112061576 莊家政
112061603 李柏叡
112061621 賈暐家

ASoC Final Project Report

Introduction:

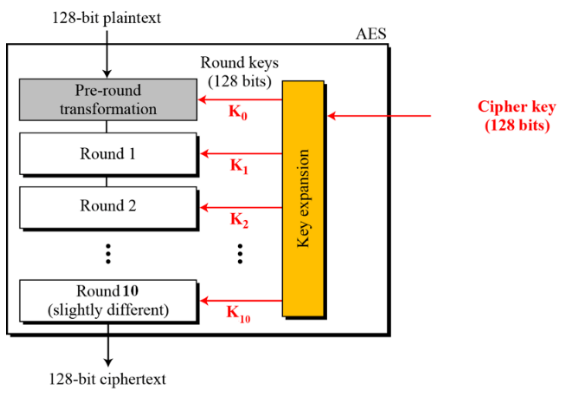
✓ Aes-128 encryption:

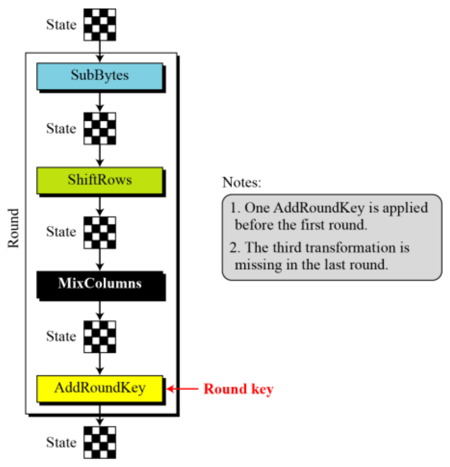
AES-128(Advanced Encryption Standard) is a symmetric encryption algorithm that encrypts data using a fixed-length key (128 bits).

AES-128 can be used to protect the privacy of sensitive data, ensuring security during transmission and storage.

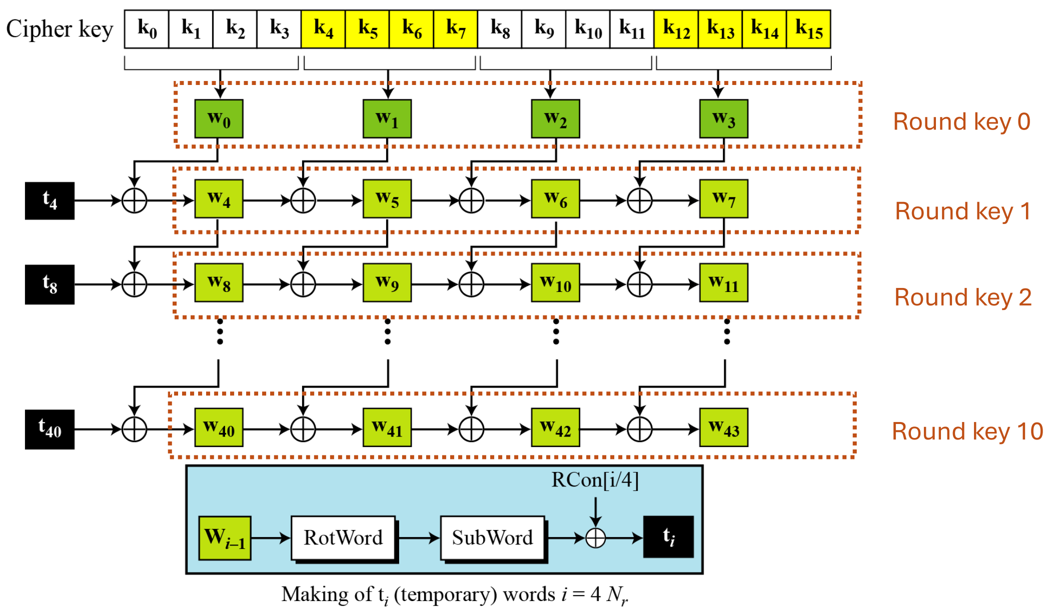
Catapult HLS:

✓ Algorithm:




State
SubBytes
ShiftRows
MixColumns
AddRoundKey
Round
Round key
Notes:
1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

Key expansion function:

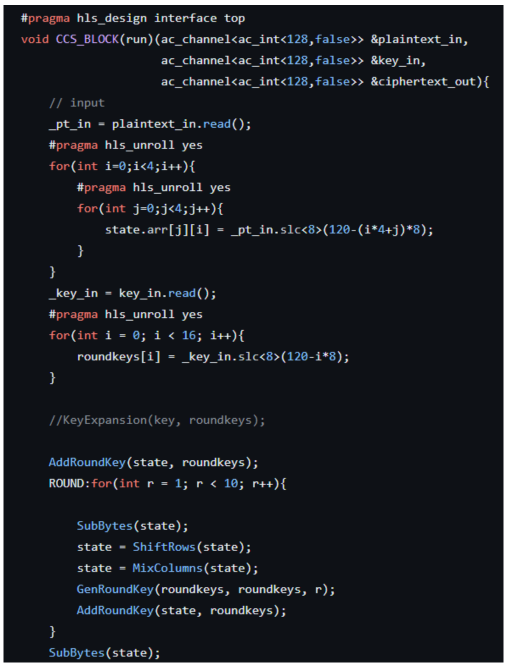


Cipher key
k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15}
w_0 w_1 w_2 w_3
Round key 0
t_4
w_4 w_5 w_6 w_7
Round key 1
t_8
w_8 w_9 w_{10} w_{11}
Round key 2
t_{40}
w_{40} w_{41} w_{42} w_{43}
Round key 10
RCon[i/4]
W_{i-1}
RotWord
SubWord
t_i

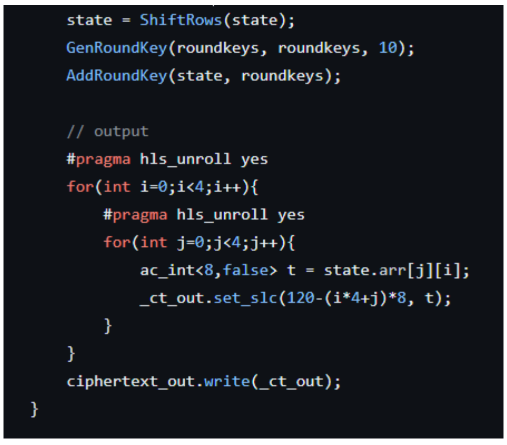
Making of t_i (temporary) words $i = 4\,N_r$

Not only is the final output ciphertext verified, but the intermediate states between each round are also checked using the test value provided by NIST.

The key is generated by **GenRoundKey**. You can see that a state array is first declared to store the plaintext, and then the array is subjected to a series of operations such as **SubBytes**, **ShiftRows**, **MixColumns**, and **AddRoundKey** with keys. After repeating 10 times, the ciphertext is obtained.



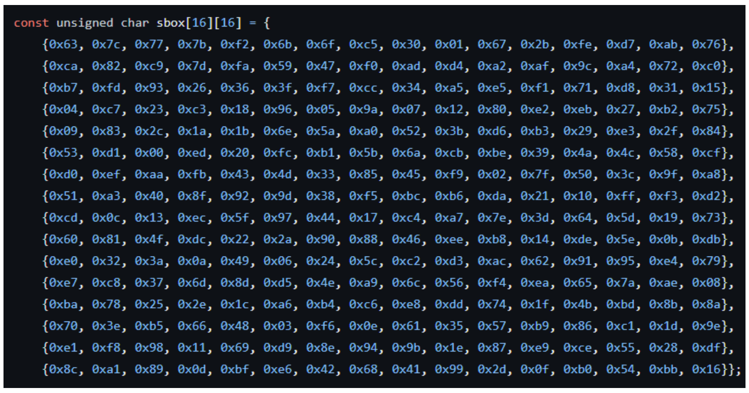
```
#pragma hls_design interface top
void CCS_BLOCK(run)(ac_channel<ac_int<128,false>> &plaintext_in,
                    ac_channel<ac_int<128,false>> &key_in,
                    ac_channel<ac_int<128,false>> &ciphertext_out){
    // input
    _pt_in = plaintext_in.read();
    #pragma hls_unroll yes
    for(int i=0;i<4;i++){
        #pragma hls_unroll yes
        for(int j=0;j<4;j++){
            state.arr[j][i] = _pt_in.slc<8>(120-(i*4+j)*8);
        }
    }
    _key_in = key_in.read();
    #pragma hls_unroll yes
    for(int i = 0; i < 16; i++){
        roundkeys[i] = _key_in.slc<8>(120-i*8);
    }

    //KeyExpansion(key, roundkeys);

    AddRoundKey(state, roundkeys);
    ROUND:for(int r = 1; r < 10; r++){

        SubBytes(state);
        state = ShiftRows(state);
        state = MixColumns(state);
        GenRoundKey(roundkeys, roundkeys, r);
        AddRoundKey(state, roundkeys);
    }
    SubBytes(state);
```



```
    state = ShiftRows(state);
    GenRoundKey(roundkeys, roundkeys, 10);
    AddRoundKey(state, roundkeys);

    // output
    #pragma hls_unroll yes
    for(int i=0;i<4;i++){
        #pragma hls_unroll yes
        for(int j=0;j<4;j++){
            ac_int<8,false> t = state.arr[j][i];
            _ct_out.set_slc(120-(i*4+j)*8, t);
        }
    }
    ciphertext_out.write(_ct_out);
}
```

- ✓ SubBytes:

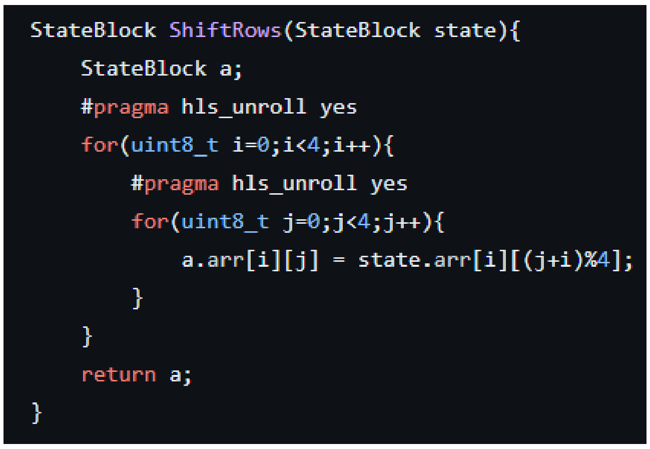
  Substitute each byte according to the lookup table.



```
const unsigned char sbox[16][16] = {
    {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}};
```

- ✓ ShiftRows:

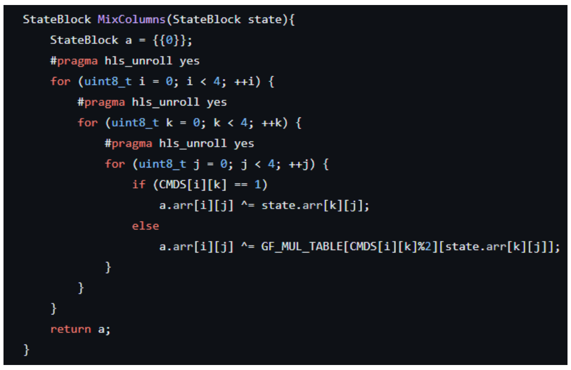
  The nth column of array shifts n bytes.



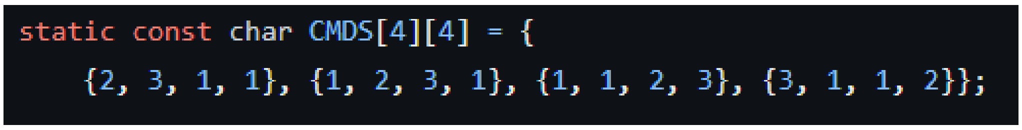
```
StateBlock ShiftRows(StateBlock state){
    StateBlock a;
    #pragma hls_unroll yes
    for(uint8_t i=0;i<4;i++){
        #pragma hls_unroll yes
        for(uint8_t j=0;j<4;j++){
            a.arr[i][j] = state.arr[i][(j+i)%4];
        }
    }
    return a;
}
```

- ✓ MixColumns:

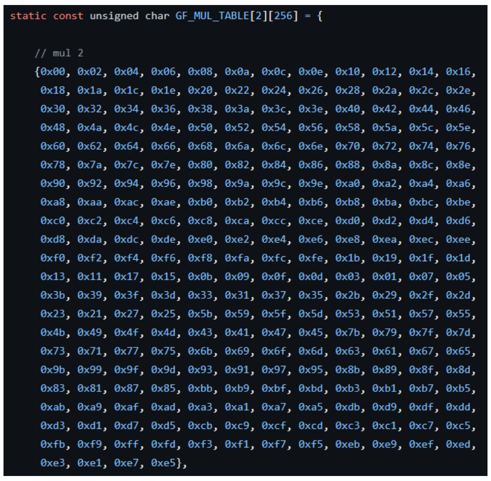
  We use the Galois Field table we provide to implement.



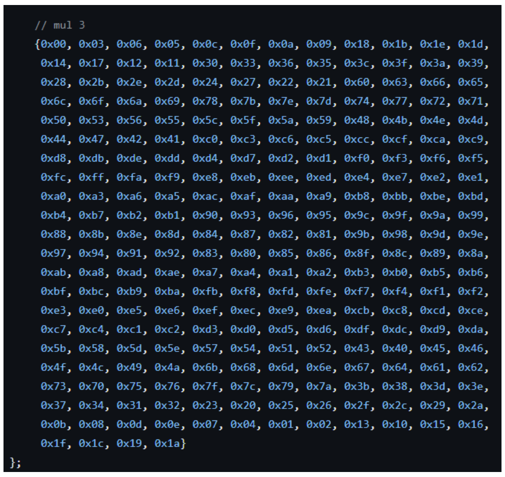
```
StateBlock MixColumns(StateBlock state){
    StateBlock a = {{0}};
    #pragma hls_unroll yes
    for (uint8_t i = 0; i < 4; ++i) {
        #pragma hls_unroll yes
        for (uint8_t k = 0; k < 4; ++k) {
            #pragma hls_unroll yes
            for (uint8_t j = 0; j < 4; ++j) {
                if (CMDS[i][k] == 1)
                    a.arr[i][j] ^= state.arr[k][j];
                else
                    a.arr[i][j] ^= GF_MUL_TABLE[CMDS[i][k]%2][state.arr[k][j]];
            }
        }
    }
    return a;
}
```



```c
static const char CMDS[4][4] = {
    {2, 3, 1, 1}, {1, 2, 3, 1}, {1, 1, 2, 3}, {3, 1, 1, 2}};
```



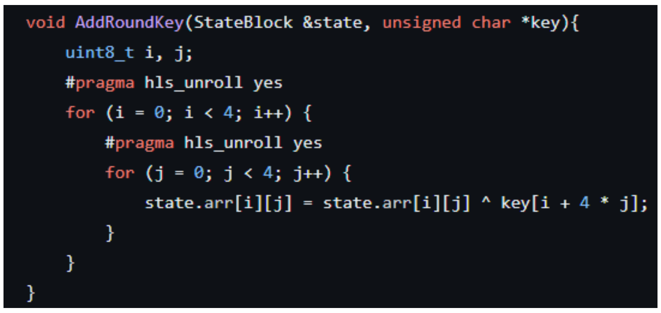
```c
static const unsigned char GF_MUL_TABLE[2][256] = {

    // mul 2
    {0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16,
     0x18, 0x1a, 0x1c, 0x1e, 0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e,
     0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e, 0x40, 0x42, 0x44, 0x46,
     0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,
     0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76,
     0x78, 0x7a, 0x7c, 0x7e, 0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e,
     0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e, 0xa0, 0xa2, 0xa4, 0xa6,
     0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,
     0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6,
     0xd8, 0xda, 0xdc, 0xde, 0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee,
     0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe, 0x1b, 0x19, 0x1f, 0x1d,
     0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d, 0x03, 0x01, 0x07, 0x05,
     0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d,
     0x23, 0x21, 0x27, 0x25, 0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55,
     0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45, 0x7b, 0x79, 0x7f, 0x7d,
     0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,
     0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d,
     0x83, 0x81, 0x87, 0x85, 0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5,
     0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5, 0xdb, 0xd9, 0xdf, 0xdd,
     0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
     0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed,
     0xe3, 0xe1, 0xe7, 0xe5},
```



```c
    // mul 3
    {0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d,
     0x14, 0x17, 0x12, 0x11, 0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39,
     0x28, 0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21, 0x60, 0x63, 0x66, 0x65,
     0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71,
     0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d,
     0x44, 0x47, 0x42, 0x41, 0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9,
     0xd8, 0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1, 0xf0, 0xf3, 0xf6, 0xf5,
     0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1,
     0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd,
     0xb4, 0xb7, 0xb2, 0xb1, 0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99,
     0x88, 0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81, 0x9b, 0x98, 0x9d, 0x9e,
     0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a,
     0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6,
     0xbf, 0xbc, 0xb9, 0xba, 0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2,
     0xe3, 0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xe9, 0xea, 0xcb, 0xc8, 0xcd, 0xce,
     0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda,
     0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46,
     0x4f, 0x4c, 0x49, 0x4a, 0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62,
     0x73, 0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a, 0x3b, 0x38, 0x3d, 0x3e,
     0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a,
     0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16,
     0x1f, 0x1c, 0x19, 0x1a}
};
```

- ✓ AddRoundKey:

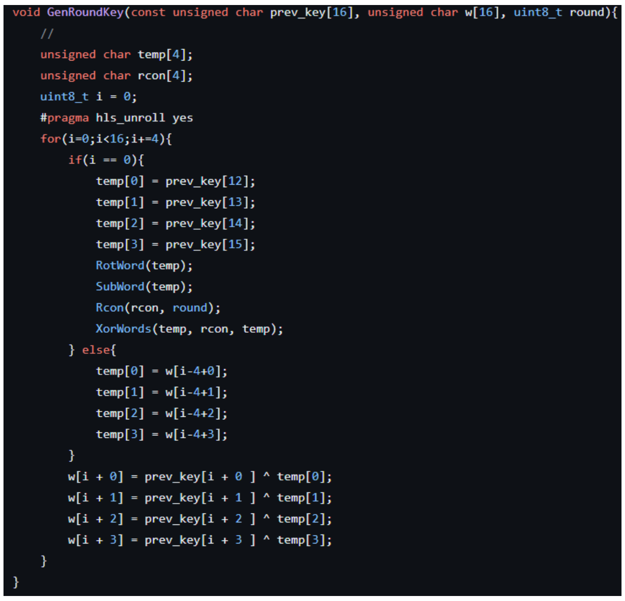
  We use key and state to calculate the new state.



```c
void AddRoundKey(StateBlock &state, unsigned char *key){
    uint8_t i, j;
    #pragma hls_unroll yes
    for (i = 0; i < 4; i++) {
        #pragma hls_unroll yes
        for (j = 0; j < 4; j++) {
            state.arr[i][j] = state.arr[i][j] ^ key[i + 4 * j];
        }
    }
}
```

✓ GenRoundKey:

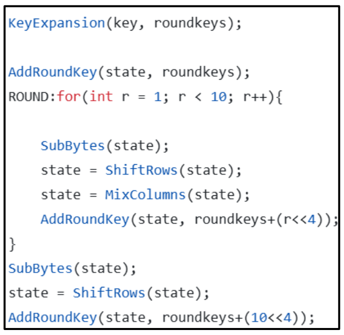
Here is key expansion. Use this function to generate the 10 keys we need.



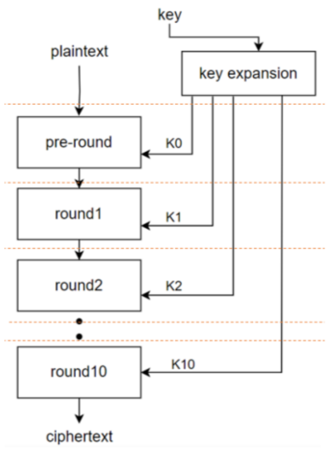
```
void GenRoundKey(const unsigned char prev_key[16], unsigned char w[16], uint8_t round){
    //
    unsigned char temp[4];
    unsigned char rcon[4];
    uint8_t i = 0;
    #pragma hls_unroll yes
    for(i=0;i<16;i+=4){
        if(i == 0){
            temp[0] = prev_key[12];
            temp[1] = prev_key[13];
            temp[2] = prev_key[14];
            temp[3] = prev_key[15];
            RotWord(temp);
            SubWord(temp);
            Rcon(rcon, round);
            XorWords(temp, rcon, temp);
        } else{
            temp[0] = w[i-4+0];
            temp[1] = w[i-4+1];
            temp[2] = w[i-4+2];
            temp[3] = w[i-4+3];
        }
        w[i + 0] = prev_key[i + 0 ] ^ temp[0];
        w[i + 1] = prev_key[i + 1 ] ^ temp[1];
        w[i + 2] = prev_key[i + 2 ] ^ temp[2];
        w[i + 3] = prev_key[i + 3 ] ^ temp[3];
    }
}
```

✓ Micro-architecture exploration:

Original:



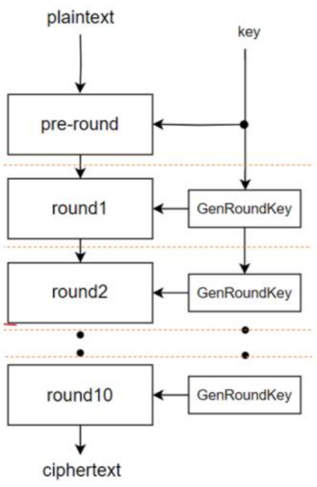
```
KeyExpansion(key, roundkeys);

AddRoundKey(state, roundkeys);
ROUND:for(int r = 1; r < 10; r++){

    SubBytes(state);
    state = ShiftRows(state);
    state = MixColumns(state);
    AddRoundKey(state, roundkeys+(r<<4));
}
SubBytes(state);
state = ShiftRows(state);
AddRoundKey(state, roundkeys+(10<<4));
```



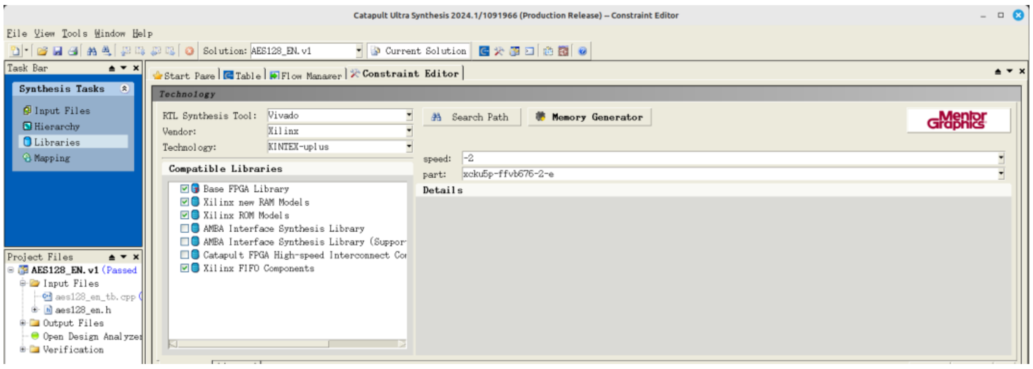
Since there’s no need to generate all round keys before the first round, and each round key can be generated using the round key of the previous round. We decided to decompose the key expansion function into several GenRoundKey functions.

V2:

```
AddRoundKey(state, roundkeys);
ROUND:for(int r = 1; r < 10; r++){

    SubBytes(state);
    state = ShiftRows(state);
    state = MixColumns(state);
    GenRoundKey(roundkeys, roundkeys, r);
    AddRoundKey(state, roundkeys);
}
SubBytes(state);
state = ShiftRows(state);
GenRoundKey(roundkeys, roundkeys, 10);
AddRoundKey(state, roundkeys);
```



✓ Result:


Catapult Ultra Synthesis 2024.1/1091966 (Production Release) – Constraint Editor

Solution: AES128_EN.v1

Technology

RTL Synthesis Tool: Vivado
Vendor: Xilinx
Technology: KINTEX-uplus

speed: -2
part: xcku5p-ffvb676-2-e

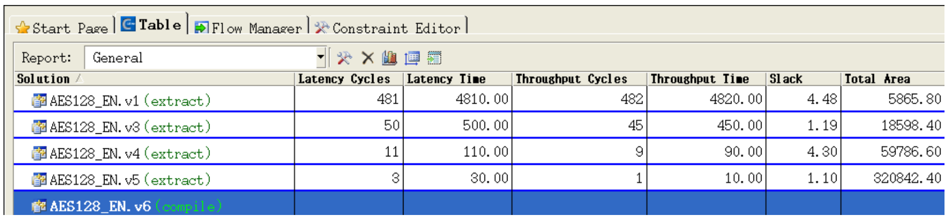
Compatible Libraries

- ☑ Base FPGA Library
- ☑ Xilinx new RAM Models
- ☑ Xilinx ROM Models
- ☐ AMBA Interface Synthesis Library
- ☐ AMBA Interface Synthesis Library (Suppor
- ☐ Catapult FPGA High-speed Interconnect Co
- ☑ Xilinx FIFO Components

Project Files: AES128_EN.v1 (Passed; Input Files: aes128_en_tb.cpp, aes128_en.h; Output Files; Open Design Analyzer; Verification

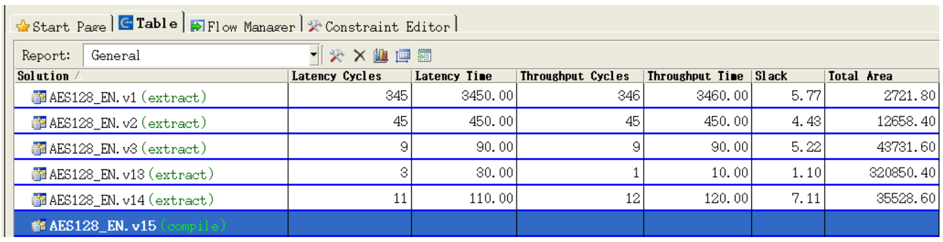
Synthesizing the design using the Xilinx library, we get the following result.

Original:


Start Page
Table
Flow Manager
Constraint Editor
Report: General

Solution /	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
AES128_EN.v1 (extract)	481	4810.00	482	4820.00	4.48	5865.80
AES128_EN.v3 (extract)	50	500.00	45	450.00	1.19	18598.40
AES128_EN.v4 (extract)	11	110.00	9	90.00	4.30	59786.60
AES128_EN.v5 (extract)	3	30.00	1	10.00	1.10	320842.40
AES128_EN.v6 (compile)						

V2:



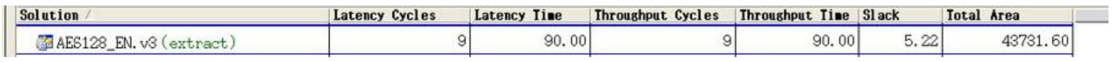
Report: General

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
AES128_EN.v1 (extract)	345	3450.00	346	3460.00	5.77	2721.80
AES128_EN.v2 (extract)	45	450.00	45	450.00	4.43	12658.40
AES128_EN.v3 (extract)	9	90.00	9	90.00	5.22	43731.60
AES128_EN.v13 (extract)	3	30.00	1	10.00	1.10	320850.40
AES128_EN.v14 (extract)	11	110.00	12	120.00	7.11	35528.60
AES128_EN.v15 (compile)						

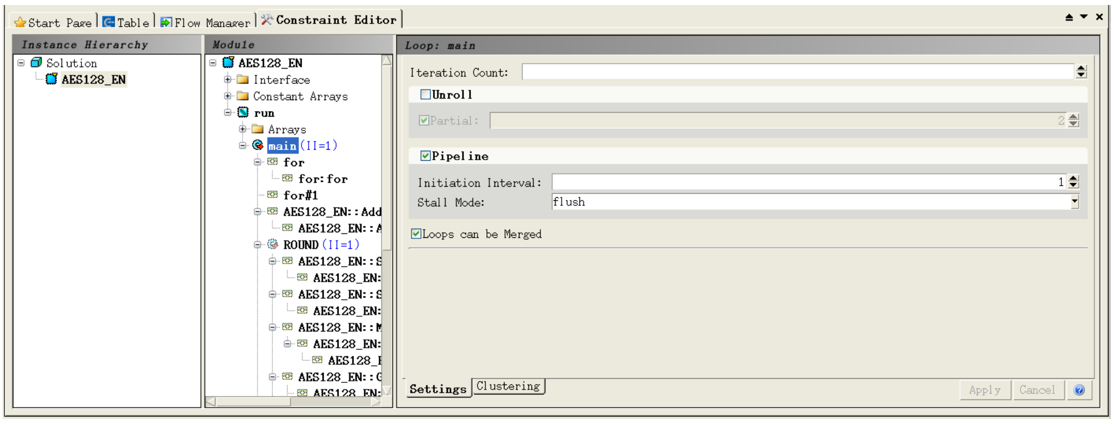
From the screenshot above, we can see that the area generated using the v2 design is much smaller than using the original design.

✓ Problems we've encountered with Catapult:

1. Catapult GUI crashed several times when I was trying out different pipeline and unroll settings.
2. Pipeline setting doesn't work as expected.



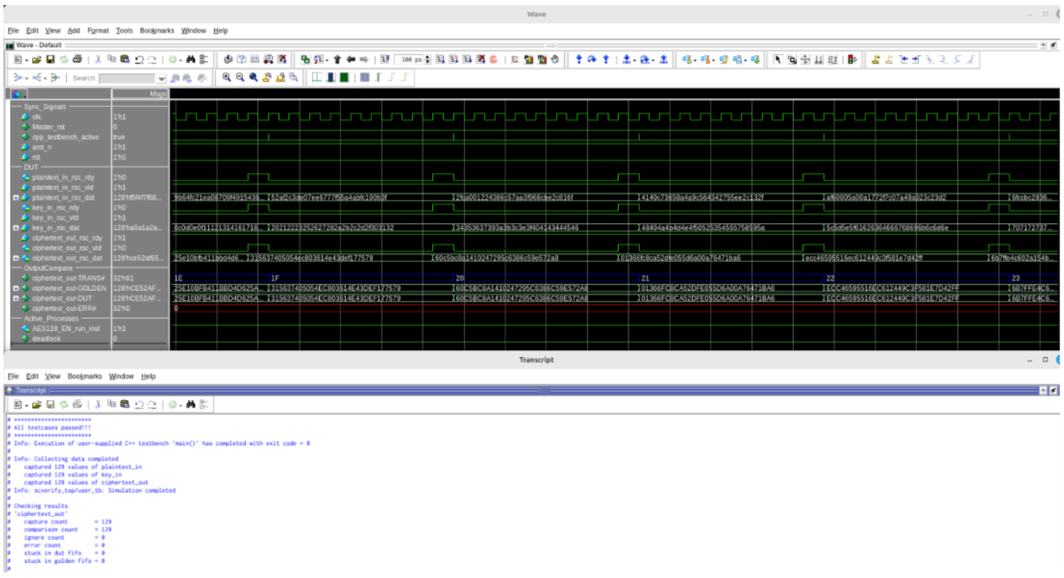
Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
AES128_EN.v3 (extract)	9	90.00	9	90.00	5.22	43731.60



Loop: main

Iteration Count:
☐ Unroll
Partial: 2
☑ Pipeline
Initiation Interval: 1
Stall Mode: flush
☑ Loops can be Merged

✓ Scverify:

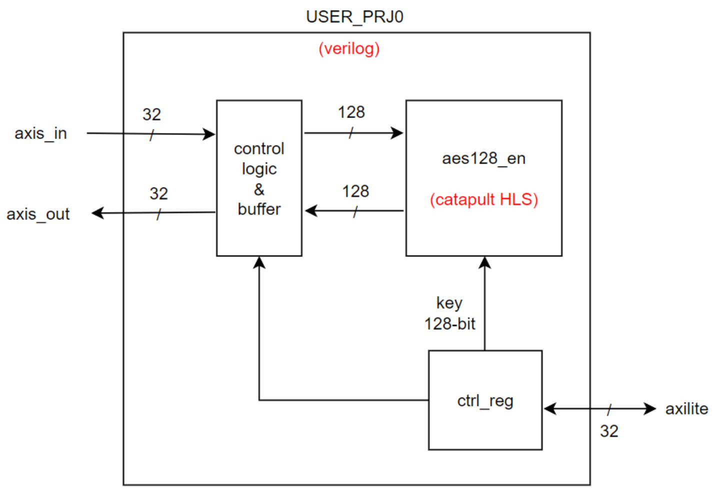


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Sync_Signals
clk 1'h1
Master_rst 0
cpp_testbench_active true
arst_n 1'h1
rst 1'h0
DUT
plaintext_in_rsc_rdy 1'h0
plaintext_in_rsc_vld 1'h1
plaintext_in_rsc_dat 128'hf5f4f7f68...
key_in_rsc_rdy 1'h0
key_in_rsc_vld 1'h1
key_in_rsc_dat 128'ha0a1a2a...
ciphertext_out_rsc_rdy 1'h1
ciphertext_out_rsc_vld 1'h0
ciphertext_out_rsc_dat 128'hce52af65...
OutputCompare
ciphertext_out-TRANS# 32'h81
ciphertext_out-GOLDEN 128'hCE52AF...
ciphertext_out-DUT 128'hCE52AF...
ciphertext_out-ERR# 32'h0
Active_Processes
AES128_EN_run_inst 1'h1
deadlock 0
Transcript
File Edit View Bookmarks Window Help

```
# **********************
# All testcases passed!!!
# **********************
# Info: Execution of user-supplied C++ testbench 'main()' has completed with exit code = 0
#
# Info: Collecting data completed
#    captured 129 values of plaintext_in
#    captured 129 values of key_in
#    captured 129 values of ciphertext_out
# Info: scverify_top/user_tb: Simulation completed
#
# Checking results
# 'ciphertext_out'
#    capture count        = 129
#    comparison count     = 129
#    ignore count         = 0
#    error count          = 0
#    stuck in dut fifo    = 0
#    stuck in golden fifo = 0
#
```

Integrate into FSIC:

✓ Block diagram:



The input and output of our aes128_en is 128-bit, but the width of axi-stream in FSIC is only 32-bit. Therefore, a control logic is added to deal with the problem.

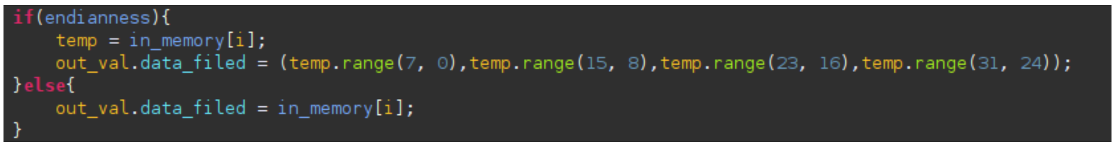
✓ Control register map:

Register	WordOffset	ResetVal	Access	Description
Reg_rst	0	0	R/W	1:reset (this won't reset data length and keys)
Done	1	0	R-O	1:done
Data_length	2	0	R/W	Data length in unit of blocks (16bytes)
Key0	4	0	R/W	Key_in{[0],[1],[2],[3]}
Key1	5	0	R/W	Key_in{[4],[5],[6],[7]}
Key2	6	0	R/W	Key_in{[8],[9],[10],[11]}
Key3	7	0	R/W	Key_in{[12],[13],[14],[15]}
clk_cnt_low	8	0	R-O	(only for fpga validation) Count how many clock cycles elapsed from receiving the first plaintext to sending the last ciphertext
Clk_cnt_high	9	0	R-O	

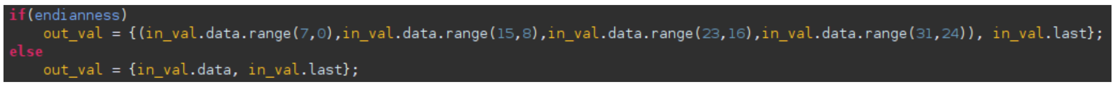
Caravel-FSIC FPGA simulation:

✓ USER DMA:

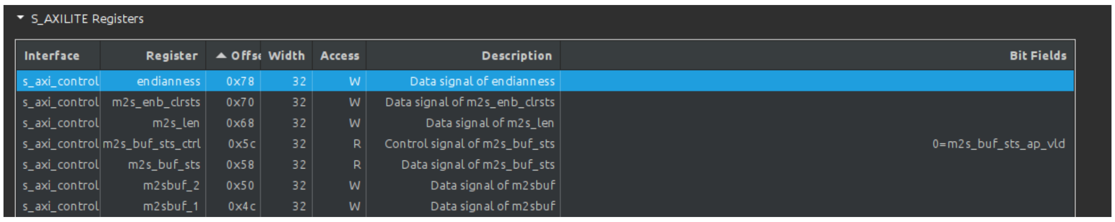
Our application accelerator is big endian, but the rest of the system is little endian. We added a register in user dma to control the conversion.



```
if(endianness){
    temp = in_memory[i];
    out_val.data_filed = (temp.range(7, 0),temp.range(15, 8),temp.range(23, 16),temp.range(31, 24));
}else{
    out_val.data_filed = in_memory[i];
}
```



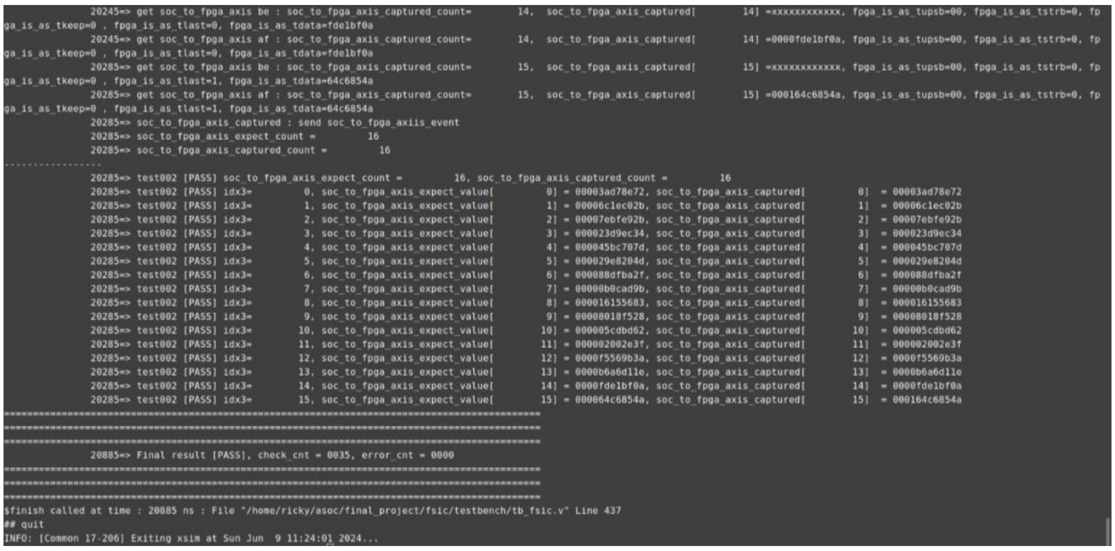
```
if(endianness)
    out_val = {(in_val.data.range(7,0),in_val.data.range(15,8),in_val.data.range(23,16),in_val.data.range(31,24)), in_val.last};
else
    out_val = {in_val.data, in_val.last};
```



S_AXILITE Registers

Interface	Register	Offse	Width	Access	Description	Bit Fields
s_axi_control	endianness	0x78	32	W	Data signal of endianness	
s_axi_control	m2s_enb_clrsts	0x70	32	W	Data signal of m2s_enb_clrsts	
s_axi_control	m2s_len	0x68	32	W	Data signal of m2s_len	
s_axi_control	m2s_buf_sts_ctrl	0x5c	32	R	Control signal of m2s_buf_sts	0=m2s_buf_sts_ap_vld
s_axi_control	m2s_buf_sts	0x58	32	R	Data signal of m2s_buf_sts	
s_axi_control	m2sbuf_2	0x50	32	W	Data signal of m2sbuf	
s_axi_control	m2sbuf_1	0x4c	32	W	Data signal of m2sbuf	

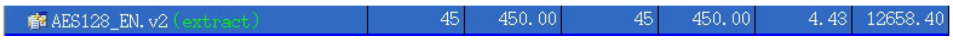
✓ Result:



```
              20245=> get soc_to_fpga_axis be : soc_to_fpga_axis_captured_count=        14,  soc_to_fpga_axis_captured[        14] =xxxxxxxxxxxx, fpga_is_as_tupsb=00, fpga_is_as_tstrb=0, fp
ga_is_as_tkeep=0 , fpga_is_as_tlast=0, fpga_is_as_tdata=fde1bf0a
              20245=> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=        14,  soc_to_fpga_axis_captured[        14] =0000fde1bf0a, fpga_is_as_tupsb=00, fpga_is_as_tstrb=0, fp
ga_is_as_tkeep=0 , fpga_is_as_tlast=0, fpga_is_as_tdata=fde1bf0a
              20285=> get soc_to_fpga_axis be : soc_to_fpga_axis_captured_count=        15,  soc_to_fpga_axis_captured[        15] =xxxxxxxxxxxx, fpga_is_as_tupsb=00, fpga_is_as_tstrb=0, fp
ga_is_as_tkeep=0 , fpga_is_as_tlast=1, fpga_is_as_tdata=64c6854a
              20285=> get soc_to_fpga_axis af : soc_to_fpga_axis_captured_count=        15,  soc_to_fpga_axis_captured[        15] =000164c6854a, fpga_is_as_tupsb=00, fpga_is_as_tstrb=0, fp
ga_is_as_tkeep=0 , fpga_is_as_tlast=1, fpga_is_as_tdata=64c6854a
              20285=> soc_to_fpga_axis_captured : send soc_to_fpga_axiis_event
              20285=> soc_to_fpga_axis_expect_count =         16
              20285=> soc_to_fpga_axis_captured_count =         16
----------------
              20285=> test002 [PASS] soc_to_fpga_axis_expect_count =         16, soc_to_fpga_axis_captured_count =         16
              20285=> test002 [PASS] idx3=         0, soc_to_fpga_axis_expect_value[         0] = 00003ad78e72, soc_to_fpga_axis_captured[         0]  = 00003ad78e72
              20285=> test002 [PASS] idx3=         1, soc_to_fpga_axis_expect_value[         1] = 00006c1ec02b, soc_to_fpga_axis_captured[         1]  = 00006c1ec02b
              20285=> test002 [PASS] idx3=         2, soc_to_fpga_axis_expect_value[         2] = 00007ebfe92b, soc_to_fpga_axis_captured[         2]  = 00007ebfe92b
              20285=> test002 [PASS] idx3=         3, soc_to_fpga_axis_expect_value[         3] = 000023d9ec34, soc_to_fpga_axis_captured[         3]  = 000023d9ec34
              20285=> test002 [PASS] idx3=         4, soc_to_fpga_axis_expect_value[         4] = 000045bc707d, soc_to_fpga_axis_captured[         4]  = 000045bc707d
              20285=> test002 [PASS] idx3=         5, soc_to_fpga_axis_expect_value[         5] = 000029e8204d, soc_to_fpga_axis_captured[         5]  = 000029e8204d
              20285=> test002 [PASS] idx3=         6, soc_to_fpga_axis_expect_value[         6] = 000088dfba2f, soc_to_fpga_axis_captured[         6]  = 000088dfba2f
              20285=> test002 [PASS] idx3=         7, soc_to_fpga_axis_expect_value[         7] = 00000b0cad9b, soc_to_fpga_axis_captured[         7]  = 00000b0cad9b
              20285=> test002 [PASS] idx3=         8, soc_to_fpga_axis_expect_value[         8] = 000016155683, soc_to_fpga_axis_captured[         8]  = 000016155683
              20285=> test002 [PASS] idx3=         9, soc_to_fpga_axis_expect_value[         9] = 00008018f528, soc_to_fpga_axis_captured[         9]  = 00008018f528
              20285=> test002 [PASS] idx3=        10, soc_to_fpga_axis_expect_value[        10] = 000005cdbd62, soc_to_fpga_axis_captured[        10]  = 000005cdbd62
              20285=> test002 [PASS] idx3=        11, soc_to_fpga_axis_expect_value[        11] = 000002002e3f, soc_to_fpga_axis_captured[        11]  = 000002002e3f
              20285=> test002 [PASS] idx3=        12, soc_to_fpga_axis_expect_value[        12] = 0000f5569b3a, soc_to_fpga_axis_captured[        12]  = 0000f5569b3a
              20285=> test002 [PASS] idx3=        13, soc_to_fpga_axis_expect_value[        13] = 0000b6a6d11e, soc_to_fpga_axis_captured[        13]  = 0000b6a6d11e
              20285=> test002 [PASS] idx3=        14, soc_to_fpga_axis_expect_value[        14] = 0000fde1bf0a, soc_to_fpga_axis_captured[        14]  = 0000fde1bf0a
              20285=> test002 [PASS] idx3=        15, soc_to_fpga_axis_expect_value[        15] = 000064c6854a, soc_to_fpga_axis_captured[        15]  = 000164c6854a
=============================================================================================
=============================================================================================
=============================================================================================
              20885=> Final result [PASS], check_cnt = 0035, error_cnt = 0000
=============================================================================================
=============================================================================================
=============================================================================================
$finish called at time : 20885 ns : File "/home/ricky/asoc/final_project/fsic/testbench/tb_fsic.v" Line 437
## quit
INFO: [Common 17-206] Exiting xsim at Sun Jun  9 11:24:01 2024...
```

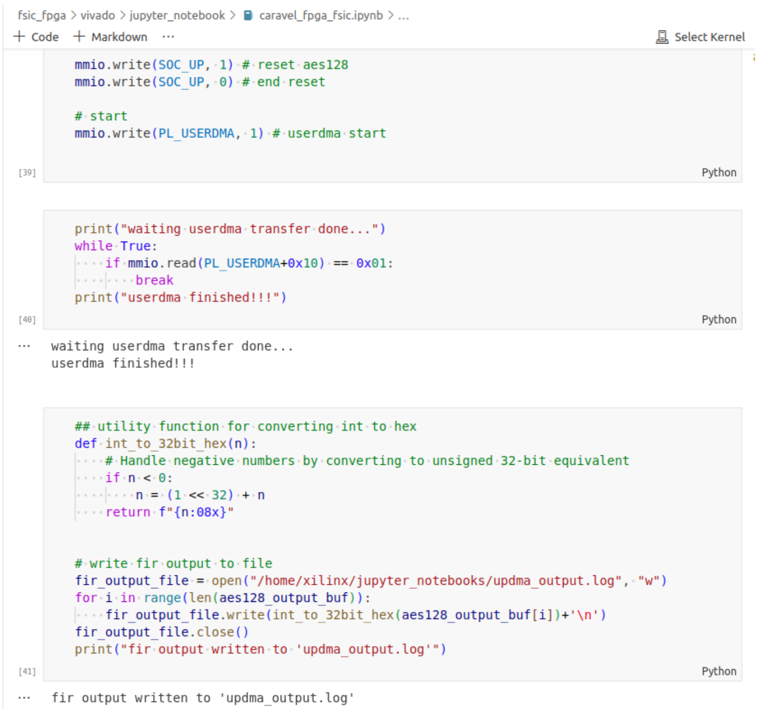
Caravel-FSIC FPGA validation:

✓ Using:



AES128_EN.v2 (extract)	45	450.00	45	450.00	4.43	12658.40

✓ Result:



fsic_fpga > vivado > jupyter_notebook > caravel_fpga_fsic.ipynb > ...

+ Code + Markdown ··· Select Kernel

```python
mmio.write(SOC_UP, 1) # reset aes128
mmio.write(SOC_UP, 0) # end reset

# start
mmio.write(PL_USERDMA, 1) # userdma start
```

[39] Python

```python
print("waiting userdma transfer done...")
while True:
    if mmio.read(PL_USERDMA+0x10) == 0x01:
        break
print("userdma finished!!!")
```

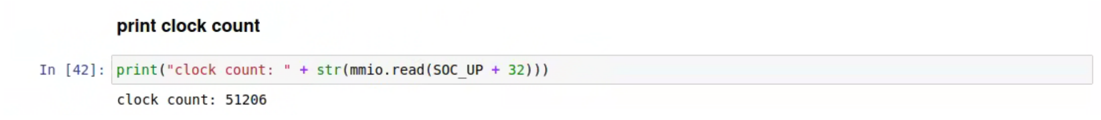
[40] Python

```
waiting userdma transfer done...
userdma finished!!!
```

```python
## utility function for converting int to hex
def int_to_32bit_hex(n):
    # Handle negative numbers by converting to unsigned 32-bit equivalent
    if n < 0:
        n = (1 << 32) + n
    return f"{n:08x}"


# write fir output to file
fir_output_file = open("/home/xilinx/jupyter_notebooks/updma_output.log", "w")
for i in range(len(aes128_output_buf)):
    fir_output_file.write(int_to_32bit_hex(aes128_output_buf[i])+'\n')
fir_output_file.close()
print("fir output written to 'updma_output.log'")
```

[41] Python

```
fir output written to 'updma_output.log'
```


print clock count

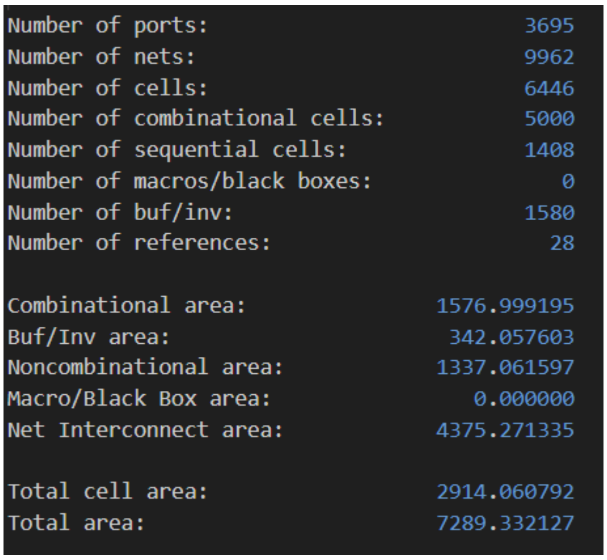
```
In [42]: print("clock count: " + str(mmio.read(SOC_UP + 32)))
         clock count: 51206
```

It took 51206 clock cycles to encrypt 1024 blocks (16384 bytes) of data, which means 50 cycles per block.

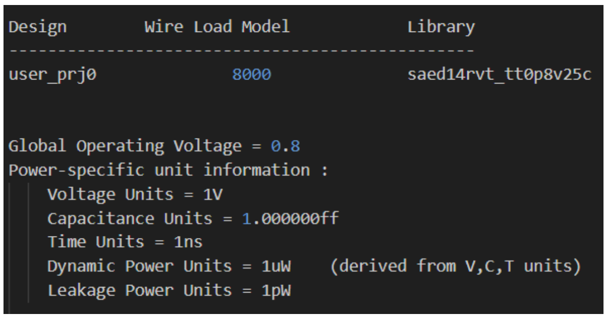
It seems like using throughput cycles=45 is not fast enough. Also, we can add a FIFO at the output port so that it doesn't have to wait until all the ciphertext is streamed out before it can start encrypting next block.

Synopsys flow:

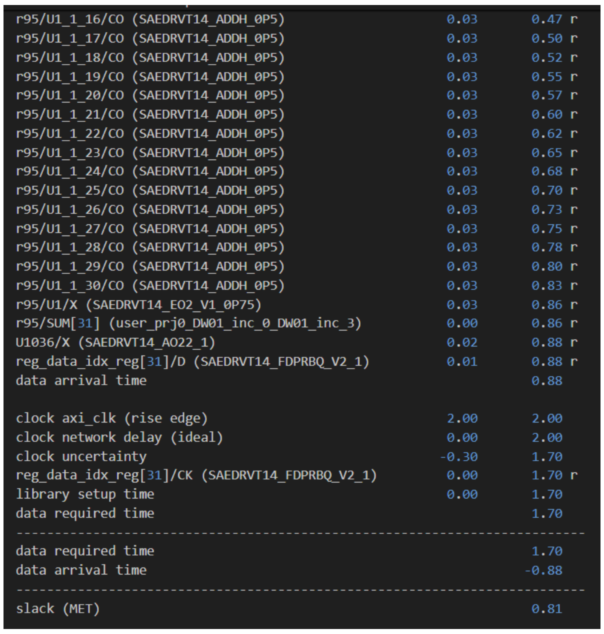
✓ Synthesis:



```
Number of ports:                          3695
Number of nets:                           9962
Number of cells:                          6446
Number of combinational cells:            5000
Number of sequential cells:               1408
Number of macros/black boxes:                0
Number of buf/inv:                        1580
Number of references:                       28

Combinational area:                1576.999195
Buf/Inv area:                       342.057603
Noncombinational area:             1337.061597
Macro/Black Box area:                 0.000000
Net Interconnect area:             4375.271335

Total cell area:                   2914.060792
Total area:                        7289.332127
```

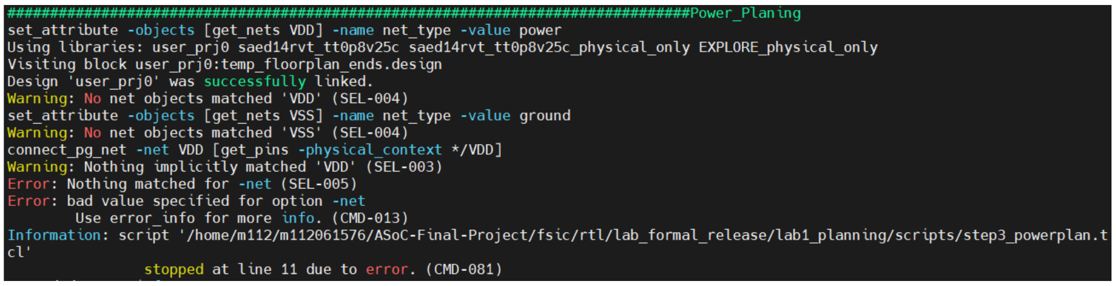


```
Design        Wire Load Model           Library
------------------------------------------------
user_prj0              8000              saed14rvt_tt0p8v25c


Global Operating Voltage = 0.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1pW
```

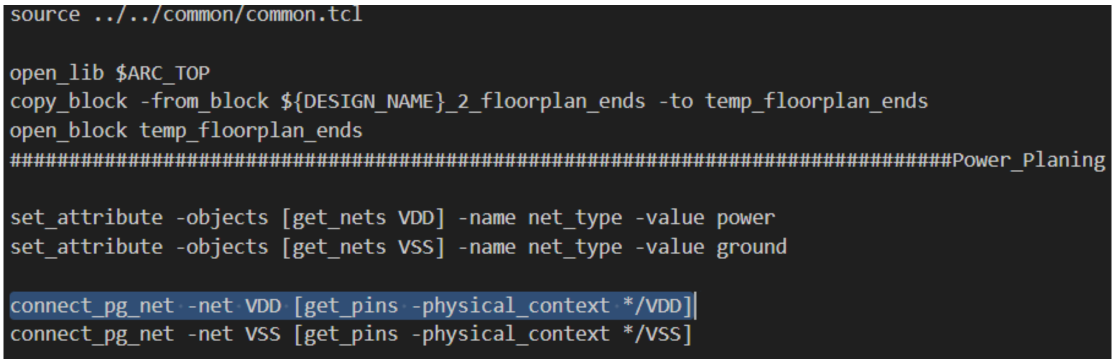


```
r95/U1_1_16/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.47 r
r95/U1_1_17/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.50 r
r95/U1_1_18/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.52 r
r95/U1_1_19/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.55 r
r95/U1_1_20/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.57 r
r95/U1_1_21/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.60 r
r95/U1_1_22/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.62 r
r95/U1_1_23/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.65 r
r95/U1_1_24/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.68 r
r95/U1_1_25/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.70 r
r95/U1_1_26/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.73 r
r95/U1_1_27/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.75 r
r95/U1_1_28/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.78 r
r95/U1_1_29/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.80 r
r95/U1_1_30/CO (SAEDRVT14_ADDH_0P5)                 0.03      0.83 r
r95/U1/X (SAEDRVT14_EO2_V1_0P75)                    0.03      0.86 r
r95/SUM[31] (user_prj0_DW01_inc_0_DW01_inc_3)       0.00      0.86 r
U1036/X (SAEDRVT14_AO22_1)                          0.02      0.88 r
reg_data_idx_reg[31]/D (SAEDRVT14_FDPRBQ_V2_1)      0.01      0.88 r
data arrival time                                             0.88

clock axi_clk (rise edge)                           2.00      2.00
clock network delay (ideal)                         0.00      2.00
clock uncertainty                                  -0.30      1.70
reg_data_idx_reg[31]/CK (SAEDRVT14_FDPRBQ_V2_1)     0.00      1.70 r
library setup time                                  0.00      1.70
data required time                                            1.70
---------------------------------------------------------------------------
data required time                                            1.70
data arrival time                                            -0.88
---------------------------------------------------------------------------
slack (MET)                                                   0.81
```

✓ Floorplan:



```
###############################################################################Power_Planing
set_attribute -objects [get_nets VDD] -name net_type -value power
Using libraries: user_prj0 saed14rvt_tt0p8v25c saed14rvt_tt0p8v25c_physical_only EXPLORE_physical_only
Visiting block user_prj0:temp_floorplan_ends.design
Design 'user_prj0' was successfully linked.
Warning: No net objects matched 'VDD' (SEL-004)
set_attribute -objects [get_nets VSS] -name net_type -value ground
Warning: No net objects matched 'VSS' (SEL-004)
connect_pg_net -net VDD [get_pins -physical_context */VDD]
Warning: Nothing implicitly matched 'VDD' (SEL-003)
Error: Nothing matched for -net (SEL-005)
Error: bad value specified for option -net
        Use error_info for more info. (CMD-013)
Information: script '/home/m112/m112061576/ASoC-Final-Project/fsic/rtl/lab_formal_release/lab1_planning/scripts/step3_powerplan.t
cl'
                stopped at line 11 due to error. (CMD-081)
```



```
source ../../common/common.tcl

open_lib $ARC_TOP
copy_block -from_block ${DESIGN_NAME}_2_floorplan_ends -to temp_floorplan_ends
open_block temp_floorplan_ends
###############################################################################Power_Planing

set_attribute -objects [get_nets VDD] -name net_type -value power
set_attribute -objects [get_nets VSS] -name net_type -value ground

connect_pg_net -net VDD [get_pins -physical_context */VDD]
connect_pg_net -net VSS [get_pins -physical_context */VSS]
```

An error occurred when executing the power plan. From the previous i2c_master_top.v of lab3, we learned that the input of this file includes vdd and gnd. And our own design does not have vdd, gnd.

It is speculated that i2c_master_top is a small circuit and may not need a power ring, so it directly provides vdd and gnd to the circuit.

Github link:

https://github.com/ruei7916/ASoC-Final-Project

Reference:

1.NIST FIPS 197 Advanced Encryption Standard (AES)

https://csrc.nist.gov/pubs/fips/197/final

2.Cryptographic Standards and Guidelines - AES Development

https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development