49 µs
		Relação: 2 664 430 s/s