AES ALGORITHM ENCRYPTION + DECRYPTION

HLS IMPLEMENTATION

Elpida Karapepera, Michael Litsos

AES IN A FEW WORDS

BY: RIJMEN & DAEMEN

SYMMETRICAL BLOCK CIPHER ALGORITHM

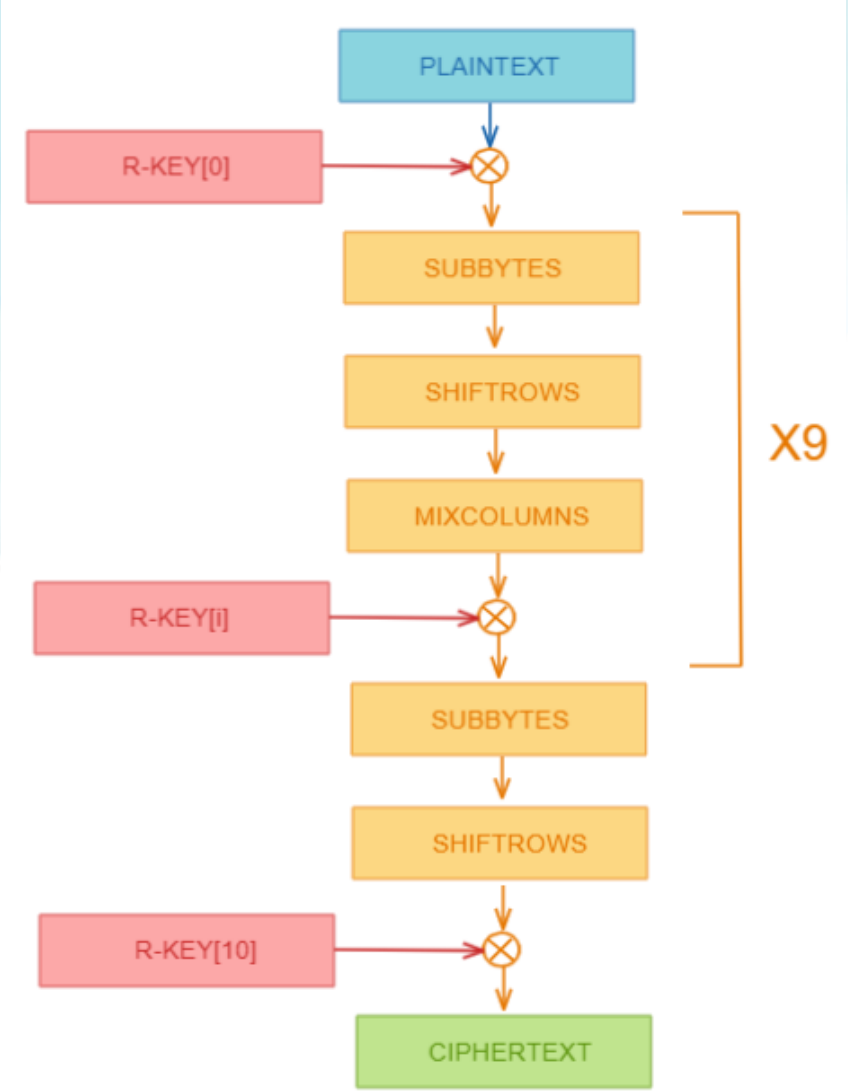
INPUT: 128-BIT PLAINTEXT

OUTPUT: 128-BIT CIPHERTEXT

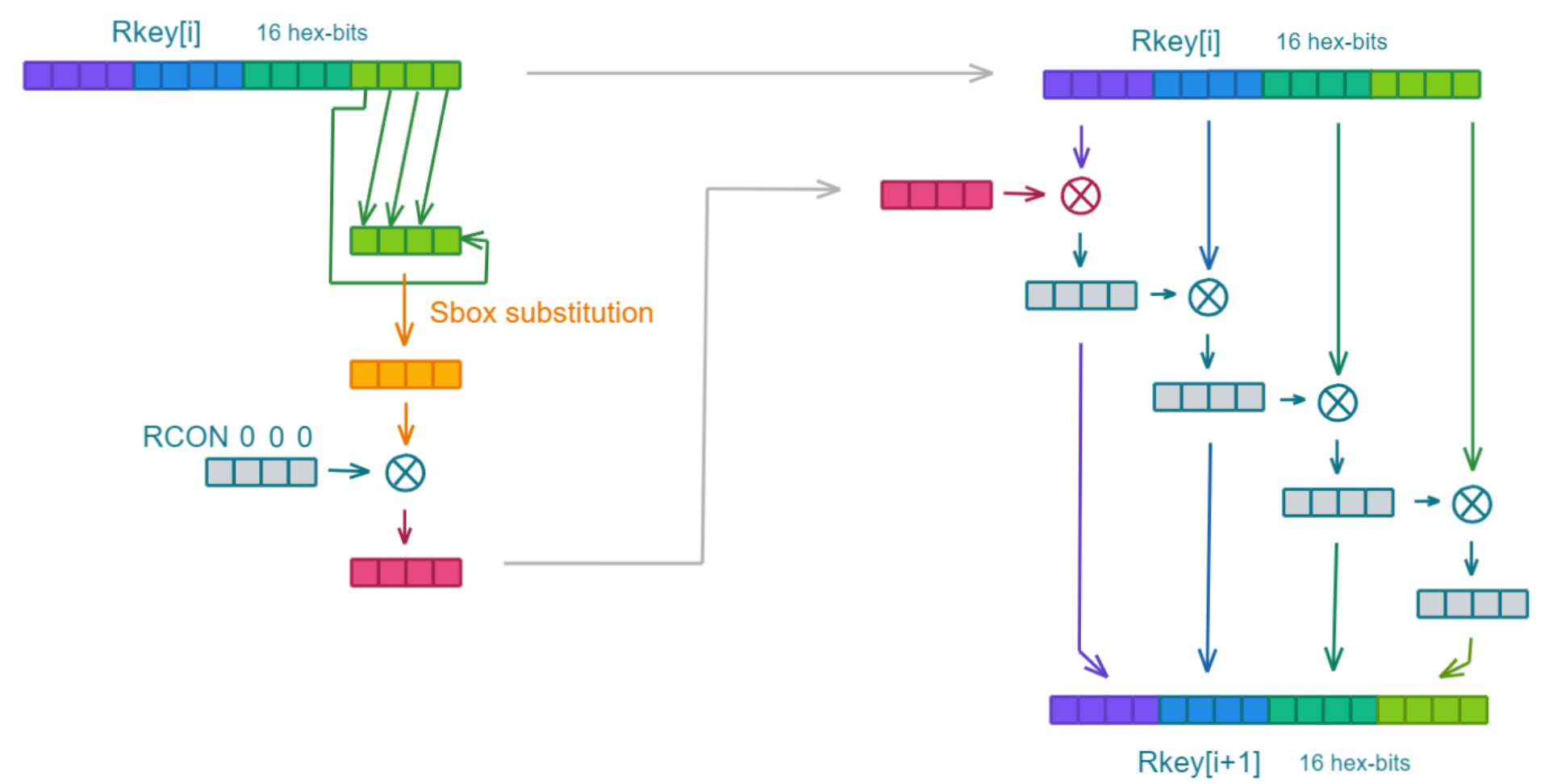
KEY: 192-BIT/256-BIT/128-BIT (THIS VERSION)

AES-128 ALGORITHM

PLAINTEXT: 128 BITS
KEY: 128 BITS
CIPHERTEXT: 128 BITS



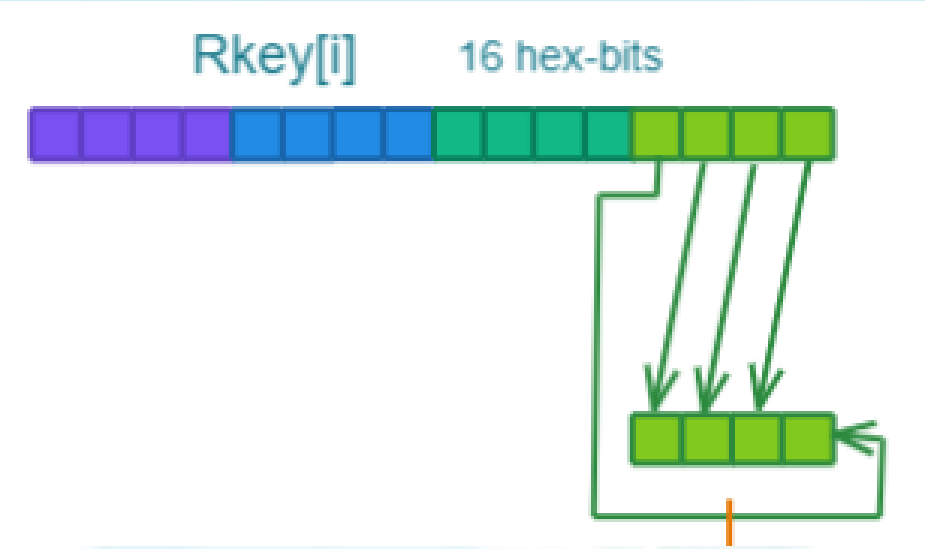
KEY GENERATION



KEY GENERATION : STEP 1

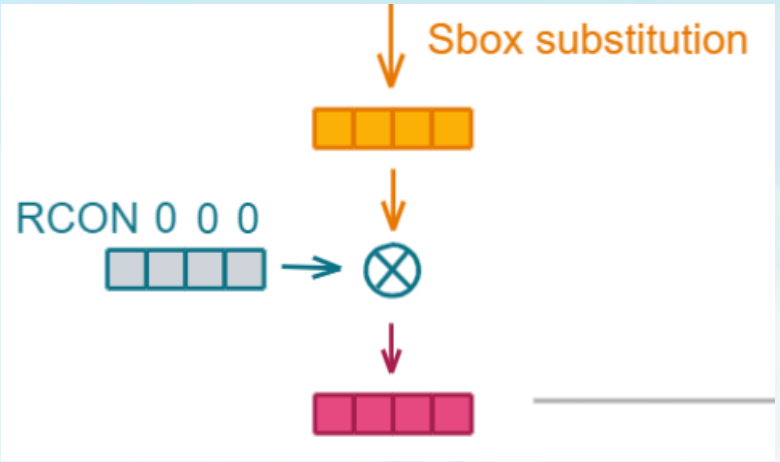

```
ac_int<32,false> W[4] = {Rkey.slc<32>(0),Rkey.slc<32>(32), Rkey.slc<32>(64), Rkey.slc<32>(96)};
```

KEY GENERATION : STEP 2



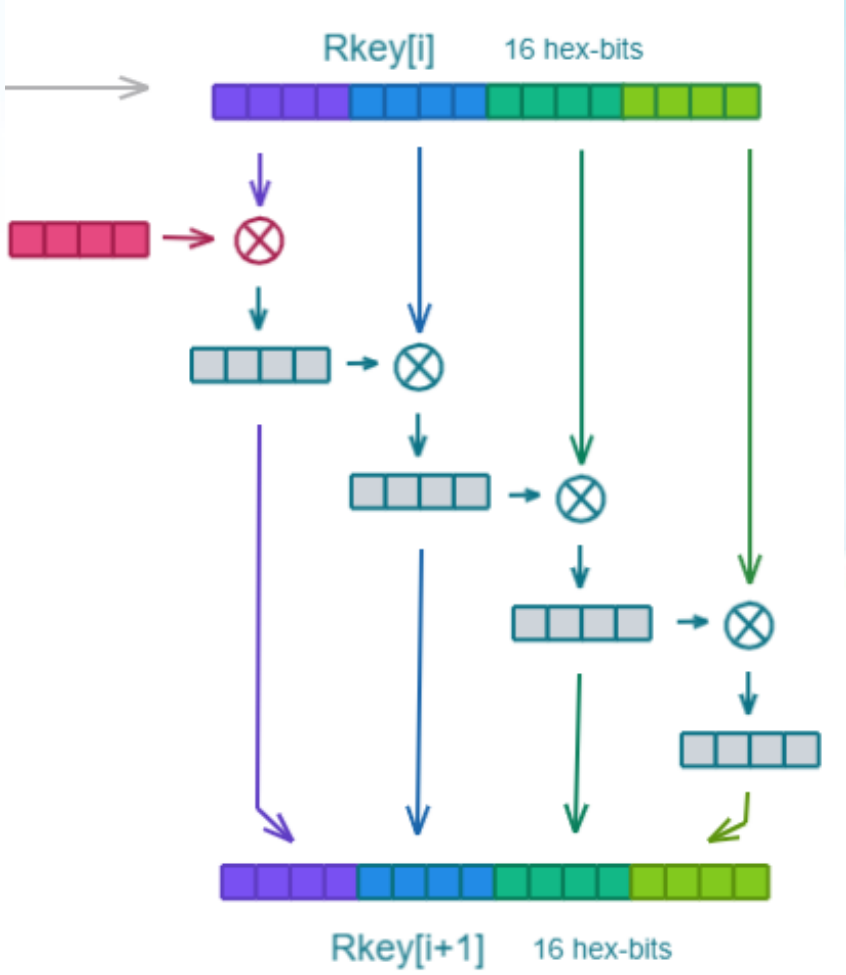

```
// Shifting
sub_word.set_slc(0, W[3].slc<8>(8));
sub_word.set_slc(8, W[3].slc<8>(16));
sub_word.set_slc(16,W[3].slc<8>(24));
sub_word.set_slc(24,W[3].slc<8>(0));
```

KEY GENERATION : STEP 3



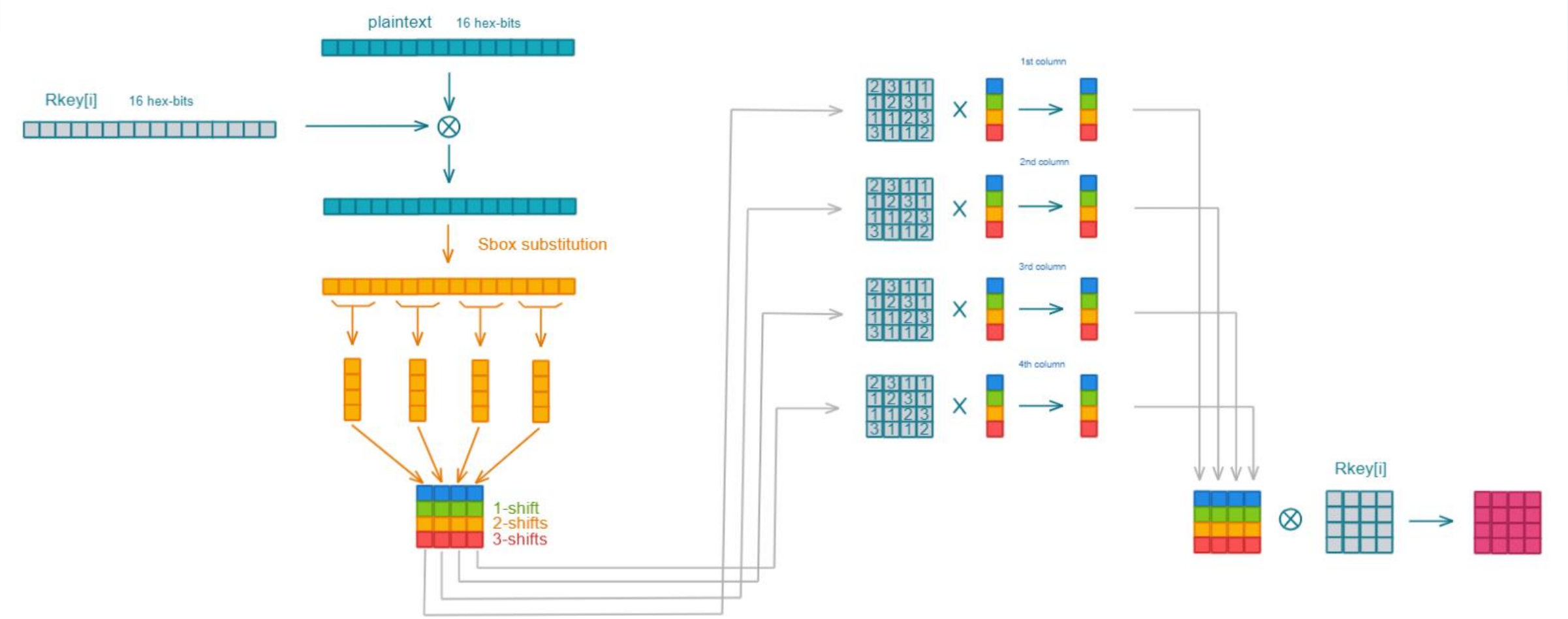

```
//Substitution
KEY_GENERATION:for(int i = 0; i < 4; i++){
    tmp = sbox[sub_word.slc<4>(8*i+4)][sub_word.slc<4>(8*i)];
    sub_word.set_slc(i*8,tmp);
}

sub_word=sub_word^Rcon[num_ofkey];
```

KEY GENERATION : STEP 4

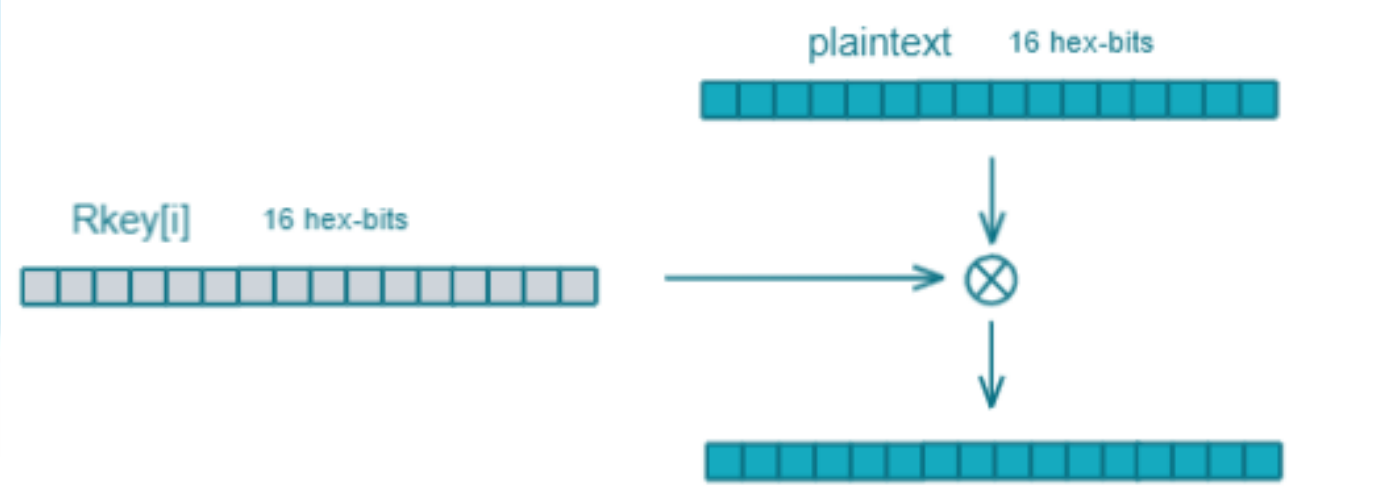



```
// XORS
W[0]    = W[0]^sub_word;
W[1]    = W[1]^W[0];
W[2]    = W[2]^W[1];
W[3]    = W[3]^W[2];

Rkey.set_slc(0,W[0]);
Rkey.set_slc(32,W[1]);
Rkey.set_slc(64,W[2]);
Rkey.set_slc(96,W[3]);
```

ENCRYPTION PROCESS

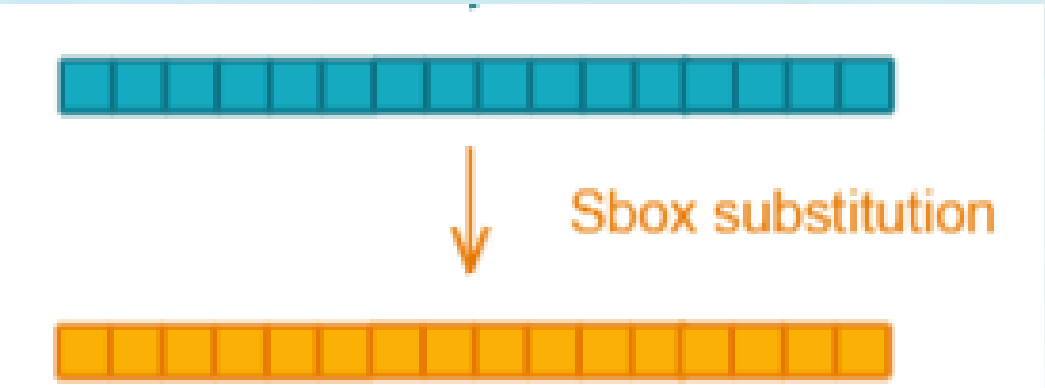


ENCRYPTION PROCESS : XOR KEY



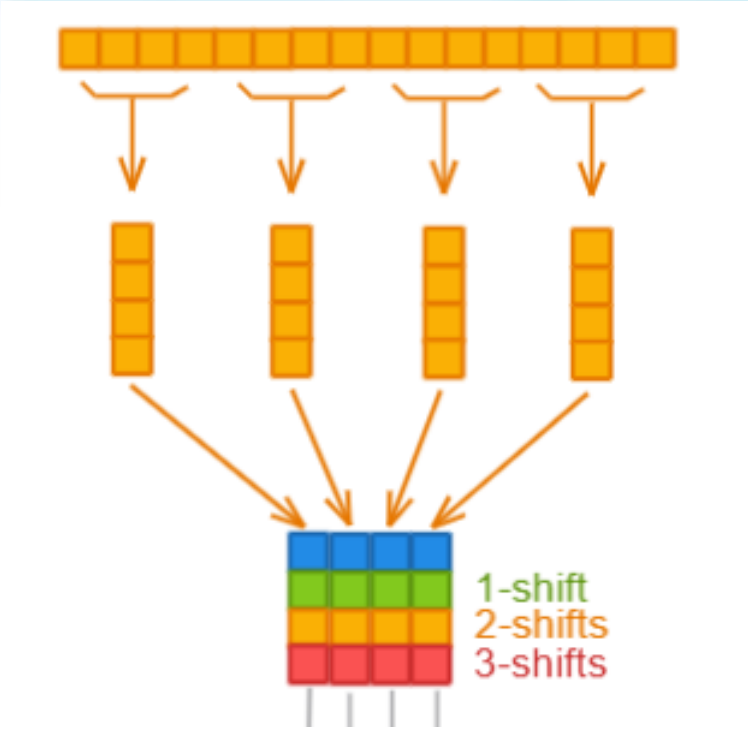

```
void XorRoundKey(ac_int<8,false> state_array[4][4],ac_int<128,false> &Rkey){
    short iter = 0;
    XOR_ROUND_KEY_j:for(int j = 0; j < 4; j++){
        XOR_ROUND_KEY_i:for(int i = 0; i < 4; i++){
            state_array[i][j] = state_array[i][j]^Rkey.slc<8>(8*iter);
            iter++;
        }
    }
}
```

ENCRYPTION PROCESS: SBOX SUB



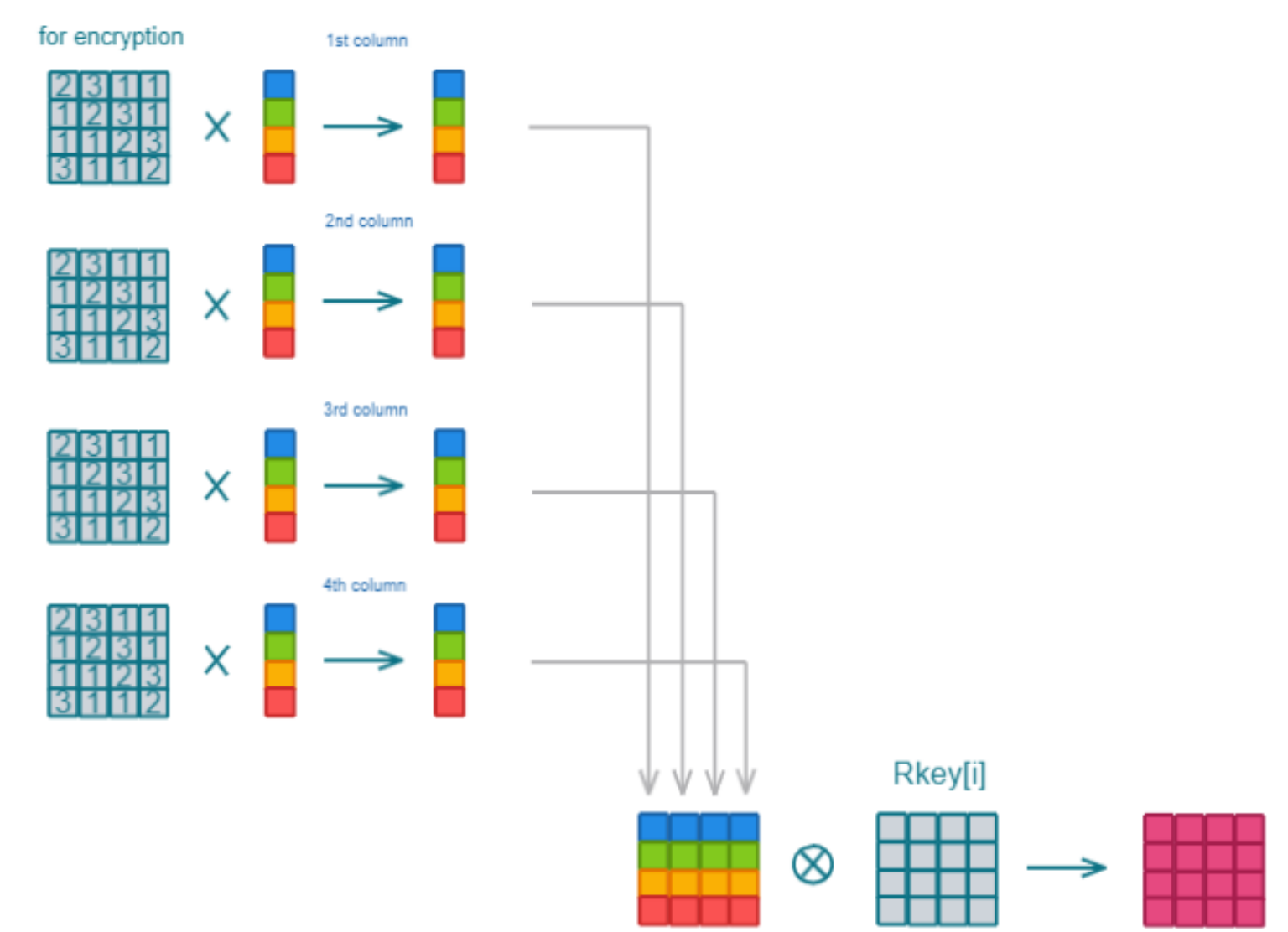

```
void SubBytes(ac_int<8,false> state_array[4][4]){
    SUB_BYTES_i:for(int i = 0; i < 4; i++){
        SUB_BYTES_j:for(int j = 0; j < 4; j++){

            state_array[i][j] = sbox[state_array[i][j].slc<4>(4)][state_array[i][j].slc<4>(0)];
        }
    }
}
```

ENCRYPTION PROCESS : SHIFT ROWS




```
// 2nd row left shift 1 step
temp1 = state_array[1][0];
state_array[1][0]=state_array[1][1];
state_array[1][1]=state_array[1][2];
state_array[1][2]=state_array[1][3];
state_array[1][3]=temp1;

// 3rd row left shift 2 steps
temp1 = state_array[2][0];
temp2 = state_array[2][1];
state_array[2][0]=state_array[2][2];
state_array[2][1]=state_array[2][3];
state_array[2][2]=temp1;
state_array[2][3]=temp2;

//4th row left shift 3 steps
temp1 = state_array[3][0];
temp2 = state_array[3][1];
temp3 = state_array[3][2];
state_array[3][0]=state_array[3][3];
state_array[3][1]=temp1;
state_array[3][2]=temp2;
state_array[3][3]=temp3;
```

MIX COLUMNS & XOR



GALOIS MUL @ GF(2^8)

Multiplication in a finite field is multiplication modulo an irreducible reducing polynomial used to define the finite field.

$$(x^6 + x^4 + x + 1)(x^7 + x^6 + x^3 + x)$$

$$= (x^{13} + x^{12} + x^9 + \mathbf{x^7}) + (x^{11} + x^{10} + \mathbf{x^7} + x^5) + (x^8 + \mathbf{x^7} + x^4 + x^2) + (\mathbf{x^7} + x^6 + x^3 + x)$$

$$= x^{13} + x^{12} + x^9 + x^{11} + x^{10} + x^5 + x^8 + x^4 + x^2 + x^6 + x^3 + x$$

$$= x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$$

reducing polynomial

$$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x \text{ modulo } x^8 + x^4 + x^3 + x^1 + 1$$

$$= (11111101111110 \bmod 100011011)$$

$$= \{3F7E \bmod 11B\} = \{01\}$$

$$= 1 \text{ (decimal)}$$

ENCRYPTION PROCESS : MIX COL

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \longrightarrow \begin{aligned} d_0 &= 2 \bullet b_0 \oplus 3 \bullet b_1 \oplus 1 \bullet b_2 \oplus 1 \bullet b_3 \\ d_1 &= 1 \bullet b_0 \oplus 2 \bullet b_1 \oplus 3 \bullet b_2 \oplus 1 \bullet b_3 \\ d_2 &= 1 \bullet b_0 \oplus 1 \bullet b_1 \oplus 2 \bullet b_2 \oplus 3 \bullet b_3 \\ d_3 &= 3 \bullet b_0 \oplus 1 \bullet b_1 \oplus 1 \bullet b_2 \oplus 2 \bullet b_3 \end{aligned}$$

Regs in code

```
1*a = a; -> c[]
2*a = a<<1; -> b[]
3*a = a+2*a = a+a<<1; -> c[]^b[]
```

ENCRYPTION PROCESS : MIX COL

```
MIX_COL_j:for(int j = 0; j < 4; j++){
    MIX_COL_g:for(int g = 0; g < 4;g++){
        // Save it iternaly
        c[g]=state_array[g][j];
        // Check for high MSB
        h = c[g]&0x80;
        // MUL by 2
        b[g]=c[g] << 1;
        // Reduce if necessary
        if(h==0x80){
            b[g]^=0x1b;
        }
    }

    // MIX
    state_array[0][j] = b[0] ^ (b[1]^c[1]) ^ c[2] ^ c[3];
    state_array[1][j] = c[0] ^ b[1] ^ (b[2]^ c[2]) ^ c[3];
    state_array[2][j] = c[0] ^ c[1] ^ b[2]^ (b[3]^ c[3]);
    state_array[3][j] = (b[0] ^ c[0]) ^ c[1] ^ c[2] ^ b[3] ;

}
```

Save it to internal register

Check MSB

MUL by 2 via shifting

reducing polynomial

```
0x11B:100011011 9bit
0x1B : 00011011 8bit

0x80-0xFF:Numbers in this range will
overflow when mul by 2.
0x02*0x80=0x100:100000000
0x02*0xFF=0x1FE:111111110

e.g. 111111110^100011011=11100101
      11111110^ 00011011=11100101
```

ENCRYPTION PROCESS

```
void /*CCS_BLOCK*/encrypt(ac_int<8,false> state_array[4][4],ac_int<128,false> &Rkey){
    XorRoundKey(state_array, Rkey);
    ENCRYPT:for(int i = 0; i < 9; i++){
        SubBytes(state_array);
        ShiftRows(state_array);
        MixColumns(state_array);
        key_generation(Rkey,i);
        XorRoundKey(state_array, Rkey);
    }
    SubBytes(state_array);
    ShiftRows(state_array);
    key_generation(Rkey,9);
    XorRoundKey(state_array, Rkey);
}
```

VERIFICATION

Για το verification του αλγορίθμου έγιναν 284 tests με γνωστά plaintexts, κλειδιά και ciphertexts. Ο κώδικας πέρασε όλα τα τεστ με επιτυχία.

Τα ίδια τεστ εφαρμόστηκαν και στο decryption και στα versions όπου ο αλγόριθμος υλοποιήθηκε με blocks, όπου και πάλι πέρασε όλα τα τεστ με επιτυχία.

Τα txt files με τα προτεινόμενα plaintexts, κλειδιά και ciphertexts που χρησιμοποιήθηκαν για το testing προέρχονται από:

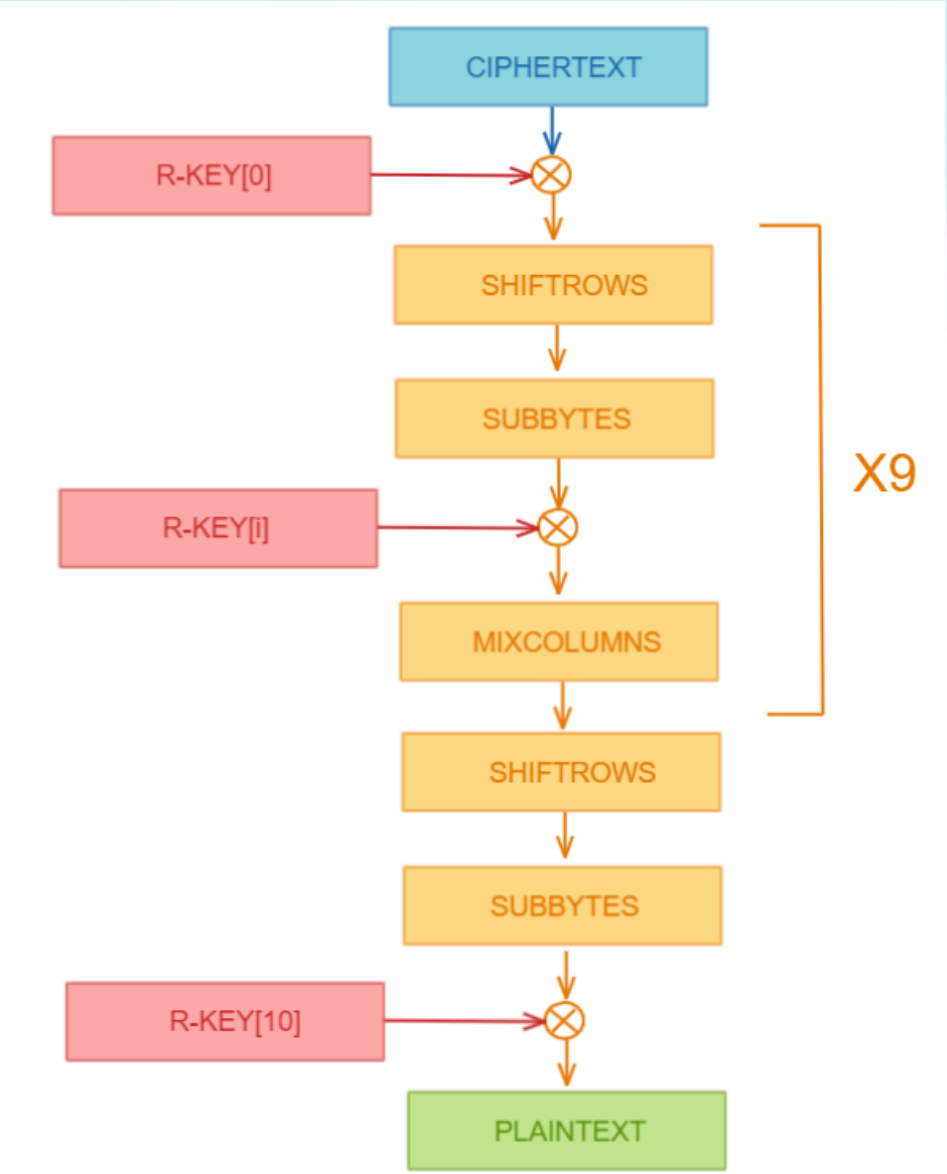
https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/block-ciphers?fbclid=IwAR0BrSHGcHcXBOmVbfL0Is7jw15dtg13KGTk4BsrX7TF8OmFHv6gTVHgfQY

```
Successfully read 6300 values:
Errors: 0
Error: 0%
```

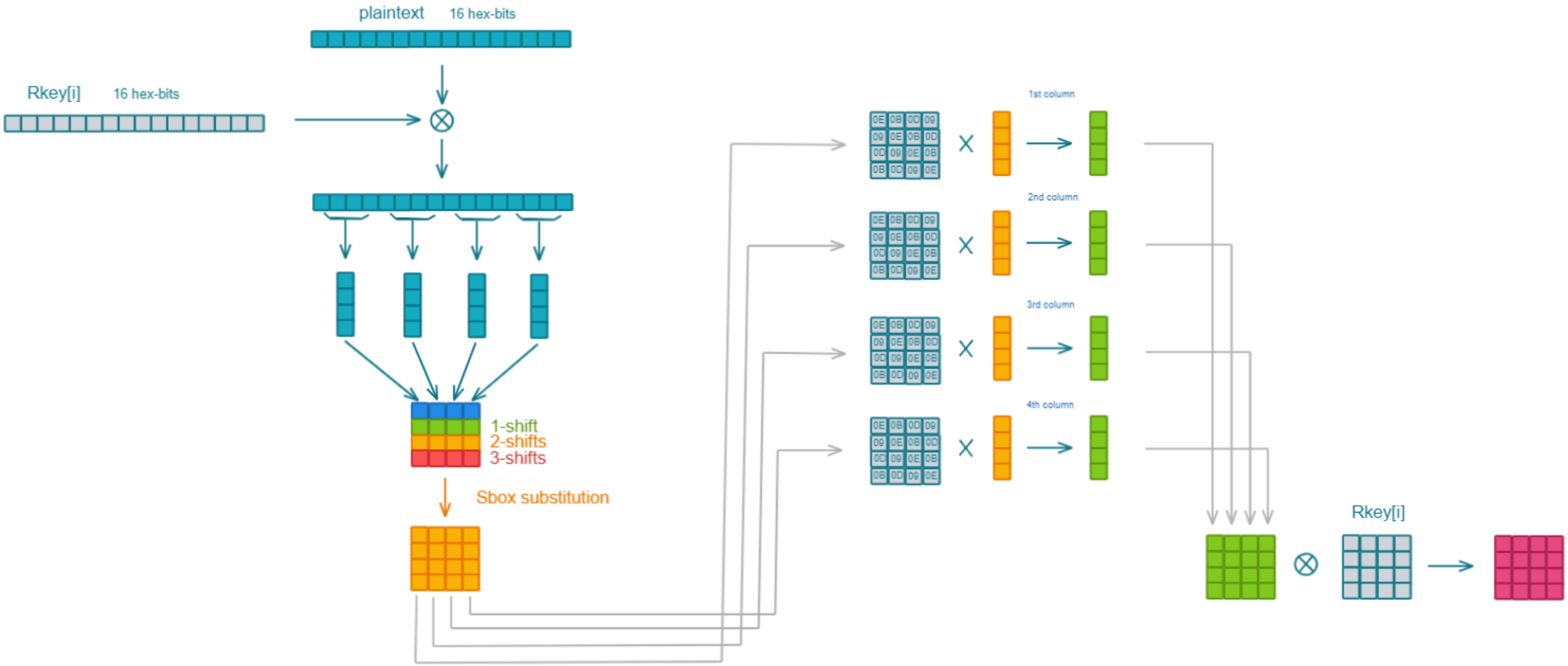
NIST Cryptographic Algorithm Validation Program CAVP

AES-128 ALGORITHM DECRYPTION

KEY: 128 BITS
CIPHERTEXT: 128 BITS
PLAINTEXT: 128 BITS

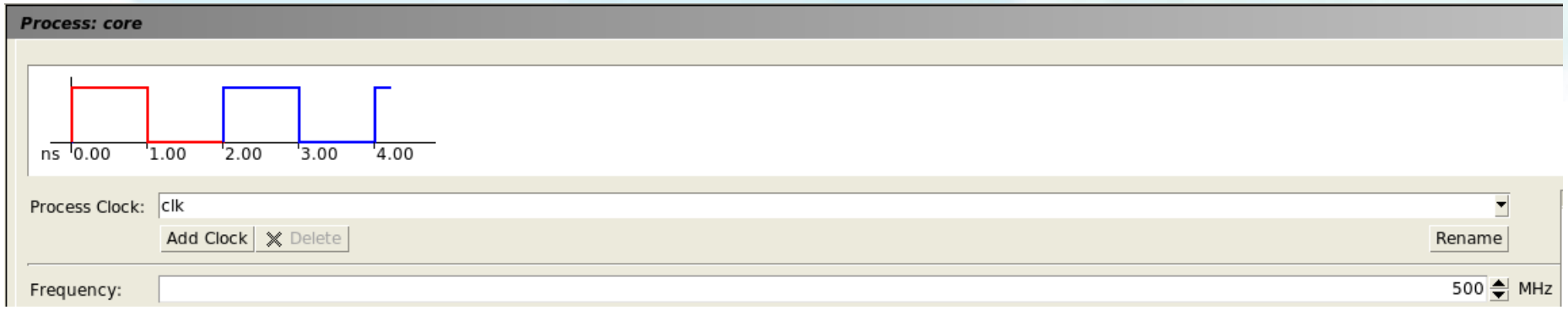


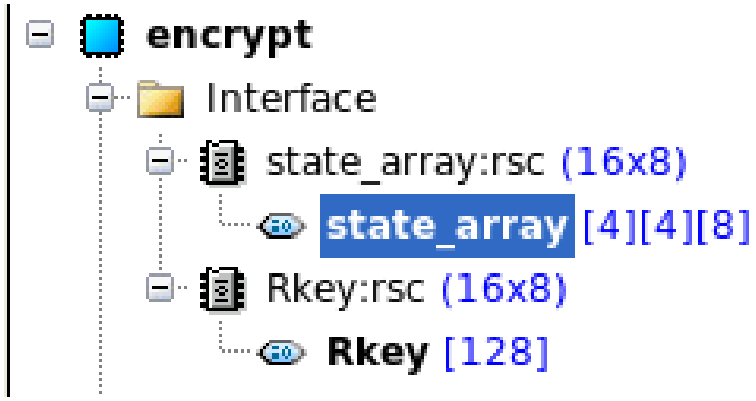
DECRYPTION PROCESS



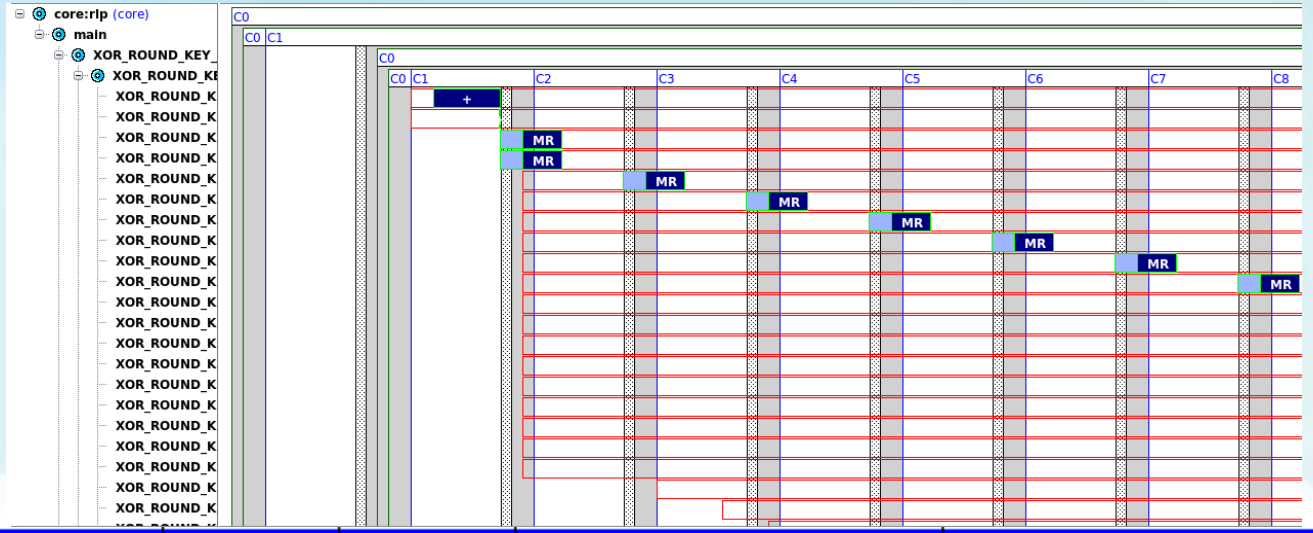
ENCRYPTION & DECRYPTION TESTS

Clock Freq @ 500MHz
State Array : 1R1W, 8bit word width
Key : 1R1W, 8bit word width



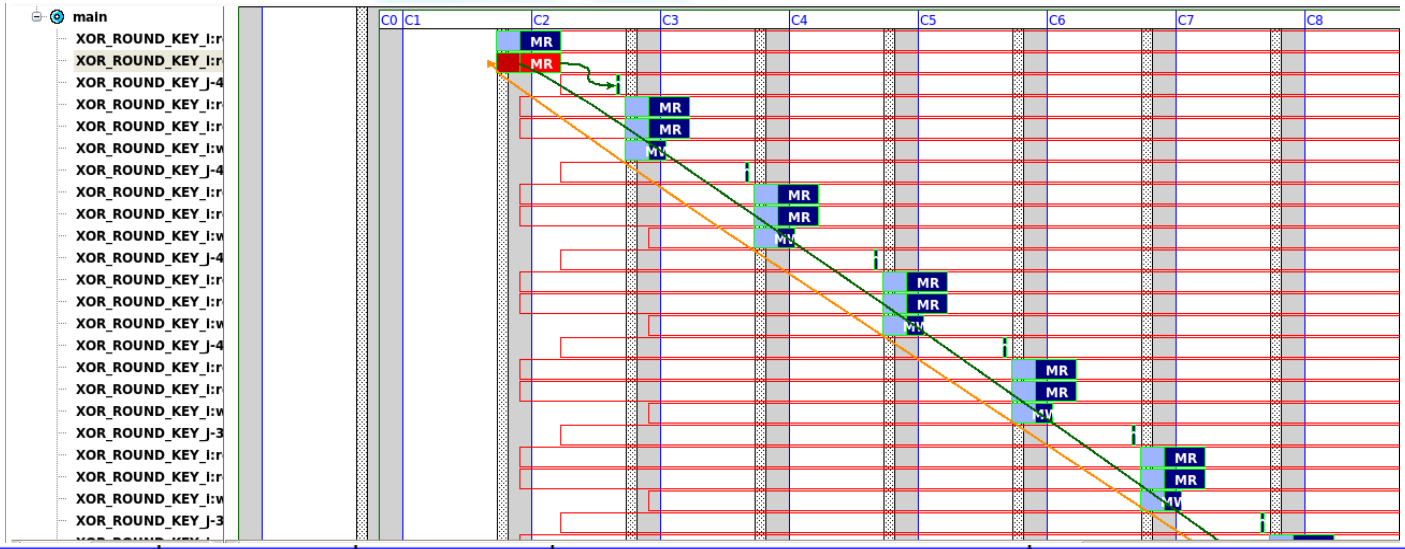


ENCRYPTION PROCESS : XOR KEY



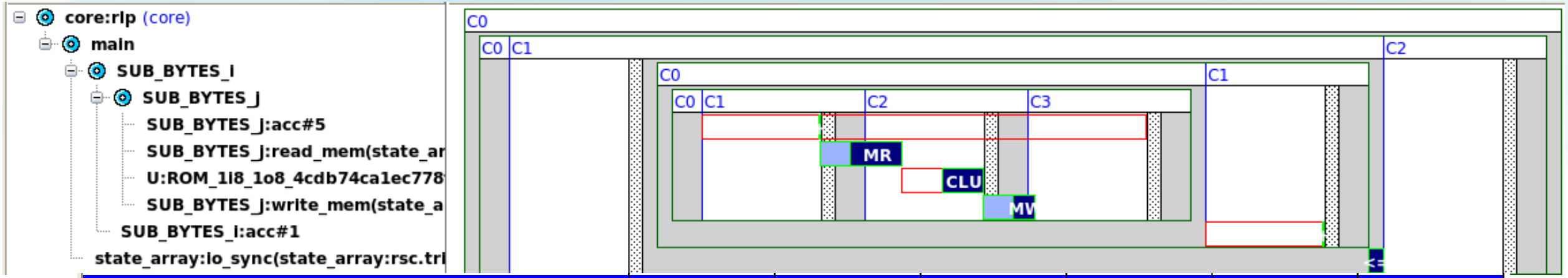
encrypt.v36 (allocate)	305	610.00	310	620.00	1869.64	

Both loops fully unrolled



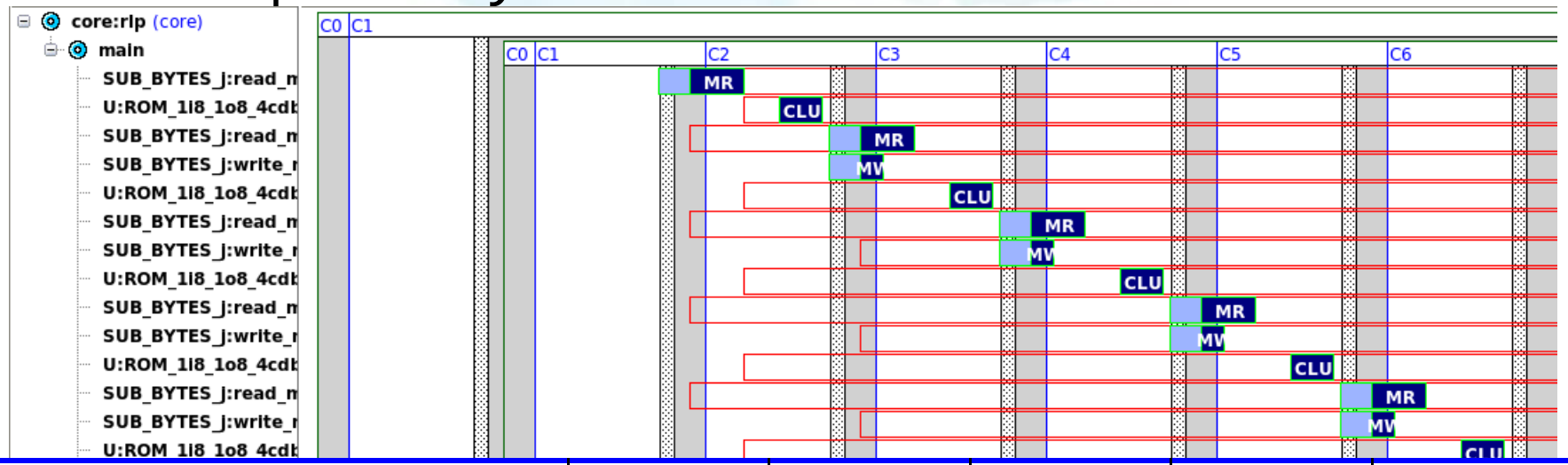
encrypt.v35 (allocate)	16	32.00	18	36.00	212.42	

ENCRYPTION PROCESS : SBOX SUB



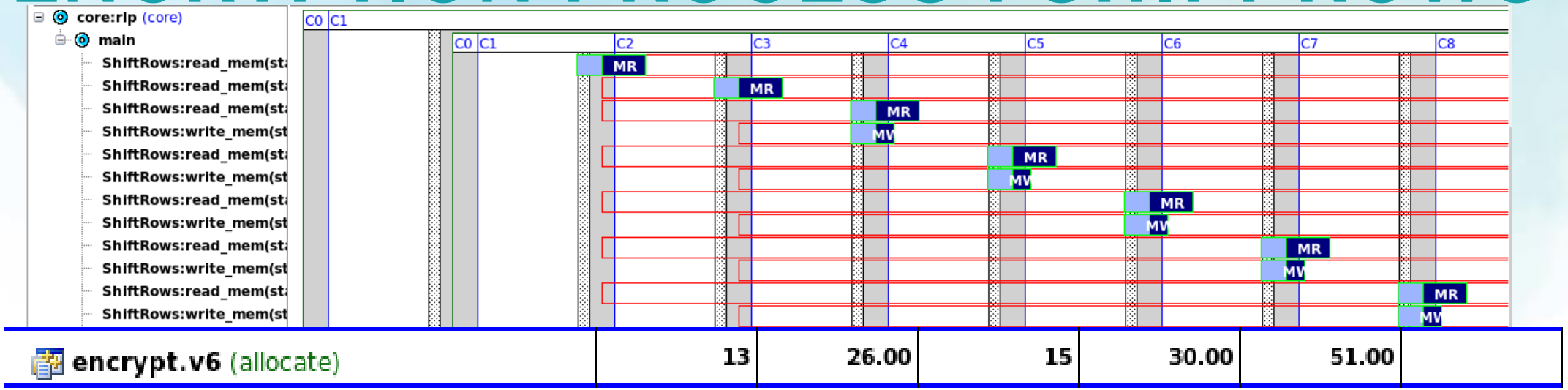
encrypt.v3 (allocate)	49	98.00	54	108.00	564.54	

Both loops fully unrolled

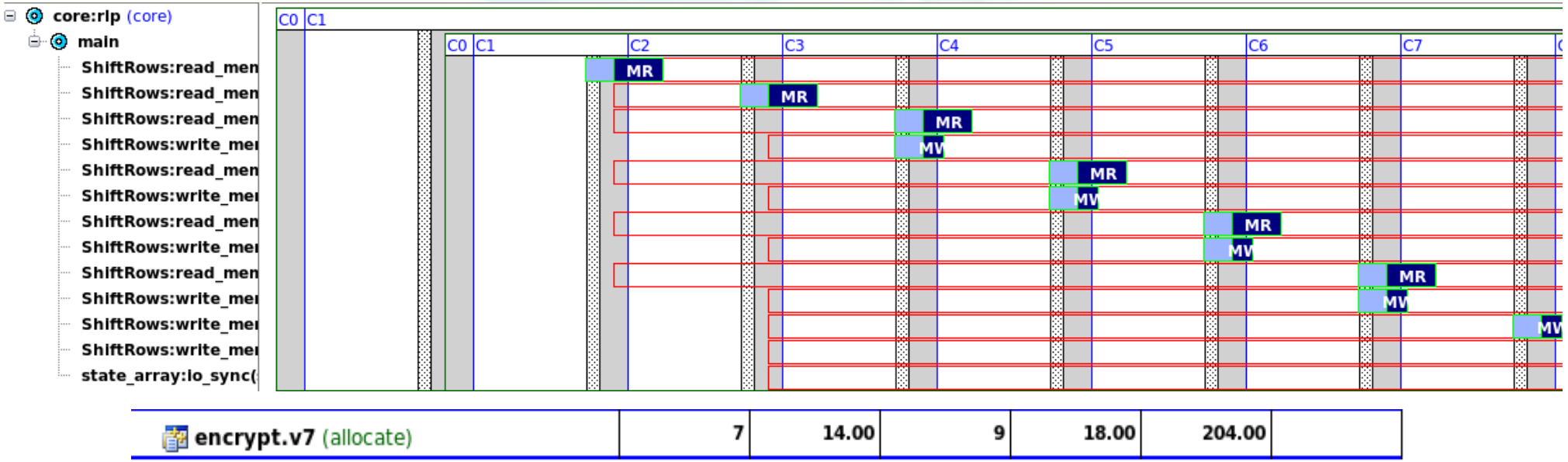


encrypt.v5 (allocate)	16	32.00	18	36.00	490.77	

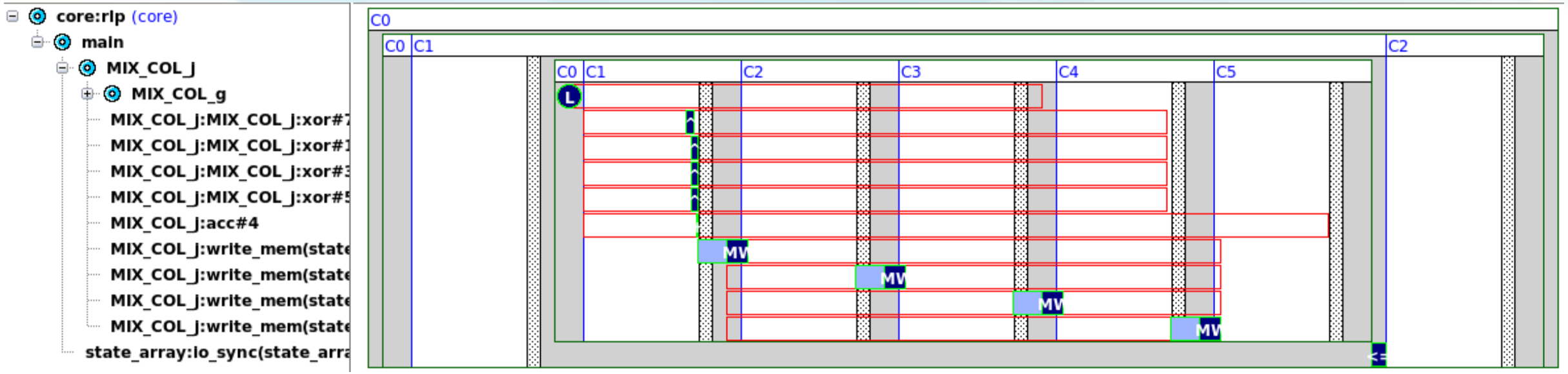
ENCRYPTION PROCESS : SHIFT ROWS



Double word length 8bit->16bit

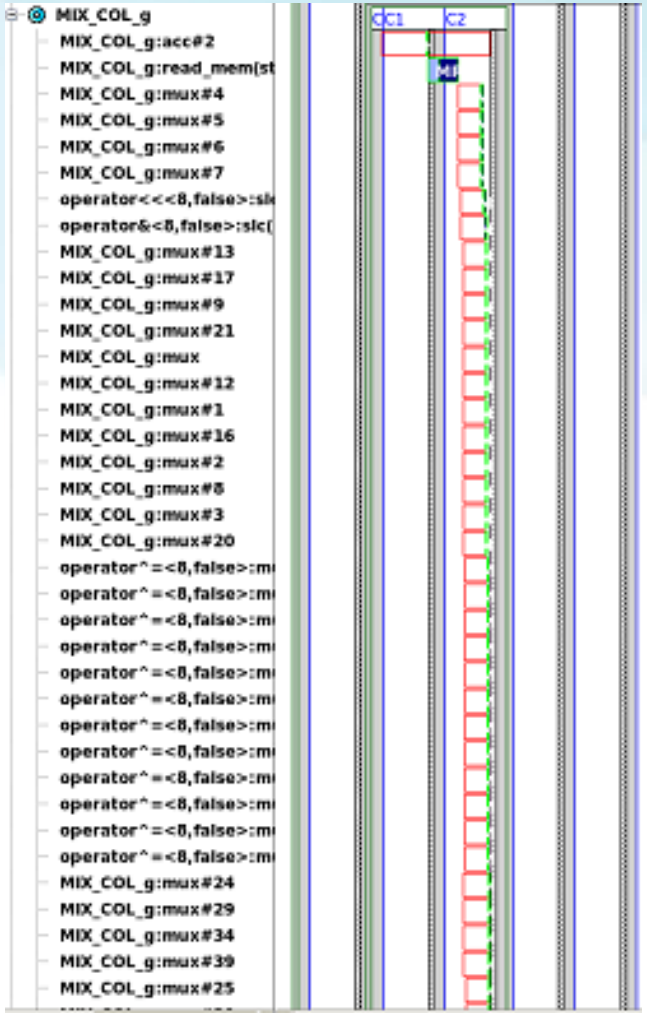


ENCRYPTION PROCESS : MIX COL



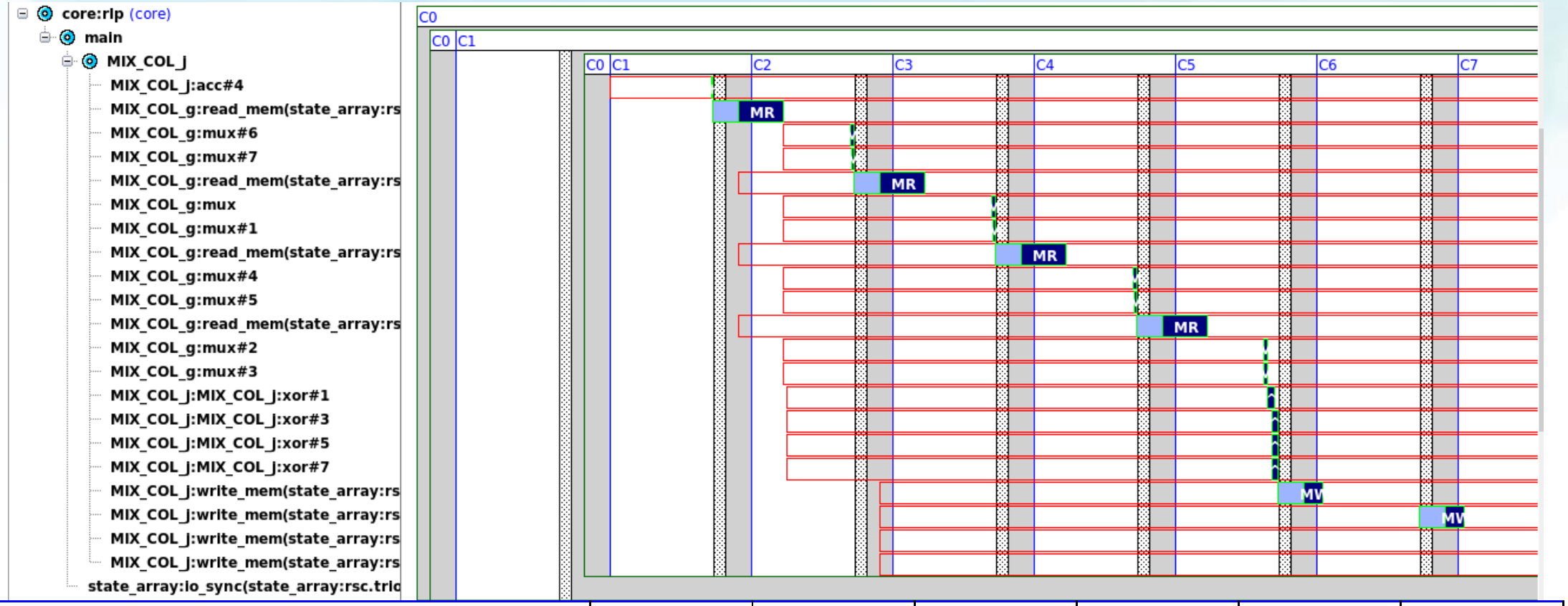
ENCRYPTION PROCESS : MIX COL

encrypt.v13 (allocate)	50	100.00	54	108.00	1266.06	



ENCRYPTION PROCESS : MIX COL

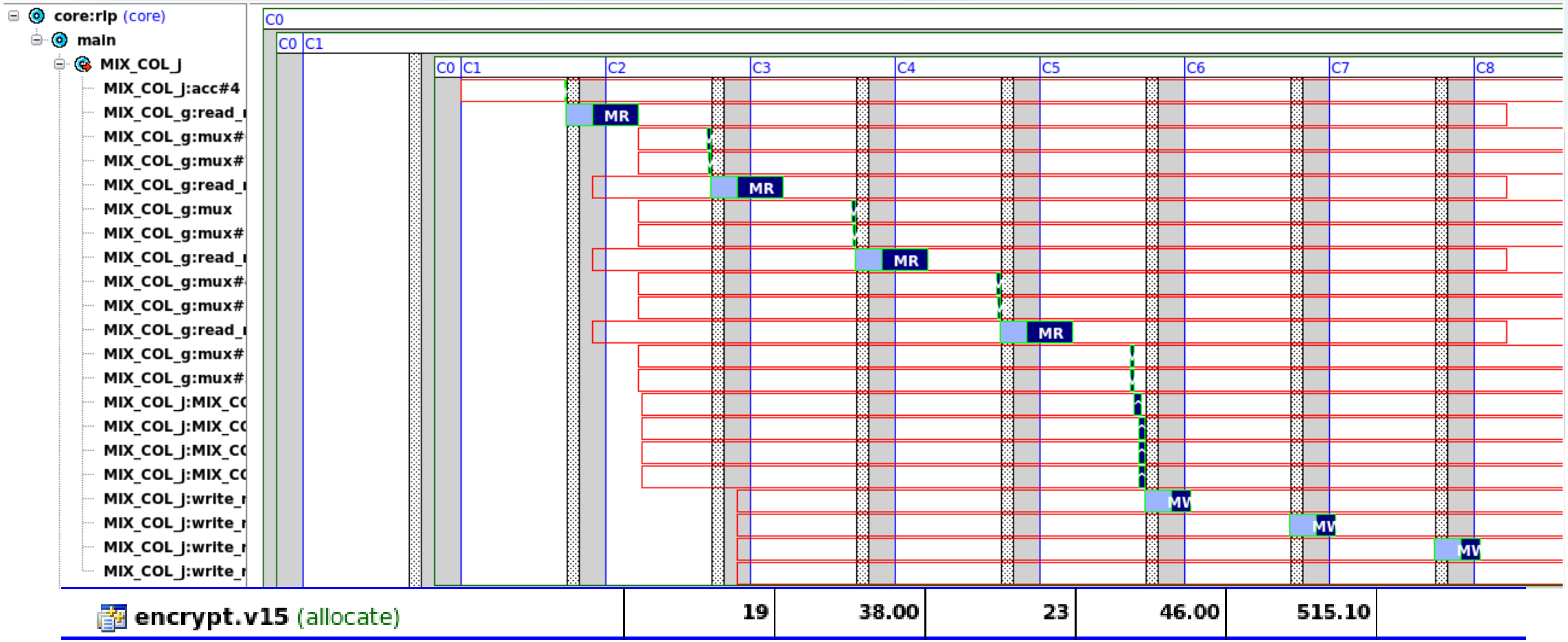
MIX_COL_g fully unrolled



encrypt.v14 (allocate)	34	68.00	38	76.00	464.10	

ENCRYPTION PROCESS : MIX COL

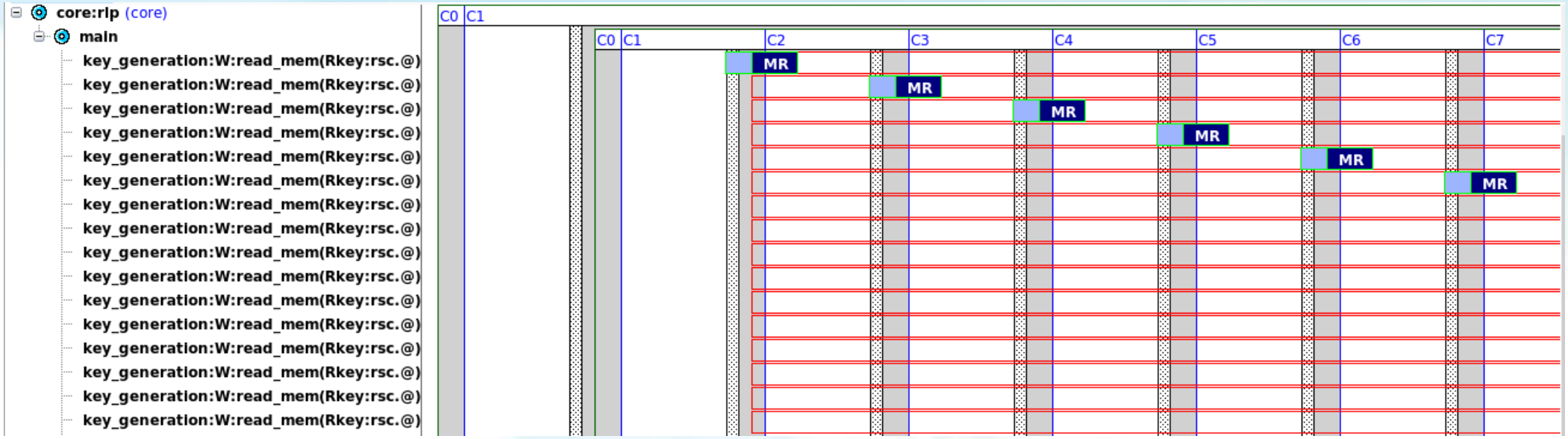
MIX_COL_G fully unrolled, MIX_COL_J pipelined II=4

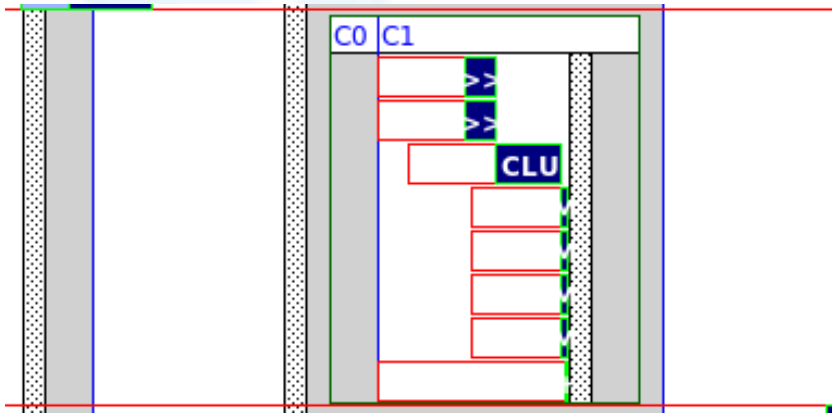


MIX_COL_G fully unrolled, MIX_COL_J pipelined II=4, 16 bit word width

```
insufficient resources 'ccs_sample_mem.ccs_ram_sync_1R1W_rwport(1,16,3,8,8,16,5)' to schedule '/encrypt/core'. 2 are needed, but only 1 instances are available
please merge accesses to 'state_array:rsc' to reduce the number of required resources
```

ENCRYPTION PROCESS : KEY GEN

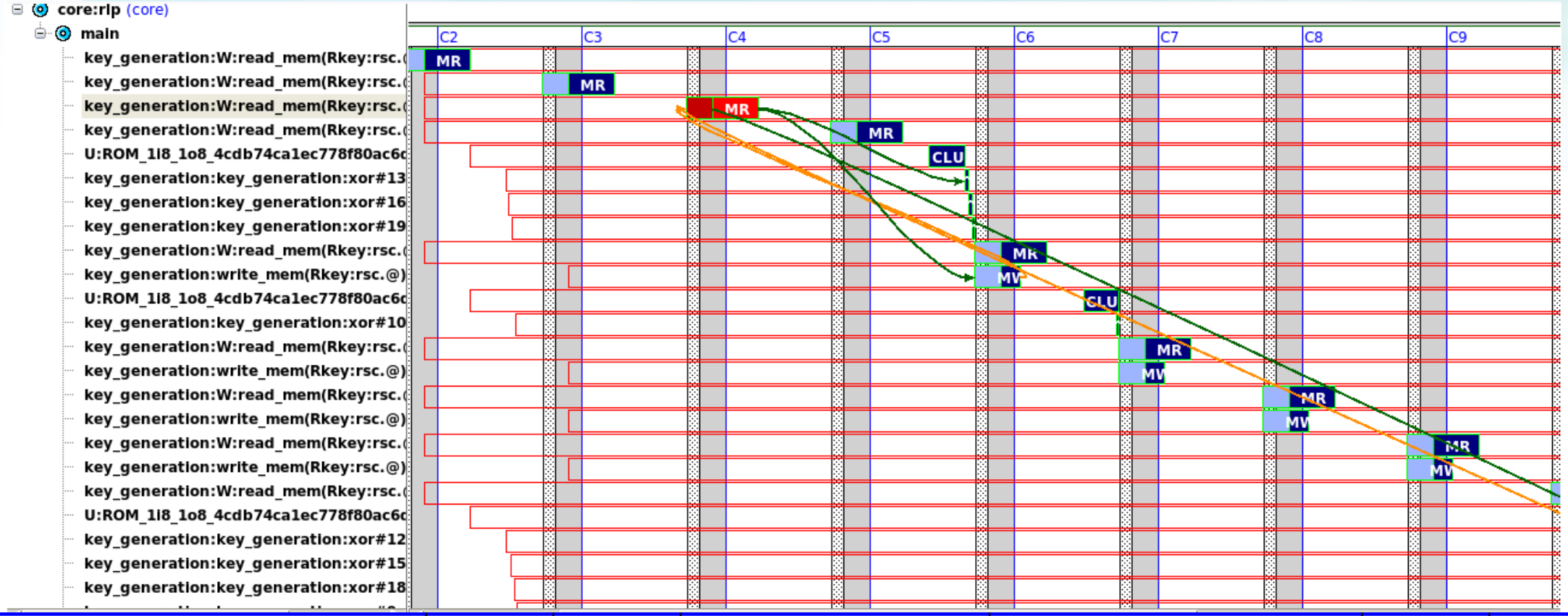




encrypt.v38 (allocate)	36	72.00	38	76.00	2170.27	

ENCRYPTION PROCESS : KEY GEN

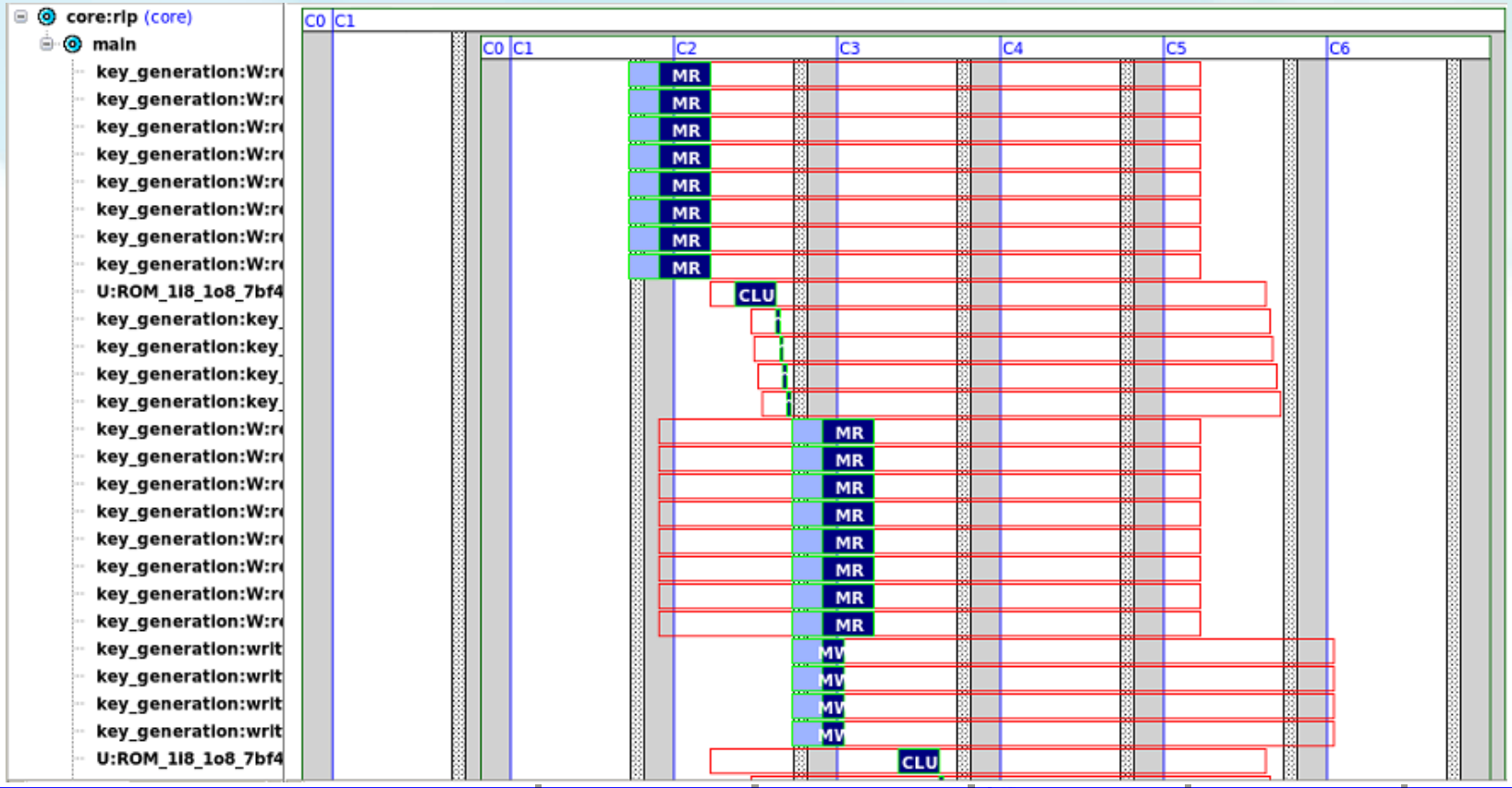
Both loops fully unrolled



encrypt.v39 (allocate)	20	40.00	22	44.00	970.66

ENCRYPTION PROCESS : KEY GEN

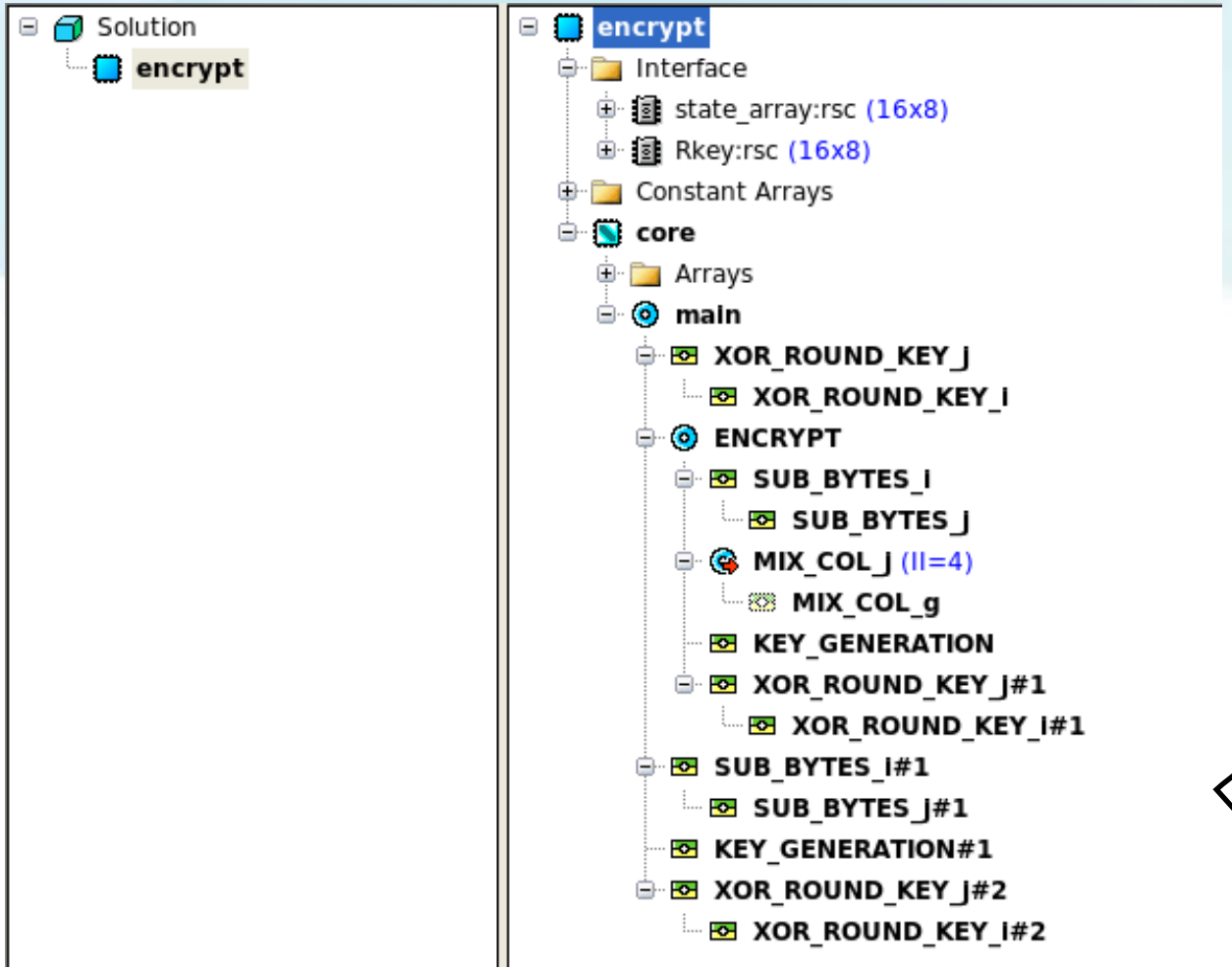
Both loops fully unrolled, block size = 2



encrypt.v48 (allocate)	4	8.00	6	12.00		1174.66

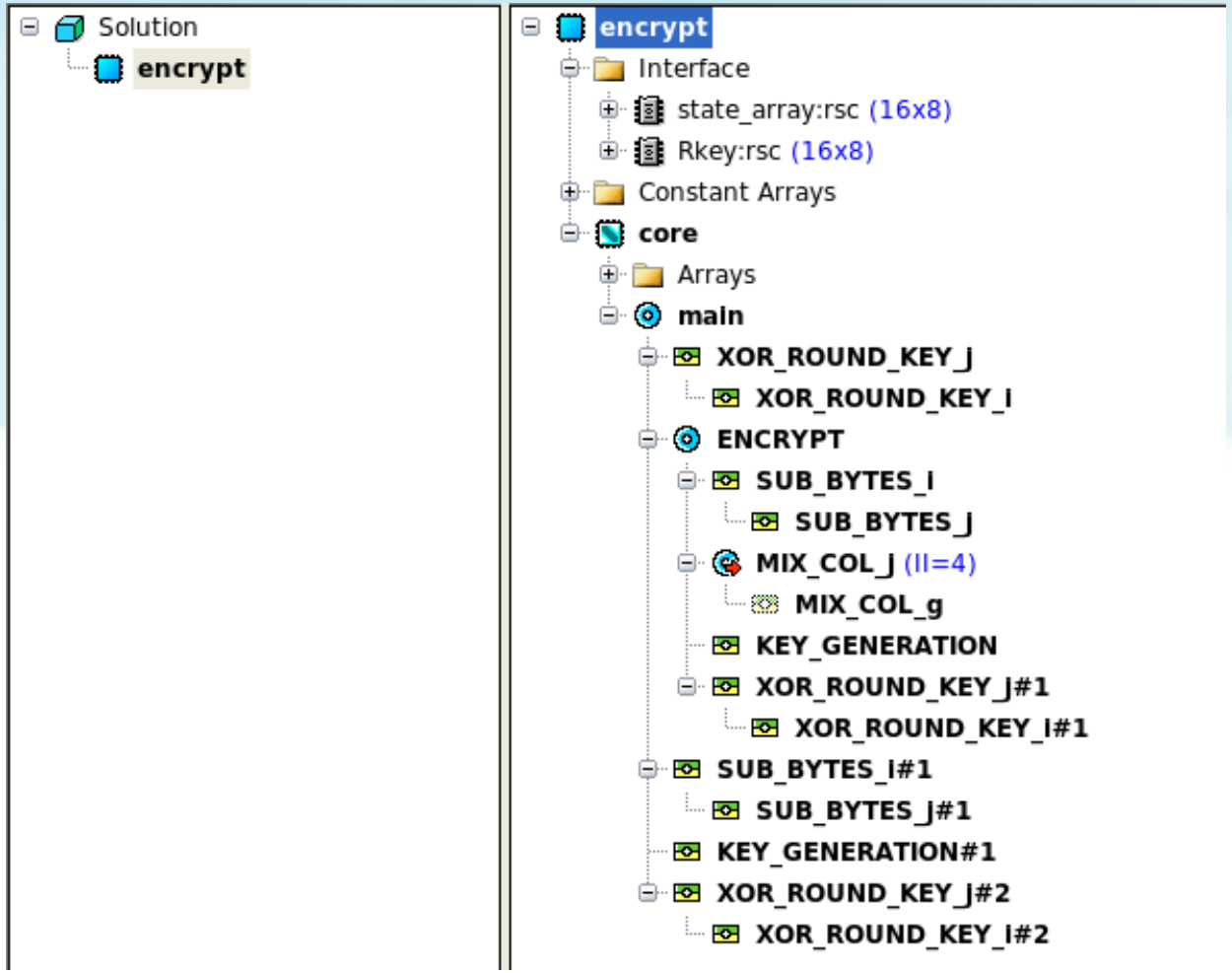
ENCRYPTION METRICS

encrypt.v33 (allocate)	4562	9124.00	4567	9134.00	4891.74	



encrypt.v32 (allocate)	638	1276.00	640	1280.00	2602.19	

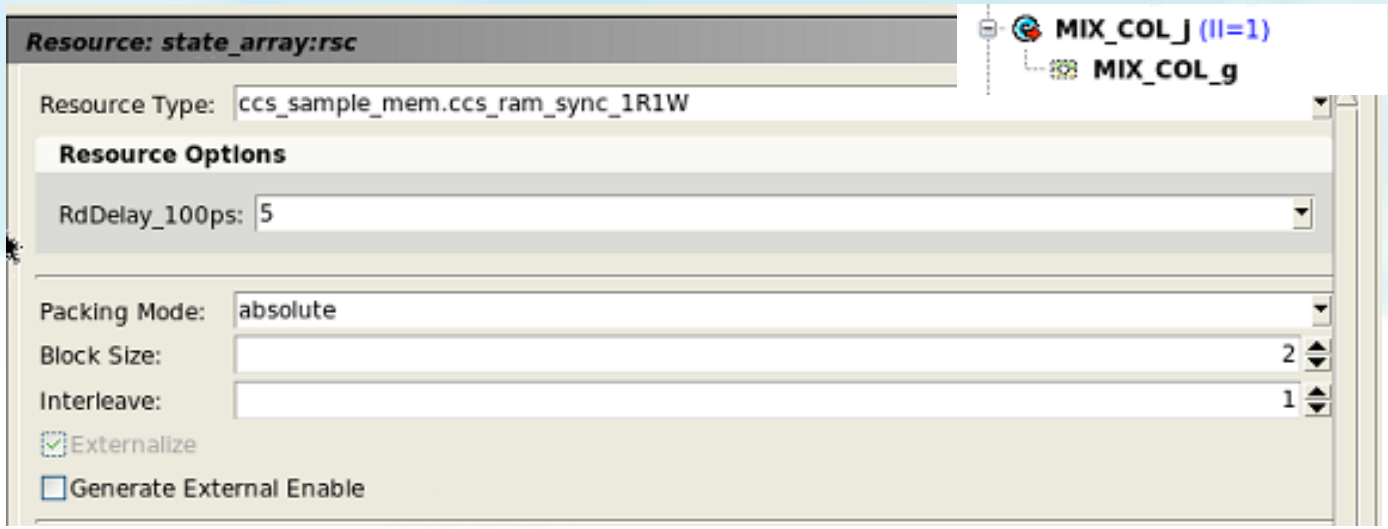
ENCRYPTION NOTES



Cannot pipeline main and ENCRYPT because of data dependencies. We must wait for an encryption round to finish before starting the next one.

ENCRYPTION EXTRAS

Blocked memory



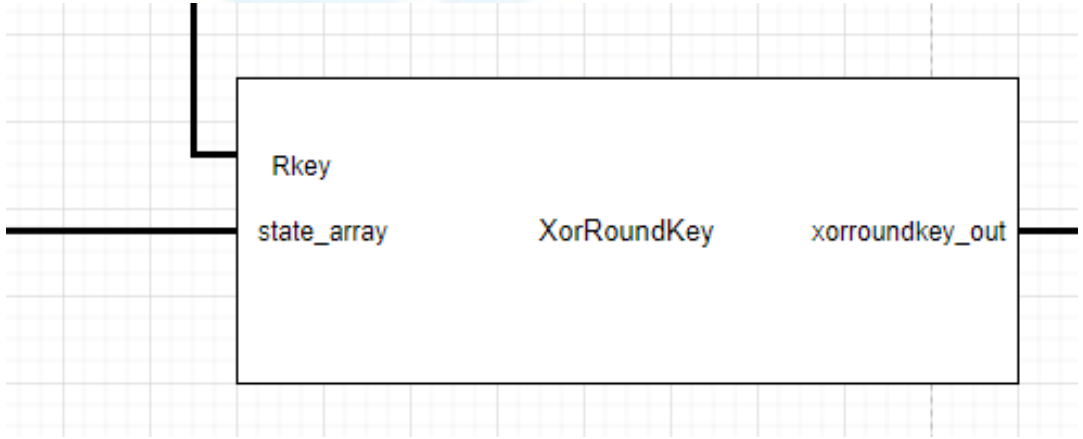
encrypt.v18 (extract)	452	904.00	454	908.00	0.63	6627.18

@1GHz

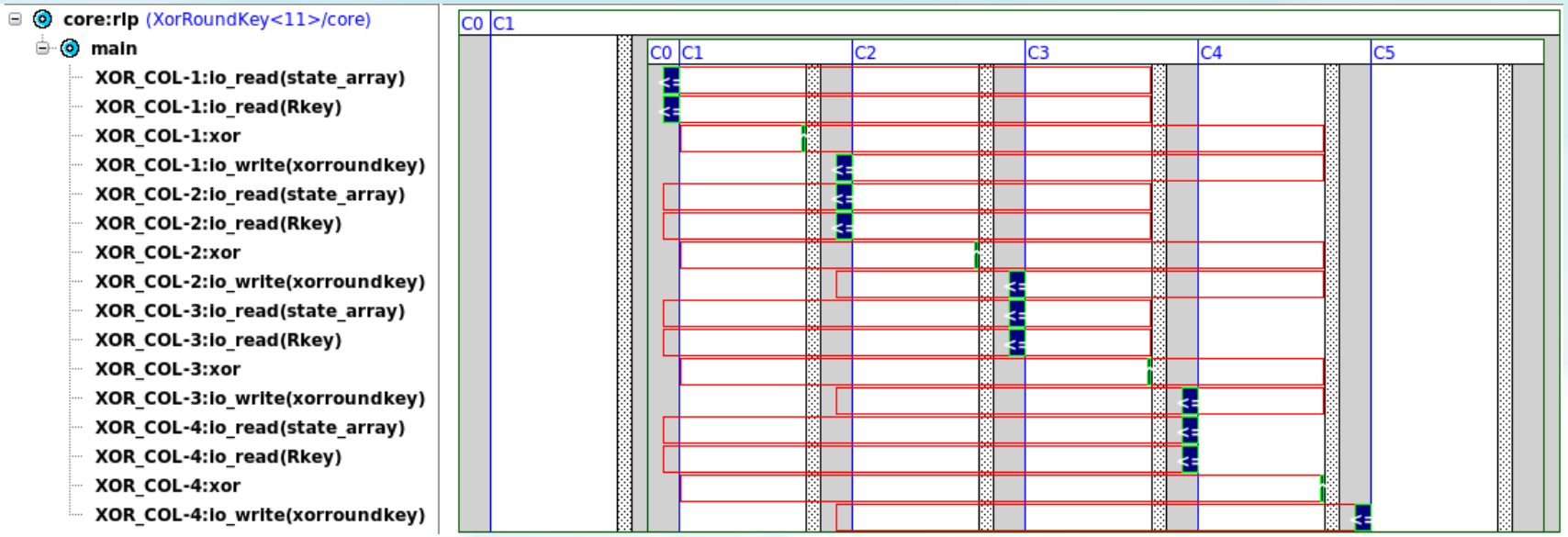
encrypt.v27 (extract)	461	461.00	463	463.00	0.00	6740.24

ENCRYPTION BLOCKS: XORKEY

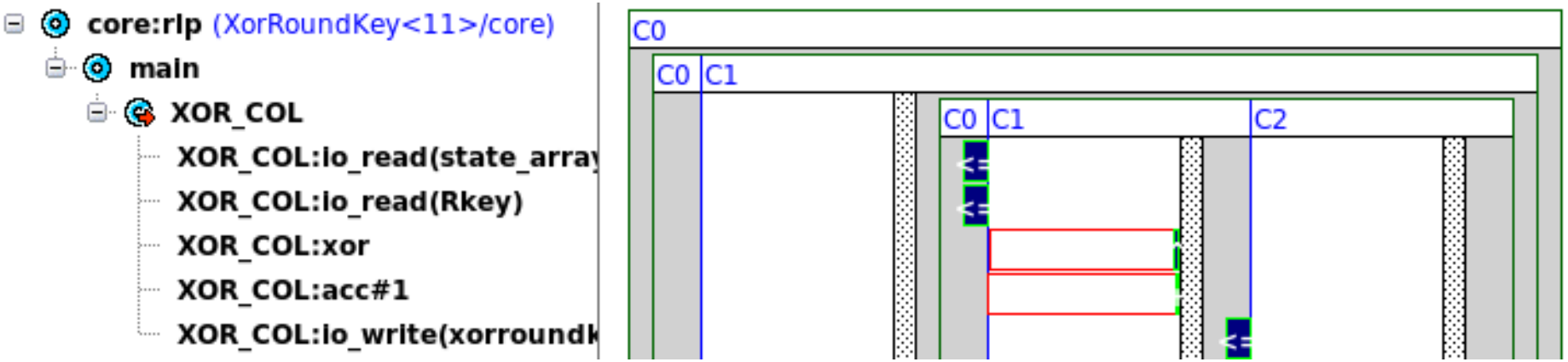
```
if(state_array.available(4) && Rkey.available(4)){
    XOR_COL:for(int col = 0; col < 4; col++){
        state = state_array.read();
        key = Rkey.read();
        state = state ^ key;
        xorroundkey.write(state);
    }
}
```



ENCRYPTION BLOCKS: XORKEY

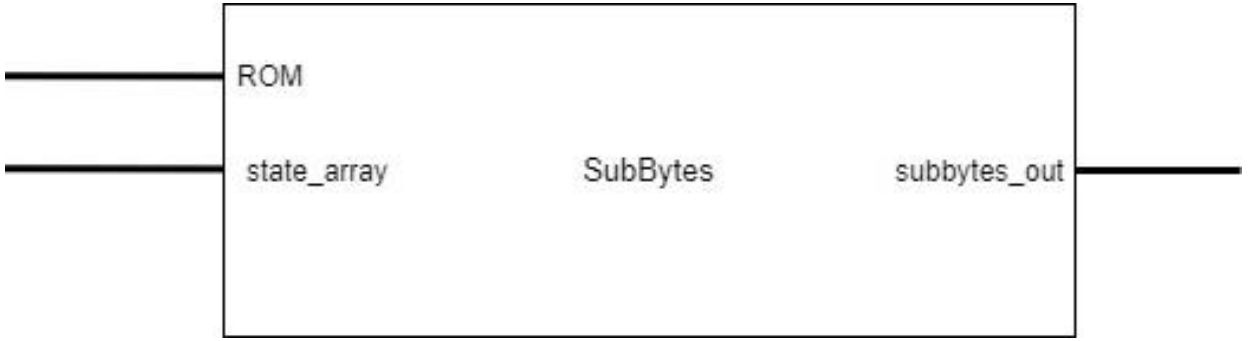


Pipelined II=1

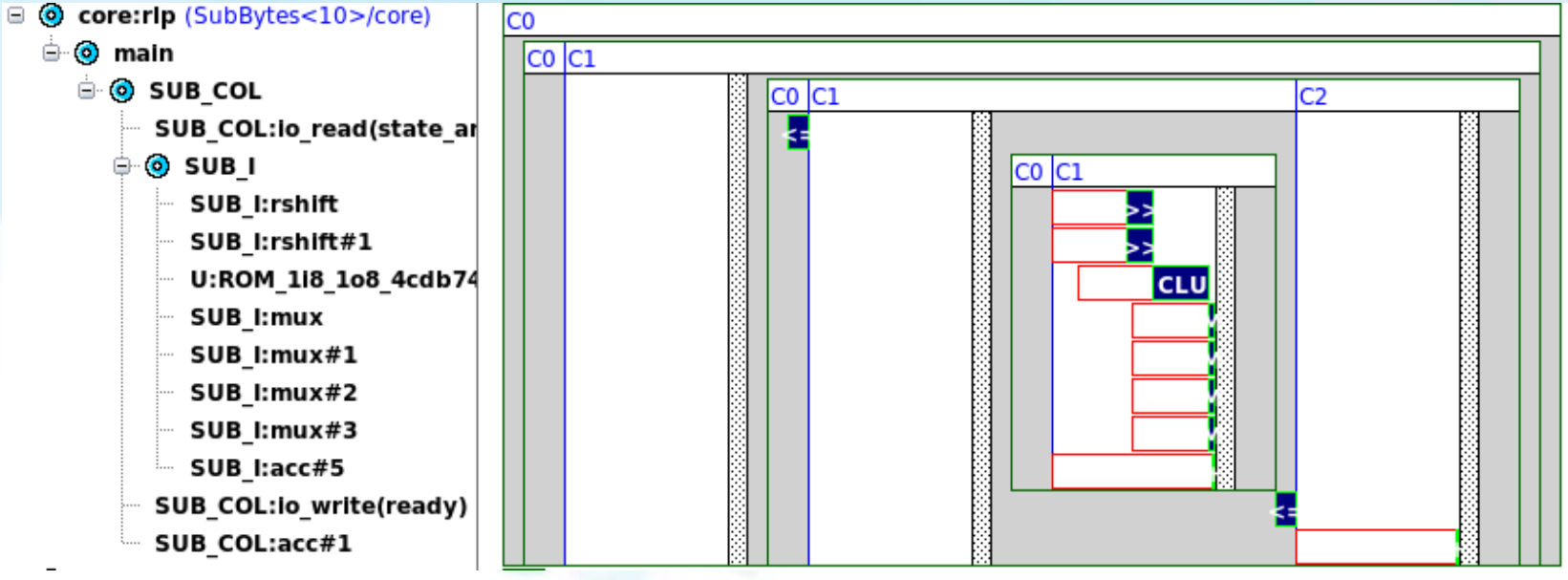


ENCRYPTION BLOCKS: SUB BYTES

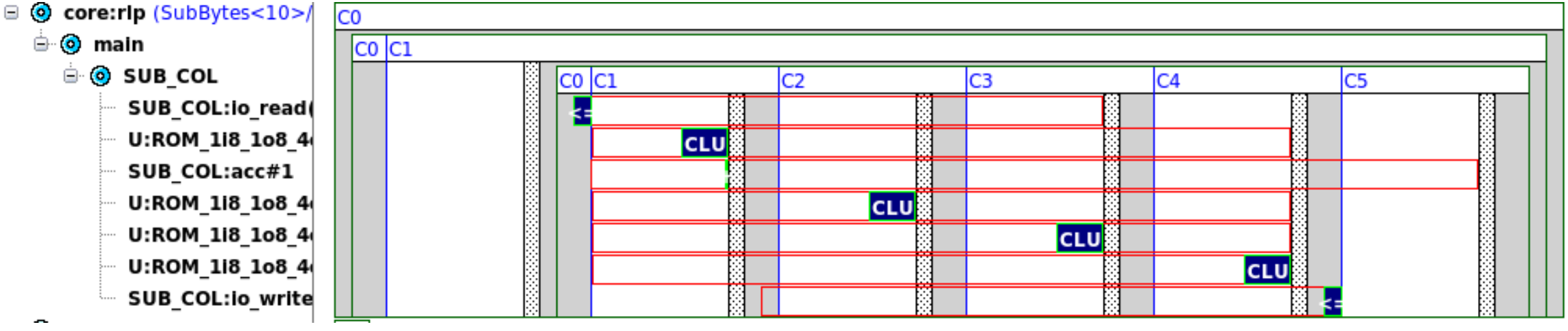
```
if(state_array.available(4)){
    for(int count_col = 0; count_col < 4; count_col ++){
        tmp=state_array.read();
        for(int i = 0; i < 4; i++){
            state[i][count_col] = sbox[tmp.slc<8>(i*8).slc<4>(4)][tmp.slc<8>(i*8).slc<4>(0)];
            col.set_slc(8*i,state[i][count_col]);
        }
        subbytes_out.write(col);
    }
}
```



ENCRYPTION BLOCKS: SUB BYTES



SUB_I fully unrolled



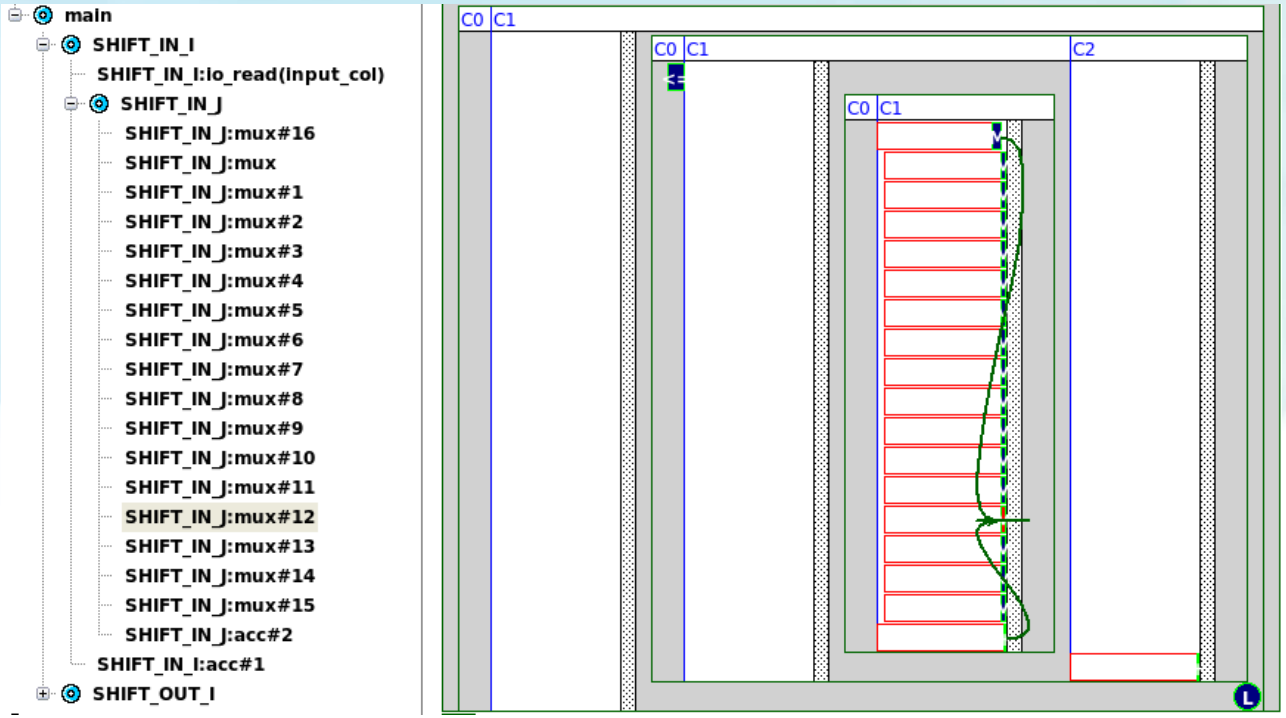
ENCRYPTION BLOCKS: SHIFT ROWS

```
if(input_col.available(4)){
    for(int i = 0; i < 4; i++){
        temp_col = input_col.read();
        for (int j = 0; j < 4; j++){
            state_array[j][i] = temp_col.slc<8>(j*8);
        }
    }

    // 2nd row left shift 1 step
    temp1 = state_array[1][0];
    state_array[1][0]=state_array[1][1];
    state_array[1][1]=state_array[1][2];
    state_array[1][2]=state_array[1][3];
    state_array[1][3]=temp1;
```

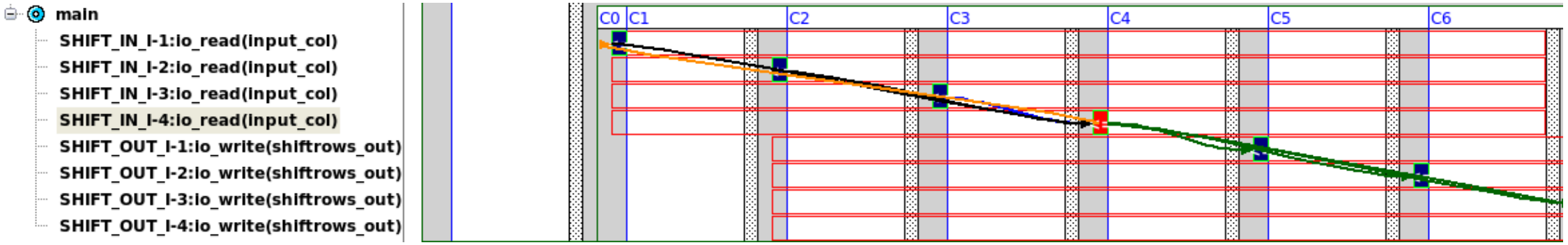
```
// 3rd row left shift 2 steps
temp1 = state_array[2][0];
temp2 = state_array[2][1];
state_array[2][0]=state_array[2][2];
state_array[2][1]=state_array[2][3];
state_array[2][2]=temp1;
state_array[2][3]=temp2;


//4th row left shift 3 steps
temp1 = state_array[3][0];
temp2 = state_array[3][1];
temp3 = state_array[3][2];
state_array[3][0]=state_array[3][3];
state_array[3][1]=temp1;
state_array[3][2]=temp2;
state_array[3][3]=temp3;
```

```
for(int i = 0; i < 4; i++){
    for(int j = 0; j < 4; j++){
        temp_col.set_slc(j*8,state_array[j][i]);
    }
    shiftrows_out.write(temp_col);
}
```

ENCRYPTION BLOCKS: SHIFT ROWS



Both loops fully unrolled

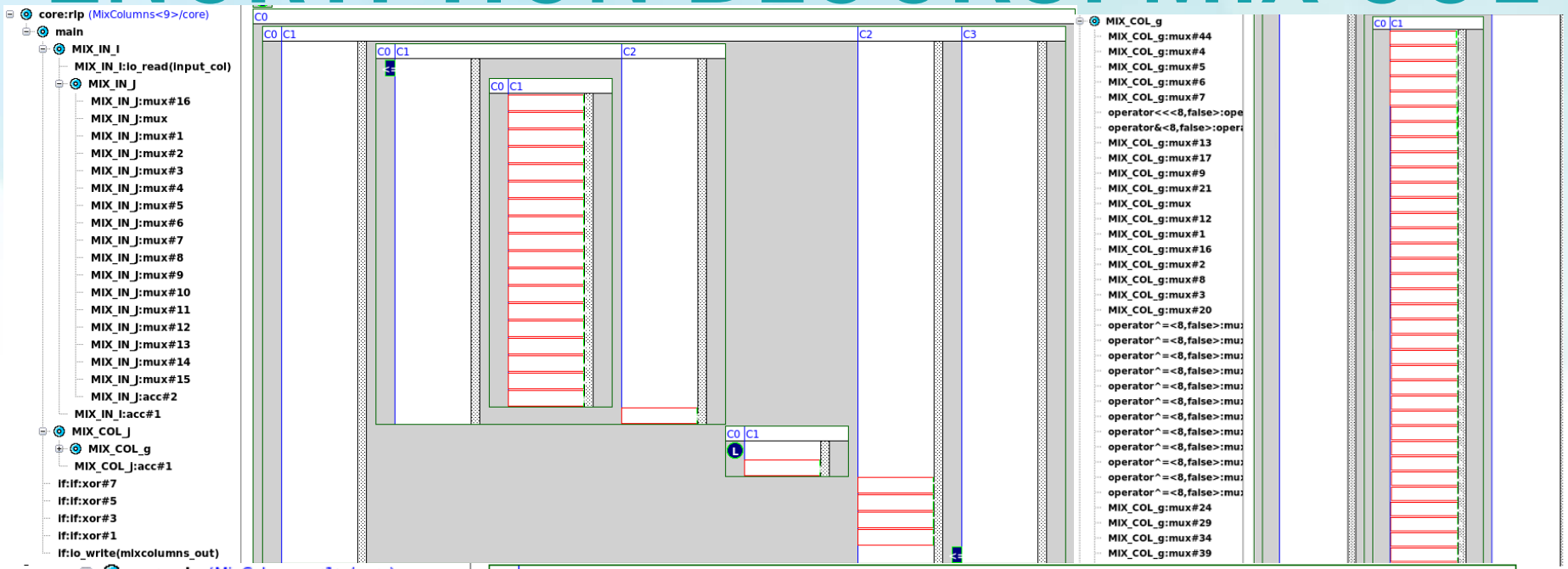


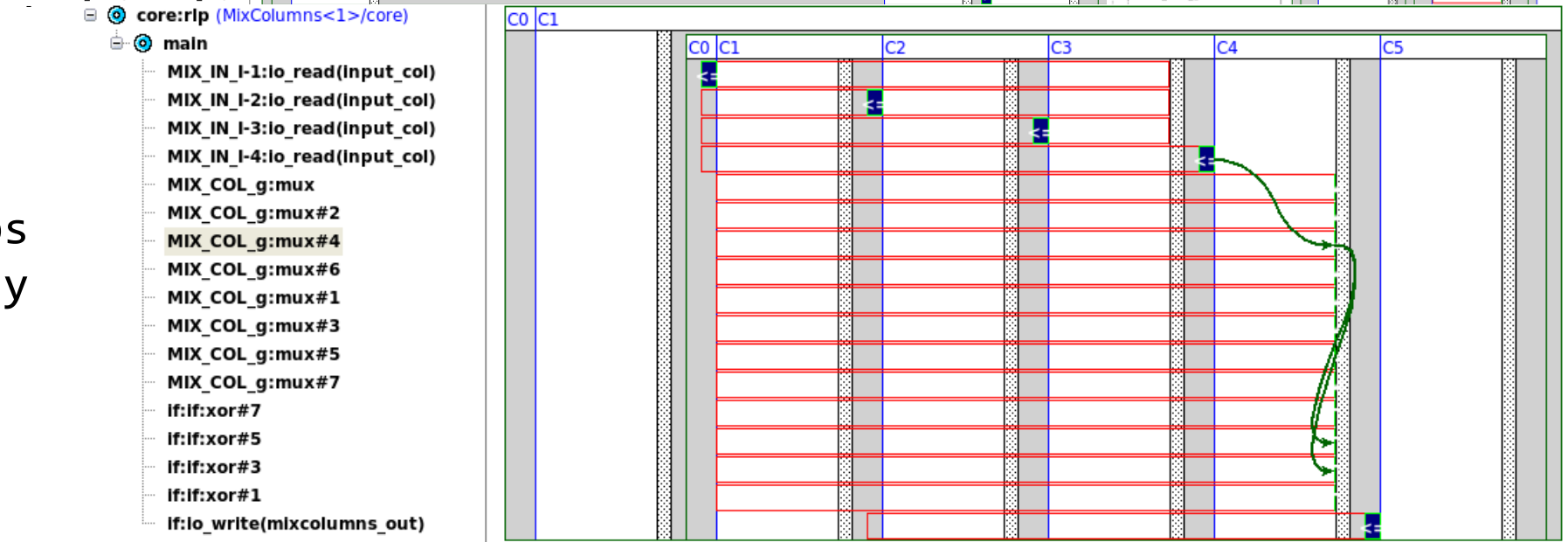
ENCRYPTION BLOCKS: MIX COL

```
if(input_col.available(1)){
    for(int i = 0; i < 4; i++){
        temp_col = input_col.read();
        for(int j = 0; j < 4; j++){
            state_array[j][i] = temp_col.slc<8>(8*j);
        }
    }
    MIX_COL_j:for(int j = 0; j < 4; j++){
        MIX_COL_g:for(int g = 0; g < 4;g++){
            c[g]=state_array[g][j];
            h = c[g]&0x80;
            b[g]=c[g] << 1;
            if(h==0x80){
                b[g]^=0x1b;
            }
        }
```

```
mix_temp.set_slc(0,b[0] ^ (b[1]^c[1]) ^ c[2] ^ c[3]);
mix_temp.set_slc(8,c[0] ^ b[1] ^ (b[2]^ c[2]) ^ c[3]);
mix_temp.set_slc(16,c[0] ^ c[1] ^ b[2]^ (b[3]^ c[3]));
mix_temp.set_slc(24,(b[0] ^ c[0]) ^ c[1] ^ c[2] ^ b[3]);

mixcolumns_out.write(mix_temp);
```

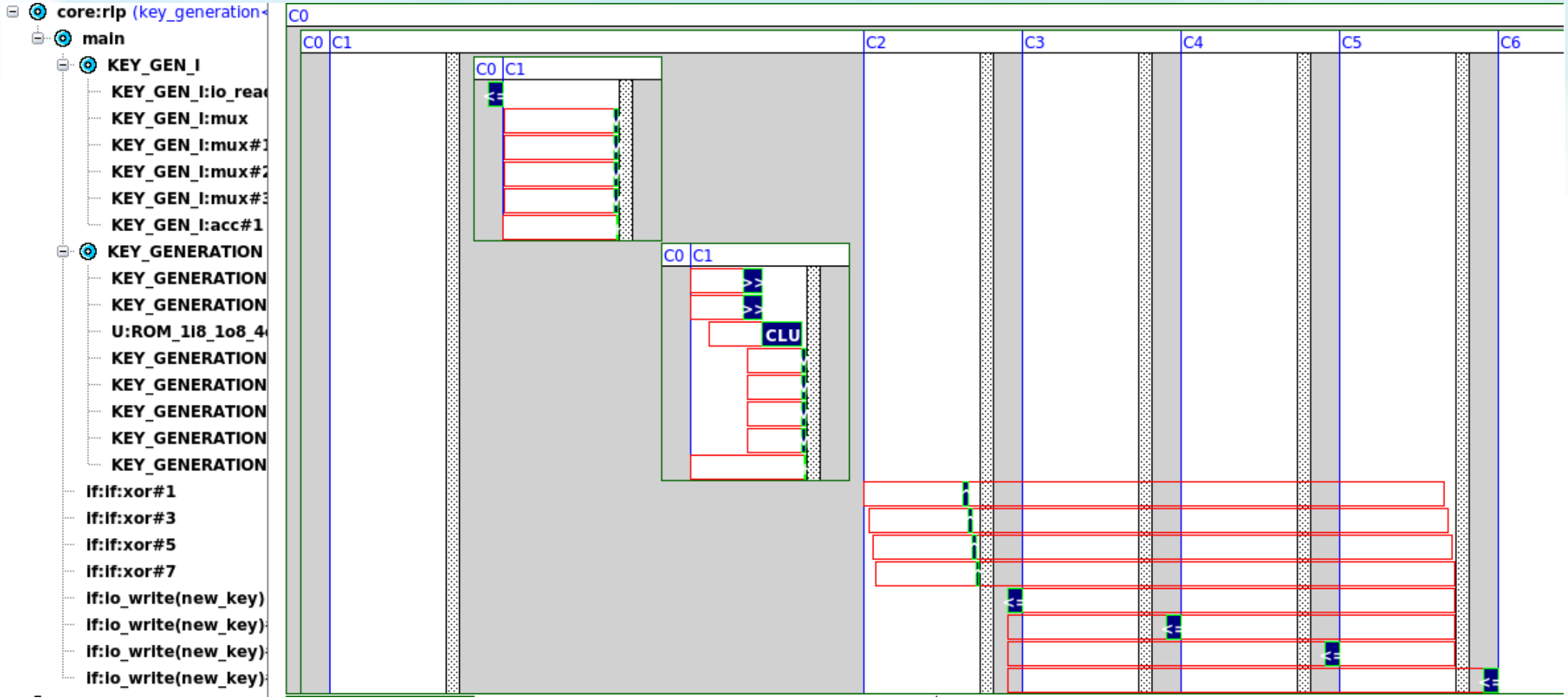
ENCRYPTION BLOCKS: MIX COL





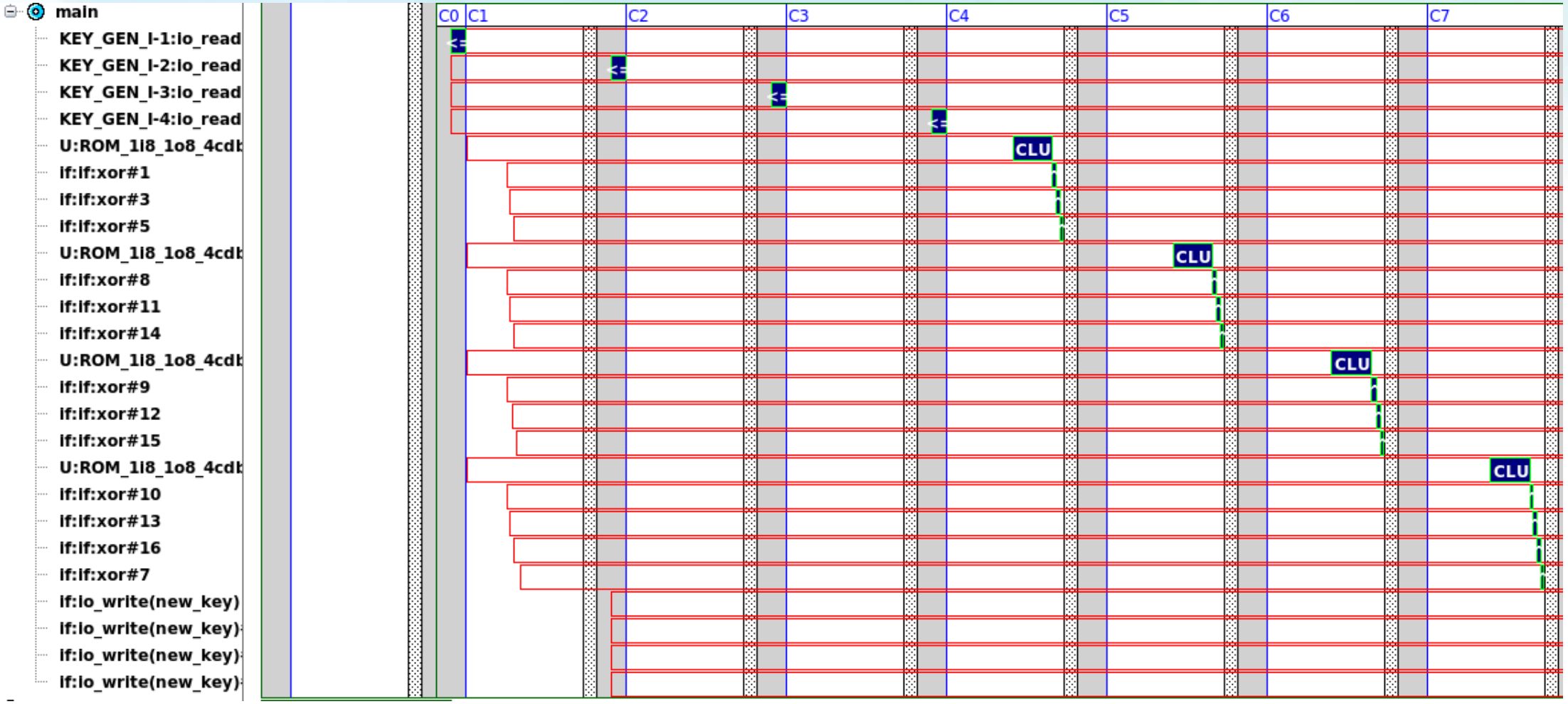
All loops are fully unrolled

ENCRYPTION BLOCKS: KEY GEN

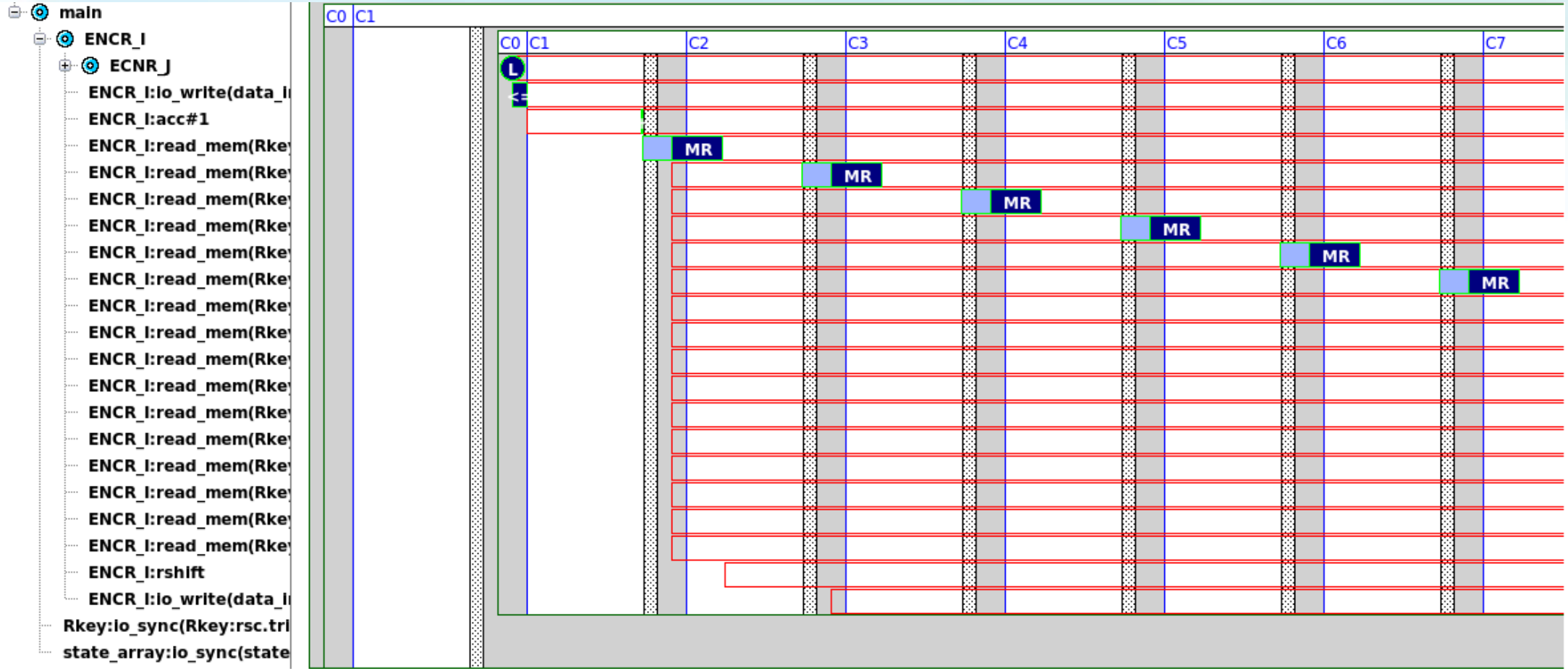


ENCRYPTION BLOCKS: KEY GEN

All loops are fully unrolled



ENCRYPTION BLOCKS: INPUT FEED



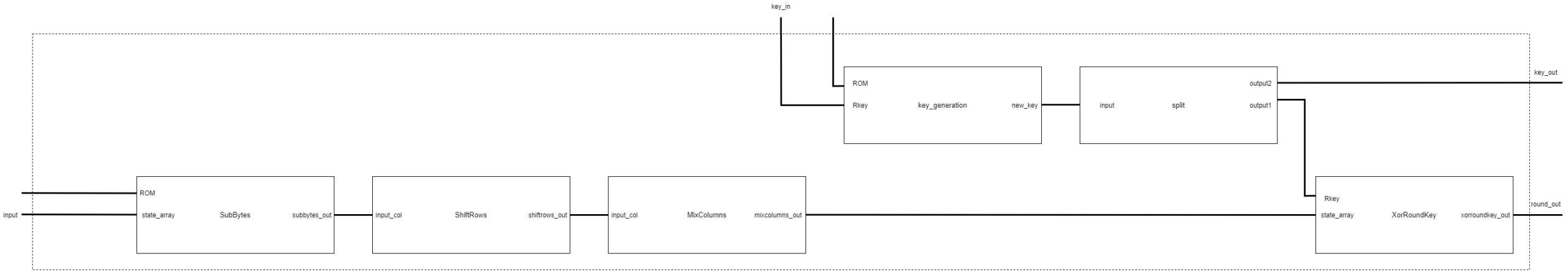
ENCRYPTION BLOCKS: INPUT FEED

`Both loops are fully unrolled`

ENCRYPTION BLOCKS: METRICS

encrypt_hier_block.v5 (extract)	1453	2906.00	106	212.00	116847.54	1.05
encrypt_hier_block.v28 (allocate)	**539**	**1078.00**	**21**	**42.00**	**46434.56**	

```
# SCVerify intercepting C++ function 'encrypt_hier_block' for RTL block 'encrypt_hier_block'
# Info: HW reset: TLS_rst active @ 0 s
# Info: Execution of user-supplied C++ testbench 'main()' has completed with exit code = 0
#
# Info: Collecting data completed
#    captured 5 values of state_array
#    captured 5 values of Rkey
#    captured 20 values of comp_out
# Info: scverify_top/user_tb: Simulation completed
#
# Checking results
# 'comp_out'
#    capture count        = 20
#    comparison count     = 20
#    ignore count         = 0
#    error count          = 0
#    stuck in dut fifo    = 0
#    stuck in golden fifo = 0
#
# Info: scverify_top/user_tb: Simulation PASSED @ 16131 ns
```



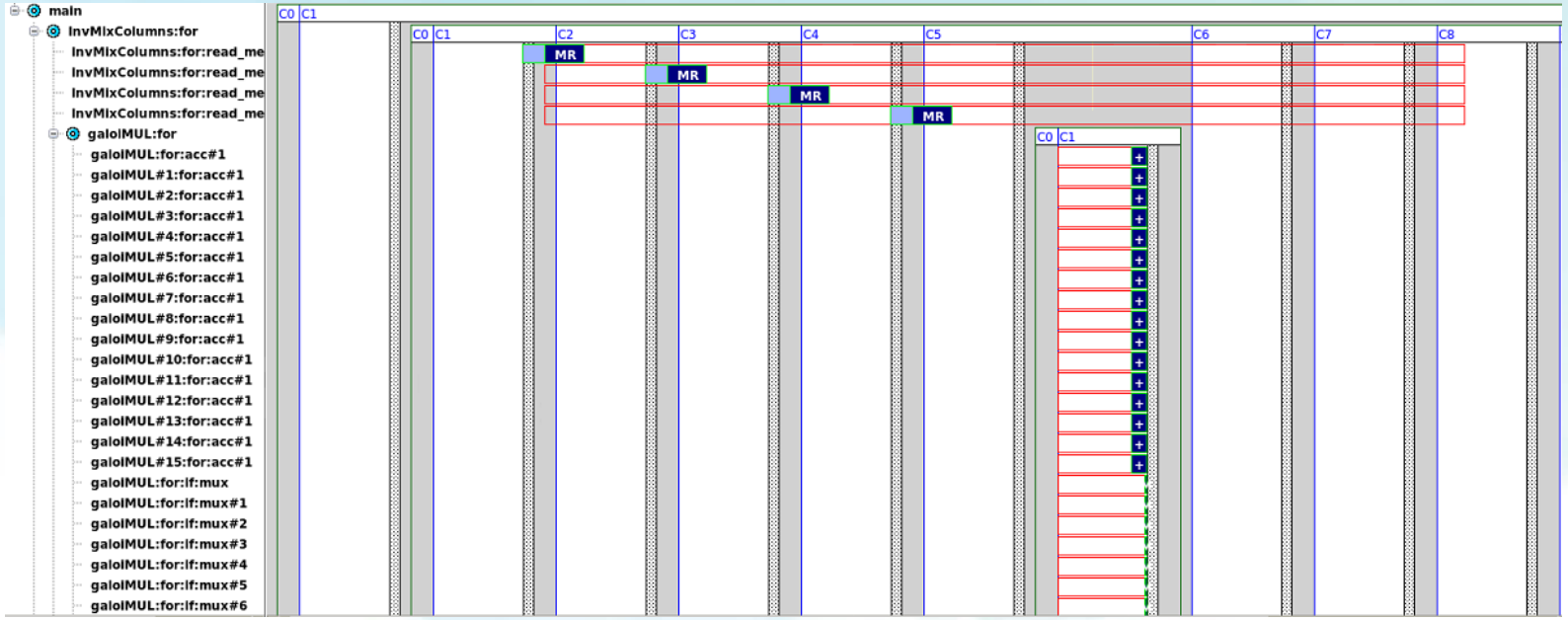
DECRYPTION: MIX COL

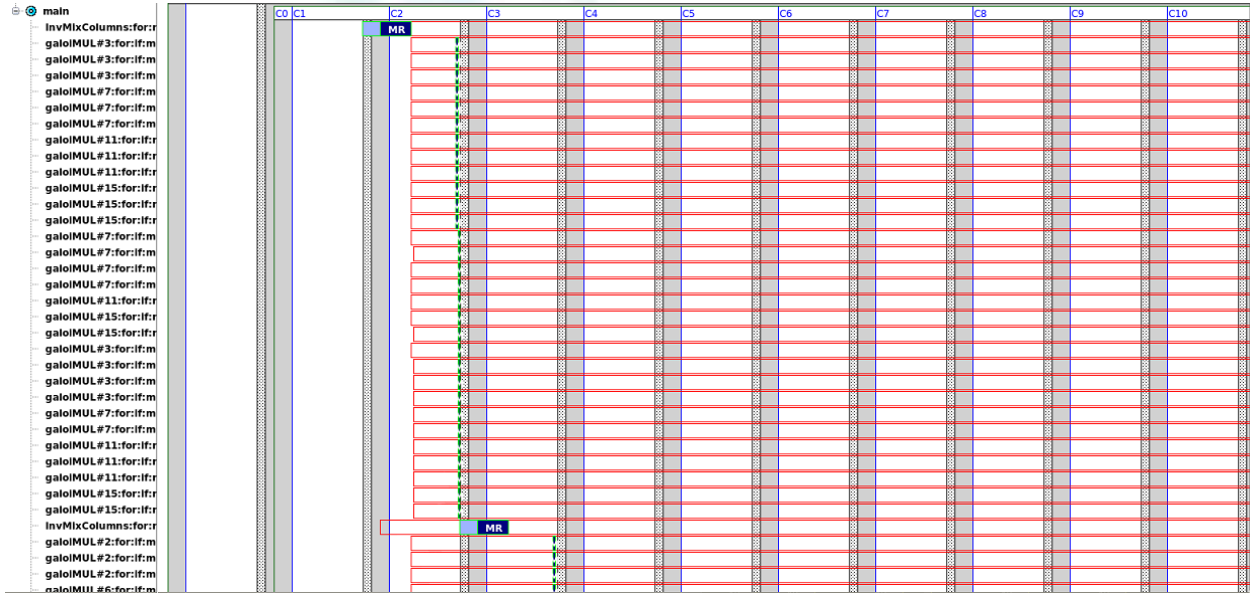
$$\underset{C}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \xleftrightarrow{\text{Inverse}} \underset{C^{-1}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}$$

```
galoiMUL(state_array[0][j],0x0e)
galoiMUL(state_array[0][j],0x09)
galoiMUL(state_array[0][j],0x0d)
galoiMUL(state_array[0][j],0x0b)
```

```
ac_int<8,false> galoiMUL( ac_int<8,false> a,  ac_int<8,false> b){
    ac_int<8,false> res = 0;
    for (int i=0; i<8; i++){
        if(b>=0){
            if(b&1){
                res^=a;
            }
            if(a&0x80){
                a=(a<<1)^0x11B;
            }
            else{
                a<<=1;
            }
            b>>=1;
        }
    }
    return res;
}
```

DECRYPTION: INVMIX COL





Both loops are fully unrolled

DECRYPTION BLOCKS : METRICS

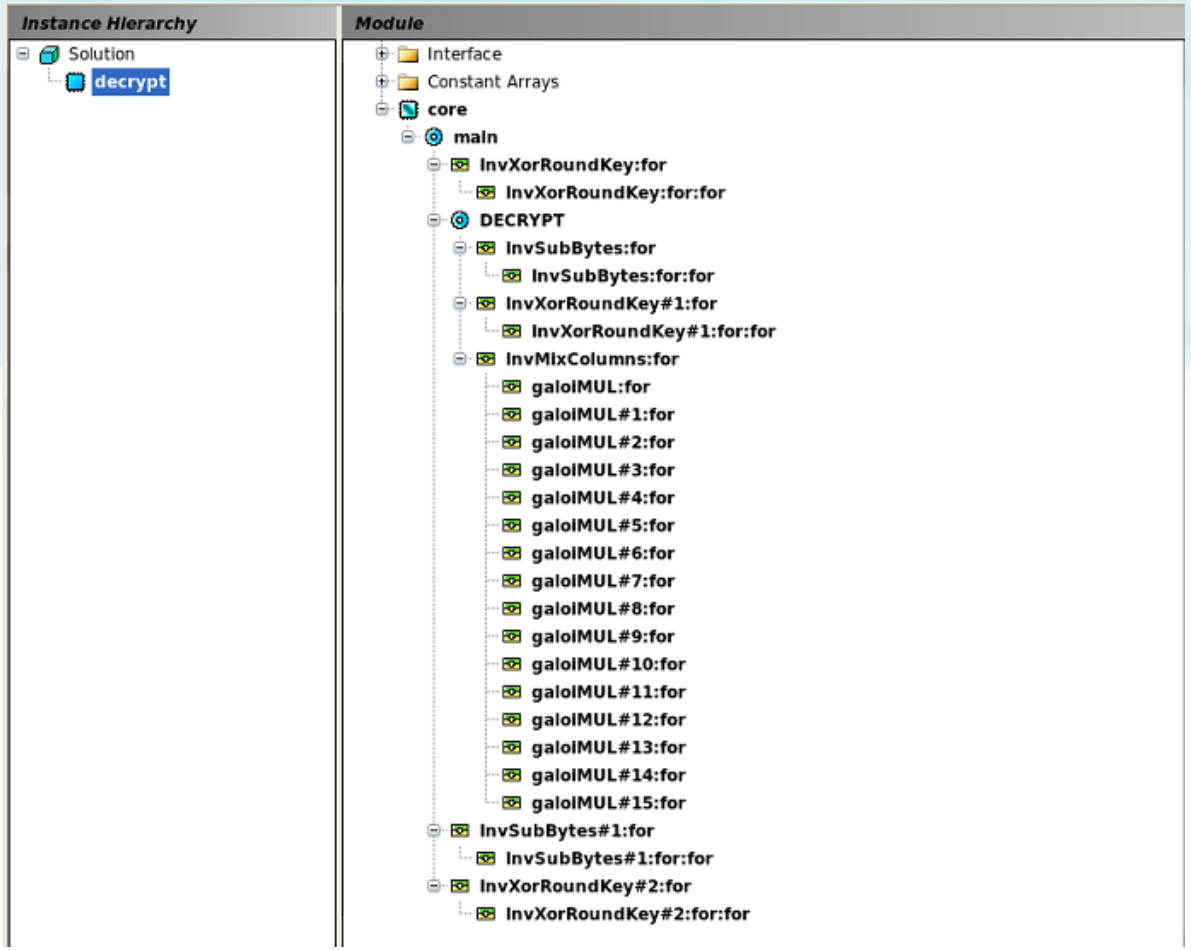
decrypt_hier_block.v2 (allocate)	2481	49620.00	904	18080.00	89217.63	

```
# SCVerify intercepting C++ function 'decrypt_hier_block' for RTL block 'decrypt_hier_block'
# Info: HW reset: TLS_rst active @ 0 s
# Info: Execution of user-supplied C++ testbench 'main()' has completed with exit code = 0
#
# Info: Collecting data completed
#    captured 5 values of state_array
#    captured 5 values of Rkey
#    captured 20 values of comp_out
# Info: scverify_top/user_tb: Simulation completed
#
# Checking results
# 'comp_out'
#    capture count        = 20
#    comparison count     = 20
#    ignore count         = 0
#    error count          = 0
#    stuck in dut fifo    = 0
#    stuck in golden fifo = 0
#
# Info: scverify_top/user_tb: Simulation PASSED @ 19511 ns
```

decrypt_hier_block.v14 (allocate)	738	14760.00	220	4400.00		35133.19

DECRYPTION METRICS

decrypt.v1 (allocate)	4712	9424.00	4717	9434.00	6107.51



decrypt.v9 (allocate)	248	496.00	250	500.00	3067.31