

AES

Dongjun Lee

AES structure

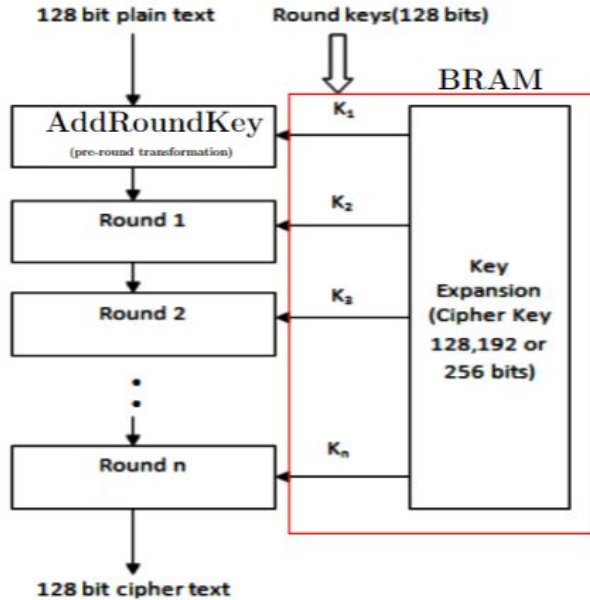


Figure 1: AES-128 (Very High Level Structure)

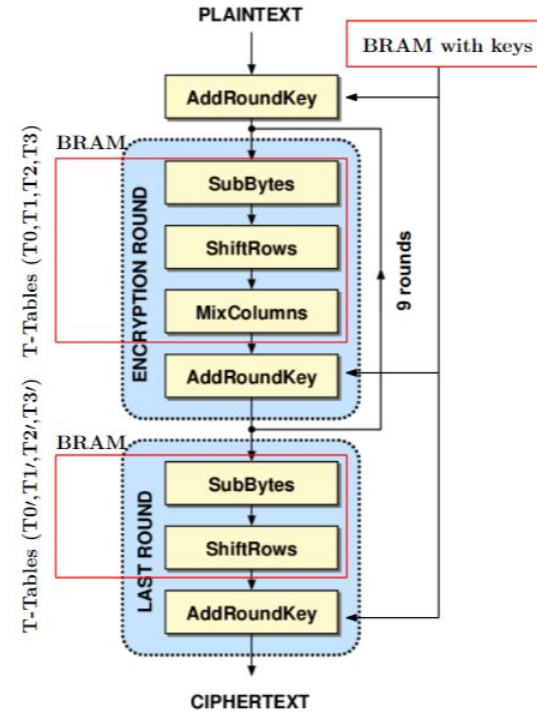


Figure 2: AES-128 (very high level structure with rounds)

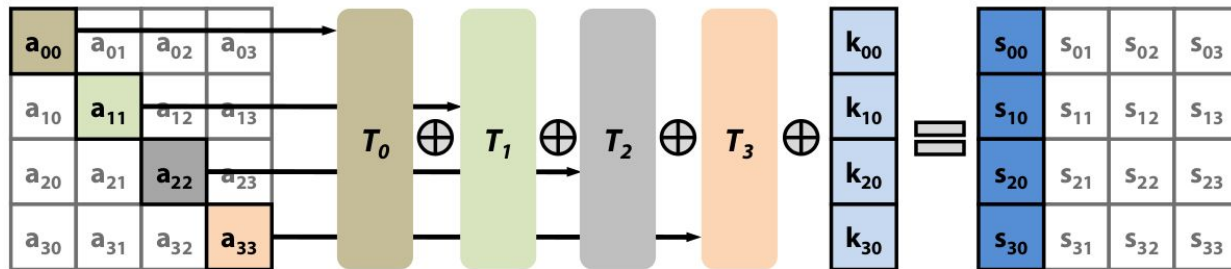
Fastest AES (by T-Table)

- Combine SubBytes, ShiftRows, MixColumns using the standard “ T -table” approach. Update each column ($0 \leq j \leq 3$):

$$[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^T = T_0[a_{c_00}] \oplus T_1[a_{c_11}] \oplus T_2[a_{c_22}] \oplus T_3[a_{c_33}] \oplus k_j,$$

where each T_i is 1KB and k_j is the j th column of the round key.

- T_i 's are rotations of one table.
- Example ($j = 0$):



Description

When describing the cipher's operations, A is denoted as the input block consisting of bytes $a_{i,j}$ in columns C_j and rows R_i , where j and i are the respective indices ranging between 0 and 3.

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

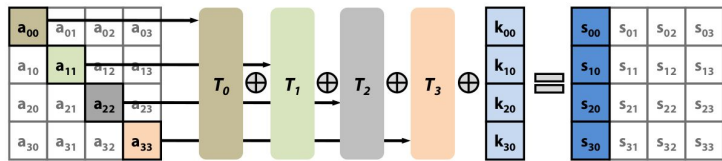
Input data to an AES encryption can be defined as four 32 bit column vectors $C_j = (a_{0,j}, a_{1,j}, a_{2,j}, a_{3,j})$ with the output similarly formatted in column vectors. According to Equation 1, these input column vectors need to be split into individual bytes since all bytes are required for the computation steps for different E'_j . For example, for column $C_0 = (a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0})$ the first byte $a_{0,0}$ is part of the computation of E'_0 , the second byte $a_{1,0}$ is used in E'_3 , etc. Since fixed (and thus simple) data paths are prefer-

$$\begin{aligned} E'_0 &= K_r[0] \oplus T_0(a_{0,0}) \oplus T_1(a_{1,1}) \oplus T_2(a_{2,2}) \oplus T_3(a_{3,3}) \\ &= (a'_{0,0}, a'_{1,0}, a'_{2,0}, a'_{3,0}) \\ E'_1 &= K_r[1] \oplus T_3(a_{3,0}) \oplus T_0(a_{0,1}) \oplus T_1(a_{1,2}) \oplus T_2(a_{2,3}) \\ &= (a'_{0,1}, a'_{1,1}, a'_{2,1}, a'_{3,1}) \\ E'_2 &= K_r[2] \oplus T_2(a_{2,0}) \oplus T_3(a_{3,1}) \oplus T_0(a_{0,2}) \oplus T_1(a_{1,3}) \\ &= (a'_{0,2}, a'_{1,2}, a'_{2,2}, a'_{3,2}) \\ E'_3 &= K_r[3] \oplus T_1(a_{1,0}) \oplus T_2(a_{2,1}) \oplus T_3(a_{3,2}) \oplus T_0(a_{0,3}) \\ &= (a'_{0,3}, a'_{1,3}, a'_{2,3}, a'_{3,3}) \end{aligned}$$

where $a_{i,j}$ denotes an input byte, and $a'_{i,j}$ the corresponding output byte after the round transformation. However, the unified input datapath still requires a look-up to all of the four T-tables for the second operand of each XOR operation. For example, the XOR component at the first position of the sequential operations E'_0 to E'_3 and thus requires the lookups $T_0(a_{0,0})$, $T_3(a_{3,0})$, $T_2(a_{2,0})$ and $T_1(a_{1,0})$ (in this order) and the corresponding round key $K_r[j]$. Though

Hints

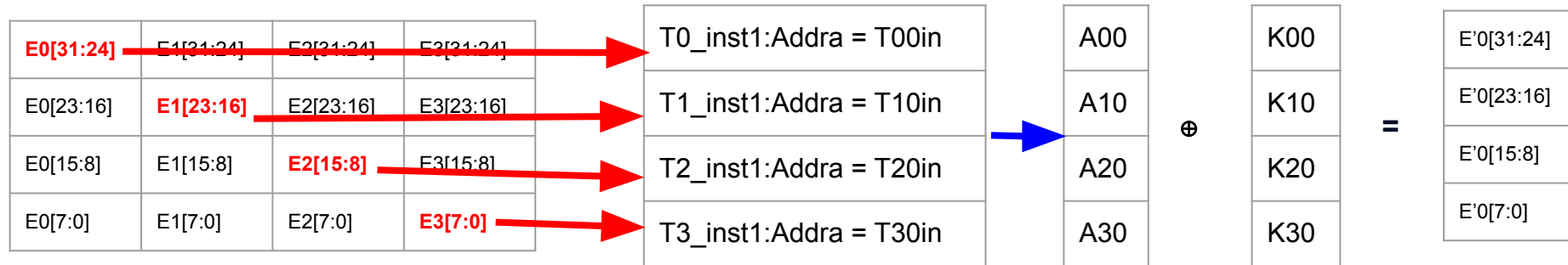
- The signals $\{e0, e1, e2, e3\}$ are the 32 bit inputs for each round
- The signals $\{a0, a1, a2, a3\}$ are the 32 bit outputs of the T-tables
- In other words: $\{a0, a1, a2, a3\}$ with additional AddRoundKey will form the new round input
- Tutorial-week4.txt assumes a structure where we need 4xT0, 4xT1, 4xT2, 4xT3 as we want to perform the quarter-round function 4x in parallel
 - a. BRAMT0_inst1 = [T00, T01] with just one .coe file, as T00 and T01 will be effectively the same memory content; they will be just used with a different address because (because we need this 4x in parallel)
 - b. BRAMT0_inst2 = [T02, T03]
 - c. Likewise, BRAMT1_inst1 = [T10,T11], BRAMT1_inst2 = [T12, T13], ...
 - d. Bram_addr[8:0] = {final_round, 8-bit-addr}
 - i. When final_round = 0, default values. When final_round = 1, transposed values.
 1. rounds 1-9: final_round = 0 || 8-bit-addr
 2. rounds 10: final_round = 1 || 8-bit-addr
- BRAM output requires 2 cycles for the output to be valid.



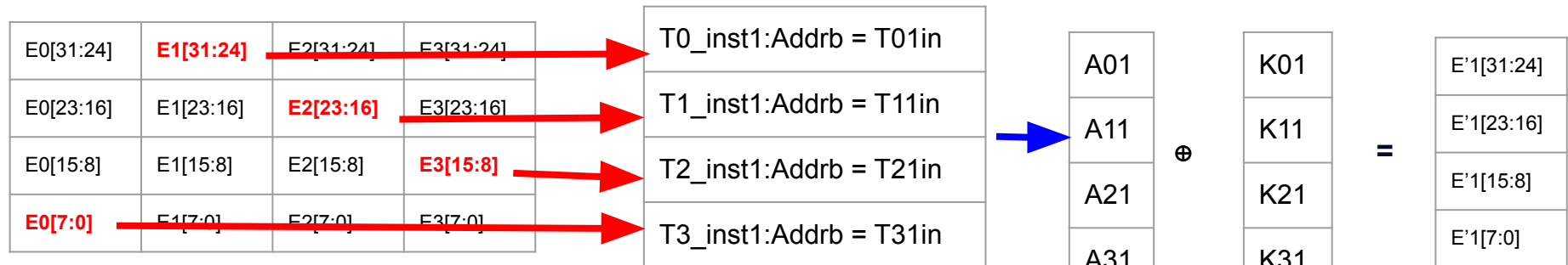
(Fig. Scenario of 1/4 round)

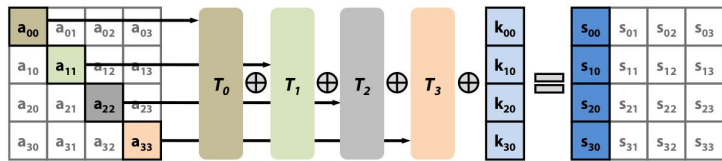
$$\begin{aligned}
 E'_0 &= K_r[0] \oplus T_0(a_{0,0}) \oplus T_1(a_{1,1}) \oplus T_2(a_{2,2}) \oplus T_3(a_{3,3}) \\
 &= (a'_{0,0}, a'_{1,0}, a'_{2,0}, a'_{3,0}) \\
 E'_1 &= K_r[1] \oplus T_3(a_{3,0}) \oplus T_0(a_{0,1}) \oplus T_1(a_{1,2}) \oplus T_2(a_{2,3}) \\
 &= (a'_{0,1}, a'_{1,1}, a'_{2,1}, a'_{3,1}) \\
 E'_2 &= K_r[2] \oplus T_2(a_{2,0}) \oplus T_3(a_{3,1}) \oplus T_0(a_{0,2}) \oplus T_1(a_{1,3}) \\
 &= (a'_{0,2}, a'_{1,2}, a'_{2,2}, a'_{3,2}) \\
 E'_3 &= K_r[3] \oplus T_1(a_{1,0}) \oplus T_2(a_{2,1}) \oplus T_3(a_{3,2}) \oplus T_0(a_{0,3}) \\
 &= (a'_{0,3}, a'_{1,3}, a'_{2,3}, a'_{3,3})
 \end{aligned}$$

Quarter round 1/4



Quarter round 2/4

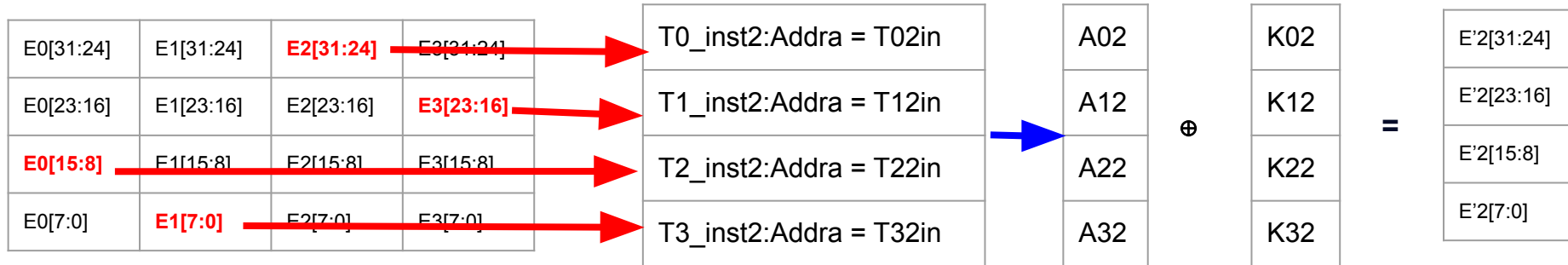




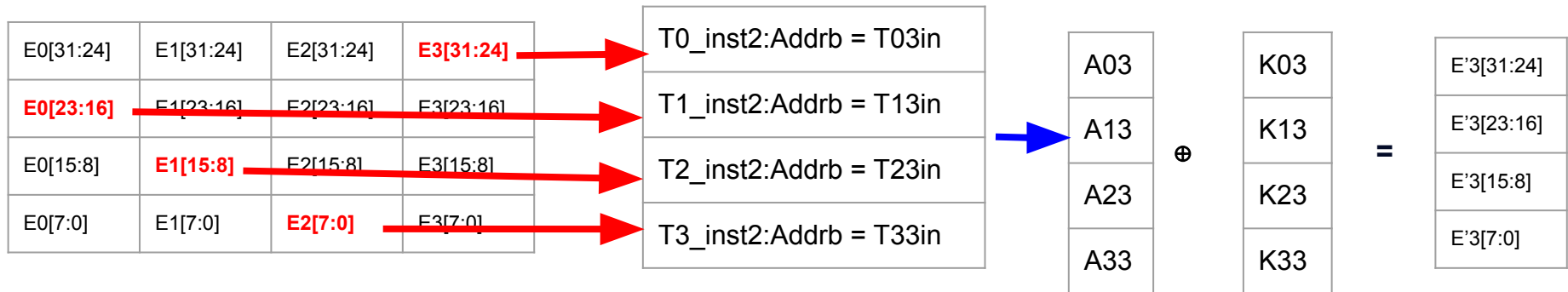
(Fig. Scenario of 1/4 round)

$$\begin{aligned}
 E'_0 &= K_r[0] \oplus T_0(a_{0,0}) \oplus T_1(a_{1,1}) \oplus T_2(a_{2,2}) \oplus T_3(a_{3,3}) \\
 &= (a'_{0,0}, a'_{1,0}, a'_{2,0}, a'_{3,0}) \\
 E'_1 &= K_r[1] \oplus T_3(a_{3,0}) \oplus T_0(a_{0,1}) \oplus T_1(a_{1,2}) \oplus T_2(a_{2,3}) \\
 &= (a'_{0,1}, a'_{1,1}, a'_{2,1}, a'_{3,1}) \\
 E'_2 &= K_r[2] \oplus T_2(a_{2,0}) \oplus T_3(a_{3,1}) \oplus T_0(a_{0,2}) \oplus T_1(a_{1,3}) \\
 &= (a'_{0,2}, a'_{1,2}, a'_{2,2}, a'_{3,2}) \\
 E'_3 &= K_r[3] \oplus T_1(a_{1,0}) \oplus T_2(a_{2,1}) \oplus T_3(a_{3,2}) \oplus T_0(a_{0,3}) \\
 &= (a'_{0,3}, a'_{1,3}, a'_{2,3}, a'_{3,3})
 \end{aligned}$$

Quarter round 3/4



Quarter round 4/4



Scenario

Round 1:

PlainTXT [127:96]	PlainTXT [95:64]	PlainTXT [63:32]	PlainTXT [31:0]
E0[31:24]	E1[31:24]	E2[31:24]	E3[31:24]
E0[23:16]	E1[23:16]	E2[23:16]	E3[23:16]
E0[15:8]	E1[15:8]	E2[15:8]	E3[15:8]
E0[7:0]	E1[7:0]	E2[7:0]	E3[7:0]

⊕

Roundkey [127:96]	Roundkey [95:64]	Roundkey [63:32]	Roundkey [31:0]
K00	K01	K02	K03
K10	K11	K12	K13
K20	K21	K22	K23
K30	K31	K32	K33

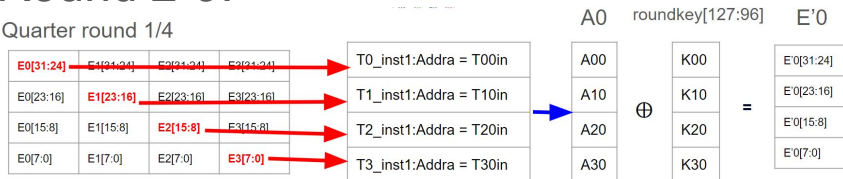


E'0[31:24]	E'1[31:24]	E'2[31:24]	E'3[31:24]
E'0[23:16]	E'1[23:16]	E'2[23:16]	E'3[23:16]
E'0[15:8]	E'1[15:8]	E'2[15:8]	E'3[15:8]
E'0[7:0]	E'1[7:0]	E'2[7:0]	E'3[7:0]

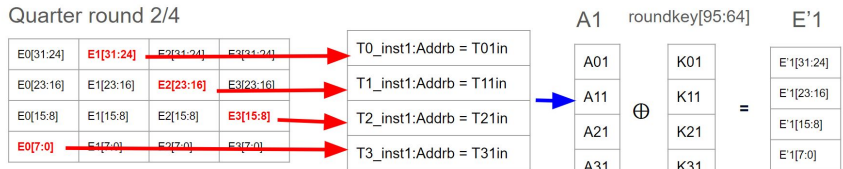
Scenario

Round 2-9: Bram_addr[8:0] = {final_round = 0, 8-bit-addr}

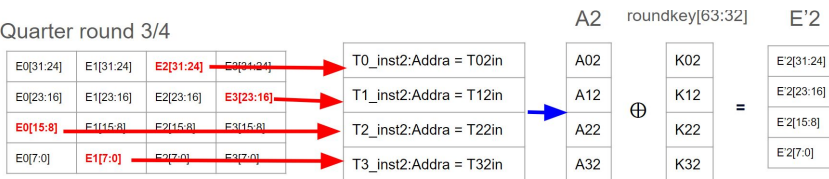
Quarter round 1/4



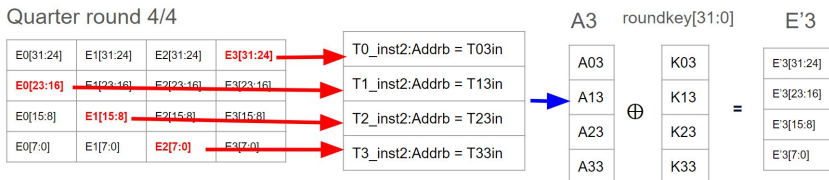
Quarter round 2/4



Quarter round 3/4



Quarter round 4/4

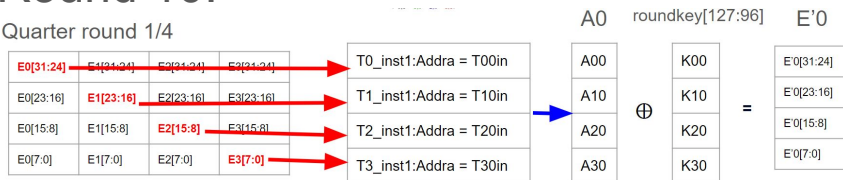


E'0[31:24]	E'1[31:24]	E'2[31:24]	E'3[31:24]
E'0[23:16]	E'1[23:16]	E'2[23:16]	E'3[23:16]
E'0[15:8]	E'1[15:8]	E'2[15:8]	E'3[15:8]
E'0[7:0]	E'1[7:0]	E'2[7:0]	E'3[7:0]

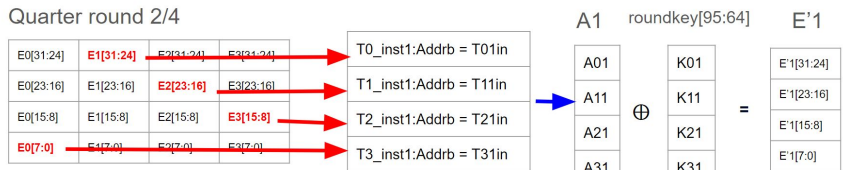
Scenario

Round 10: Bram_addr[8:0] = {final_round = 1, 8-bit-addr}

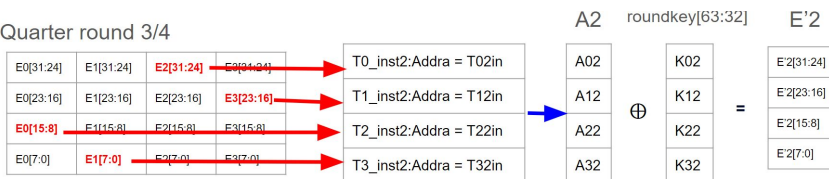
Quarter round 1/4



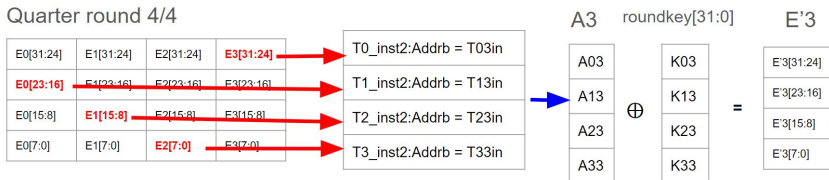
Quarter round 2/4



Quarter round 3/4

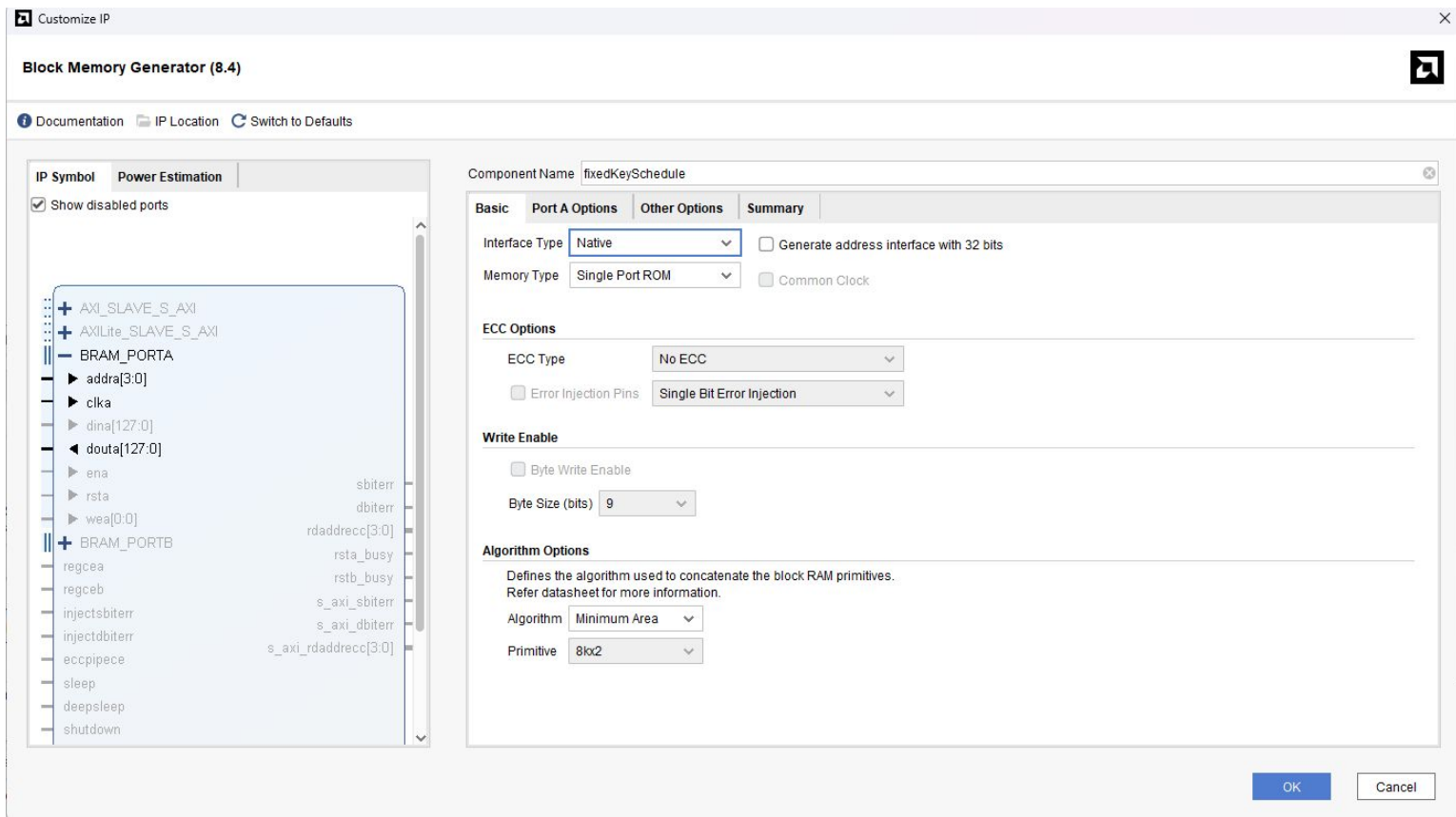


Quarter round 4/4



E'0[31:24]	E'1[31:24]	E'2[31:24]	E'3[31:24]
E'0[23:16]	E'1[23:16]	E'2[23:16]	E'3[23:16]
E'0[15:8]	E'1[15:8]	E'2[15:8]	E'3[15:8]
E'0[7:0]	E'1[7:0]	E'2[7:0]	E'3[7:0]

BRAM for fixedkeyschedule



BRAM for fixedkeyschedule

Customize IP

Block Memory Generator (8.4)

Documentation

IP Location

Switch to Defaults

IP Symbol

Power Estimation

Show disabled ports

AXI_SLAVE_S_AXI

AXILite_SLAVE_S_AXI

BRAM_PORTA

addra[3:0]

clka

dina[127:0]

douta[127:0]

ena

rsta

wea[0:0]

BRAM_PORTB

regcea

regceb

injectsbiterr

injectdbiterr

eccpipece

sleep

deepsleep

shutdown

sbiterr

dbiterr

rdaddrecc[3:0]

rsta_busy

rstb_busy

s_axi_sbiterr

s_axi_dbiterr

s_axi_rdadrecc[3:0]

Component Name

fixedKeySchedule

Basic

Port A Options

Other Options

Summary

Memory Size

Port A Width128Range: 1 to 4608 (bits)

Port A Depth16Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

Operating ModeWrite FirstEnable Port TypeAlways Enabled

Port A Optional Output Registers

☒ Primitives Output Register

☐ Core Output Register

☐ SoftECC Input Register

☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (set/reset pin)

Output Reset Value (Hex)0

☐ Reset Memory Latch

Reset PriorityCE (Latch or Register Enable)

READ Address Change A

☐ Read Address Change A

OK

Cancel

BRAM for fixedkeyschedule

Customize IP

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

☒ Show disabled ports

+

AXI_SLAVE_S_AXI

+

AXILite_SLAVE_S_AXI

||

BRAM_PORTA

▶

addra[3:0]

▶

clka

▶

dina[127:0]

◀

douta[127:0]

▶

ena

▶

rsta

▶

wea[0:0]

||

+

BRAM_PORTB

—

regcea

—

regceb

—

injectsbiterr

—

injectdbiterr

—

eccpipece

—

sleep

—

deepsleep

—

shutdown

sbiterr

dbiterr

rdaddrrecc[3:0]

rsta_busy

rstb_busy

s_axi_sbiterr

s_axi_dbiterr

s_axi_rdaddrrecc[3:0]

Component NamefixedKeySchedule

BasicPort A OptionsOther OptionsSummary

Pipeline Stages within Mux0Mux Size: 1x1

Memory Initialization

☒ Load Init File

Coe File024_Dongjun/HA5_verilog/coe files/fixedKeyschedule.coeBrowseEdit

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex)0

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision WarningsAll

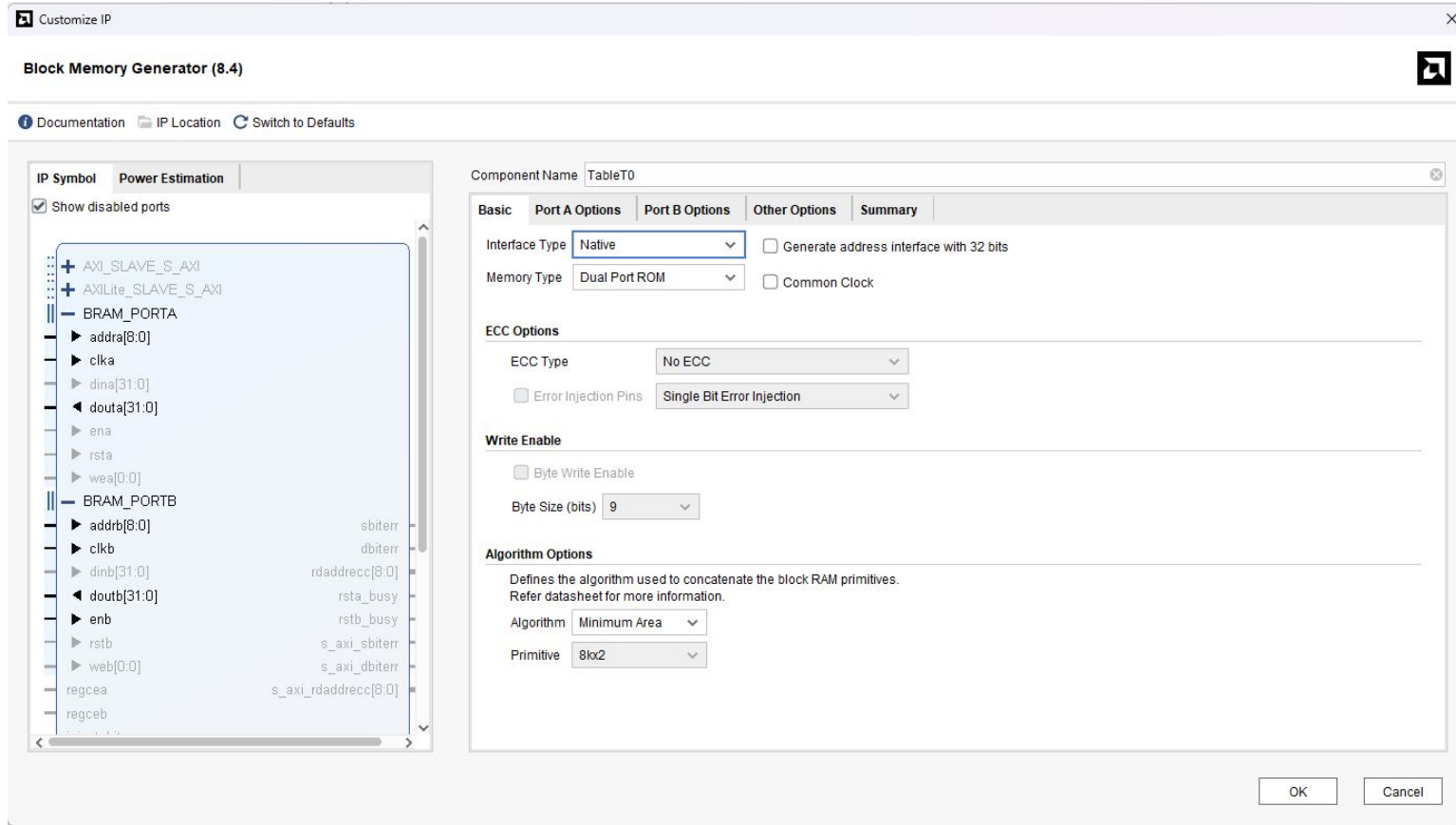
Behavioral Simulation Model Options

☐ Disable Collision Warnings☐ Disable Out of Range Warnings

OK

Cancel

BRAM for LUT: T0, T1, T2, and T3



BRAM for LUT: T0, T1, T2, and T3

Customize IP

Block Memory Generator (8.4)

[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP Symbol

Power Estimation

☒ Show disabled ports

+ AXI_SLAVE_S_AXI

+ AXILite_SLAVE_S_AXI

- BRAM_PORTA

▶ addra[8:0]

▶ clka

▶ dina[31:0]

◀ douta[31:0]

▶ ena

▶ rsta

▶ wea[0:0]

- BRAM_PORTB

▶ addrb[8:0]

▶ clk_b

▶ dinb[31:0]

◀ doutb[31:0]

▶ enb

▶ rstb

▶ web[0:0]

- regcea

- regceb

sbiterr

dbiterr

rdaddrecc[8:0]

rsta_busy

rstb_busy

s_axi_sbiterr

s_axi_dbiterr

s_axi_rdaddrecc[8:0]

Component Name

TableT0

Basic

Port A Options

Port B Options

Other Options

Summary

Memory Size

Port A Width Range: 1 to 4608 (bits)

Port A Depth Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

Operating Mode Enable Port Type

Port A Optional Output Registers

☒ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (set/reset pin) Output Reset Value (Hex)

☐ Reset Memory Latch Reset Priority

READ Address Change A

☐ Read Address Change A

OK

Cancel

BRAM for LUT: T0, T1, T2, and T3

Re-customize IP

Block Memory Generator (8.4)

[Documentation](#) [IP Location](#) [Switch to Defaults](#)

IP SymbolPower Estimation

☒ Show disabled ports

+

AXI_SLAVE_S_AXI

+

AXILite_SLAVE_S_AXI

+

BRAM_PORTA

+

BRAM_PORTB

regcea

regceb

injectsbiterr

injectdbiterr

eccpipece

sleep

deepsleep

shutdown

s_ack

s_aretstn

s_axi_injectsbiterr

s_axi_injectdbiterr

sbiterr

dbiterr

rdaddrecc[8:0]

rsta_busy

rstb_busy

s_axi_sbiterr

s_axi_dbiterr

s_axi_rdaddrecc[8:0]

Component NameTableT0

BasicPort A OptionsPort B OptionsOther OptionsSummary

Memory Size

Port B Width32

Port B Depth : 512

The Width and Depth values are used for Read Operation in Port B

Operating ModeWrite First

Enable Port TypeAlways Enabled

Port B Optional Output Registers

☒ Primitives Output Register☐ Core Output Register

☐ SoftECC Output Register☐ REGCEB Pin

Port B Output Reset Options

☐ RSTB Pin (set/reset pin)Output Reset Value (Hex)0

☐ Reset Memory LatchReset PriorityCE (Latch or Register Enable)

READ Address Change B

☐ Read Address Change B

OK

Cancel

BRAM for LUT: T0 (T1, T2, and T3)

Customize IP

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol Power Estimation

☒ Show disabled ports

- + AXI_SLAVE_S_AXI
- + AXILite_SLAVE_S_AXI
- BRAM_PORTA
 - ▶ addr[8:0]
 - ▶ clka
 - ▶ dina[31:0]
 - ◀ douta[31:0]
 - ▶ ena
 - ▶ rsta
 - ▶ wea[0:0]
- BRAM_PORTB
 - ▶ addrb[8:0]
 - ▶ clkb
 - ▶ dinb[31:0]
 - ◀ doutb[31:0]
 - ▶ enb
 - ▶ rstb
 - ▶ web[0:0]
 - regcea
 - regceb

sbiterr
dbiterr
rdaddrecc[8:0]
rsta_busy
rstb_busy
s_axi_sbiterr
s_axi_dbiterr
s_axi_rdaddrecc[8:0]

Component Name TableT0

Basic Port A Options Port B Options Other Options Summary

Pipeline Stages within Mux 0 Mux Size: 1x1

Memory Initialization

☒ Load Init File

Coe File ingjun/HA5_verilog/coe files/TableT0_with final_round.coe Browse Edit

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex) 0

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

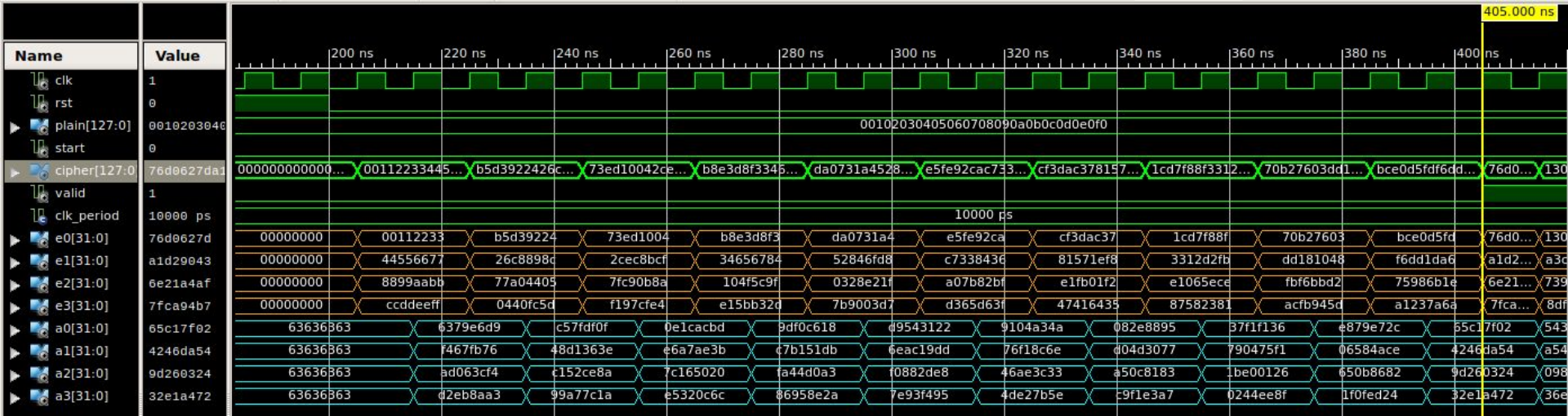
Collision Warnings All

Behavioral Simulation Model Options

☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

OK Cancel

Output



AES-128 plaintext input:

00102030405060708090a0b0c0d0e0f0

The final output is:

76d0627da1d290436e21a4af7fca94b7