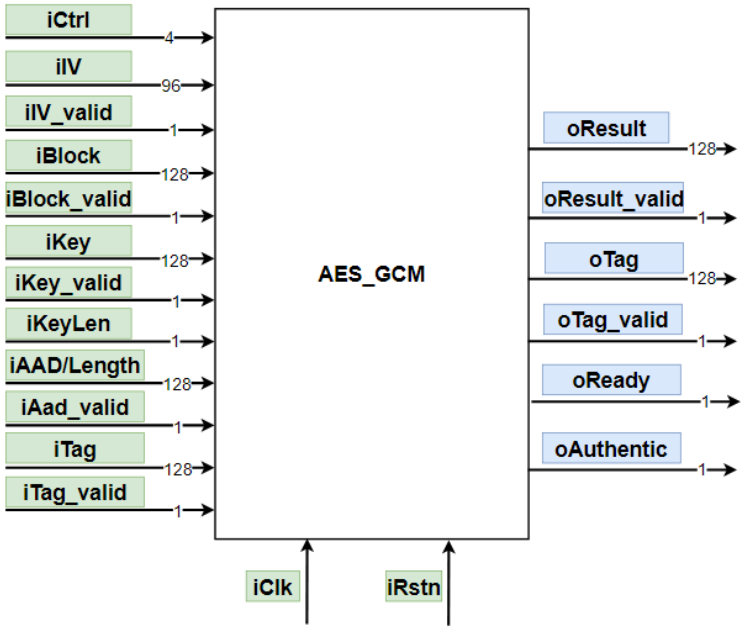
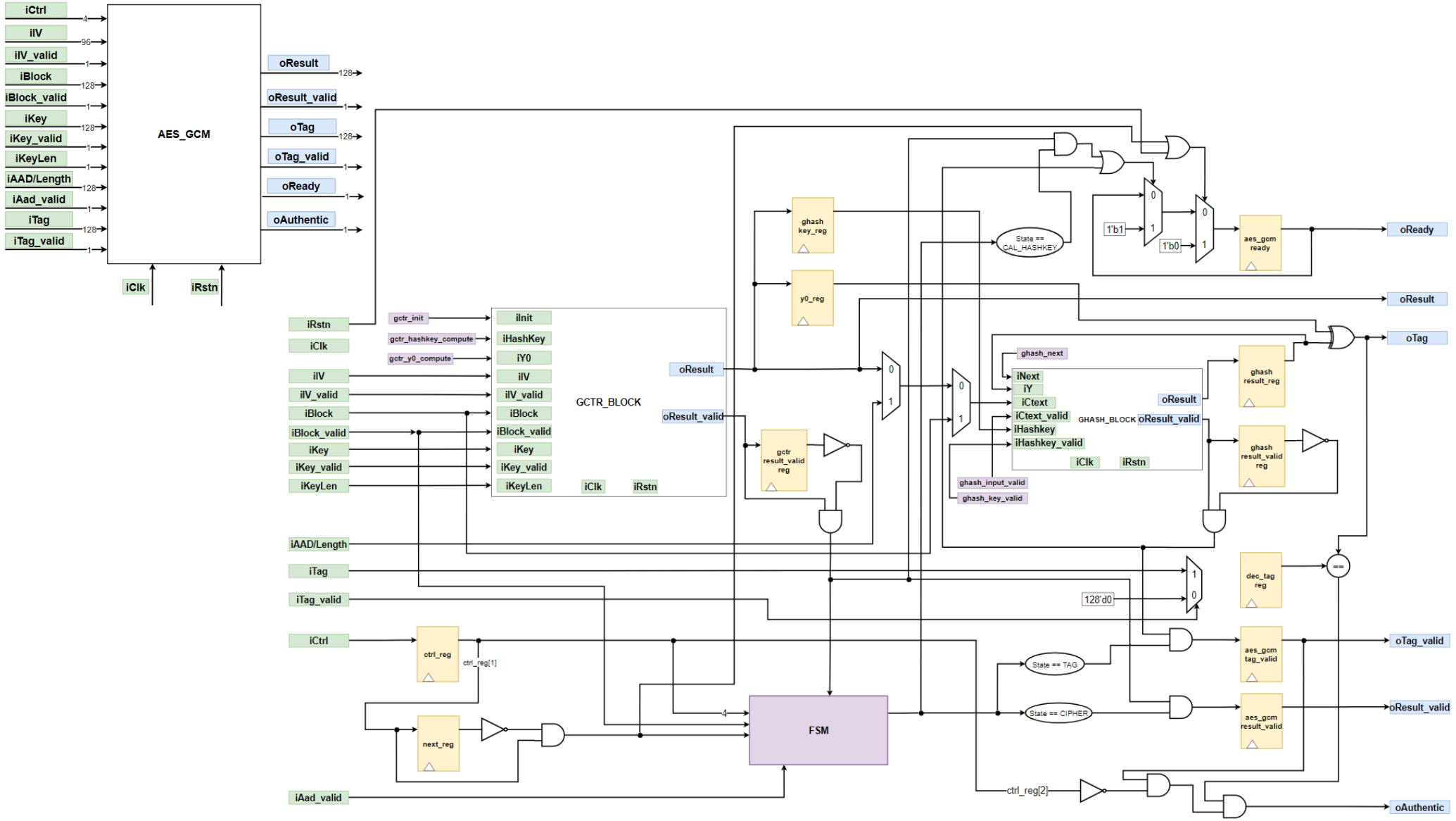
AES-GCM

BLOCK DIAGRAMs

and STATE MACHINEs

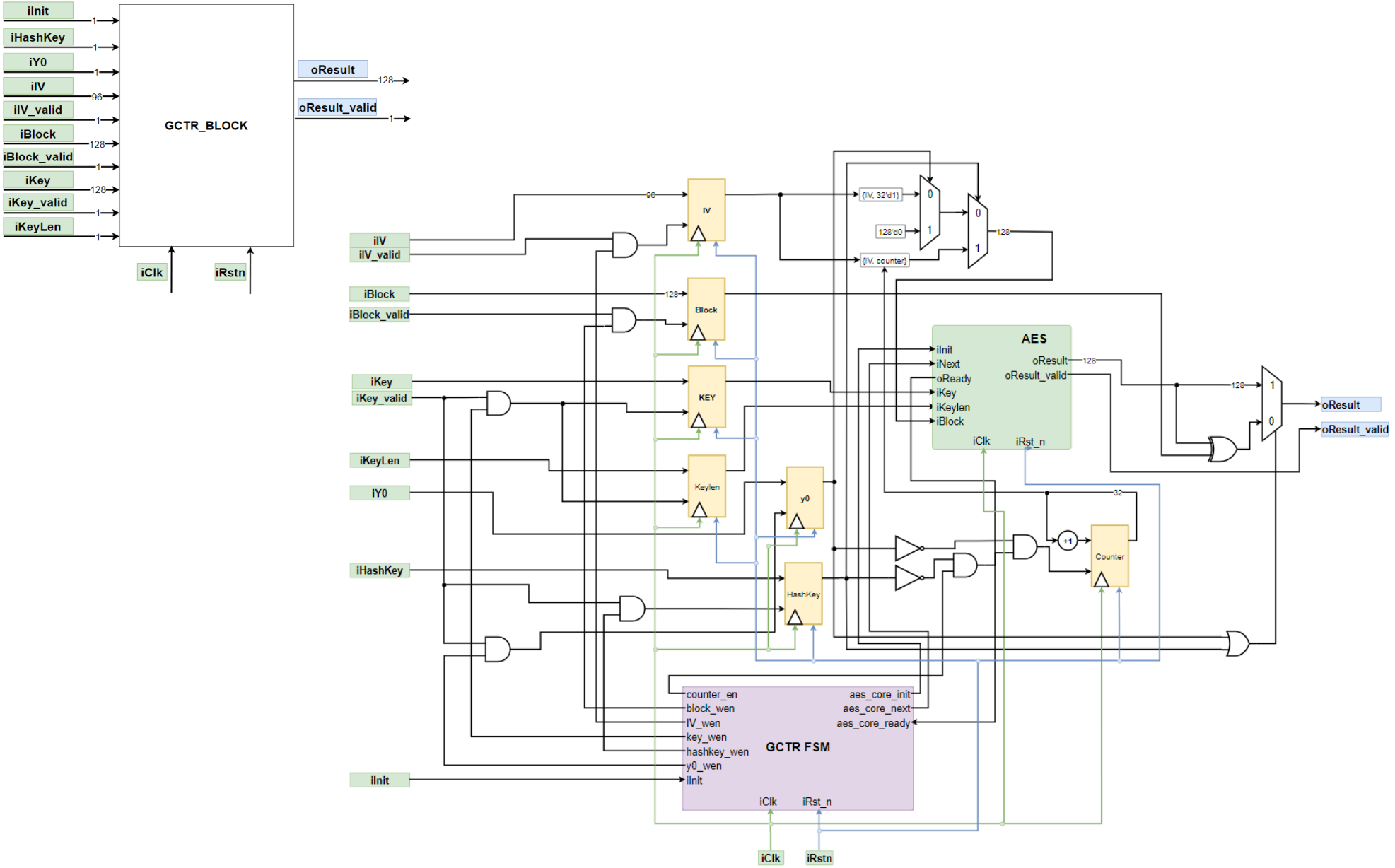

iCtrl
4
iIV
96
iIV_valid
1
iBlock
128
iBlock_valid
1
iKey
128
iKey_valid
1
iKeyLen
1
iAAD/Length
128
iAad_valid
1
iTag
128
iTag_valid
1
AES_GCM
oResult
128
oResult_valid
1
oTag
128
oTag_valid
1
oReady
1
oAuthentic
1
iClk
iRstn

PIN	Direction	Width	Description
iClk	Input	1	Clock
iRstn	Input	1	Negative edge reset: reset after using the core
CONTROL			
iCtrl	Input	4	Initialize signal: assert when using the core
oReady	Output	1	Output ready signal: input new data when ready is asserted
DATA			
iIV	Input	96	96-bit length IV
iIV_valid	Input	1	When asserted, IV is valid
iKey	Input	256	Key
iKey_valid	Input	1	When asserted, Key is valid
iKey_len	Input	1	When asserted, Key length is 256-bit. When deasserted, Key length is 128-bit.
iAad	Input	128	Additional Authentic Data or Length(A,C): when Ready = 1, change data every clock cycle
iAad_valid	Input	1	When asserted, AAD is valid.
iBlock	Input	128	Input Plaintext or Ciphertext.
iBlock_valid	Input	1	When asserted, Plaintext or Ciphertext is valid.
iTag	Input	128	Input Tag for authentication in Decryption mode
iTag_valid	Input	1	When asserted, Tag is valid
OUTPUT			
oResult	Output	128	Output Ciphertext or Ciphertext
oResult_valid	Output	1	When asserted, Result is valid
oTag	Output	128	Output Tag in Encryption mode.
oTag_valid	Output	1	When asserted, output Tag is valid
oAuthentic	Output	1	When asserted, indicating authentic Block in Decryption mode.



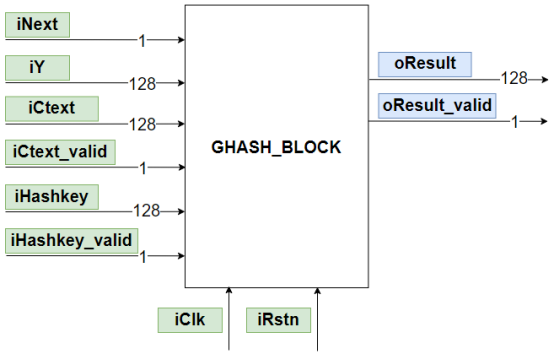
Submodule	File
GCTR_BLOCK	gctr_block.v
GHASH_BLOCK	ghash_block.v

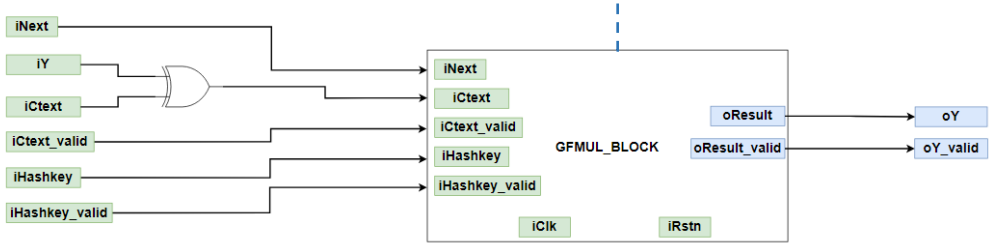
Name	AES_GCM
File	aes_gcm.v

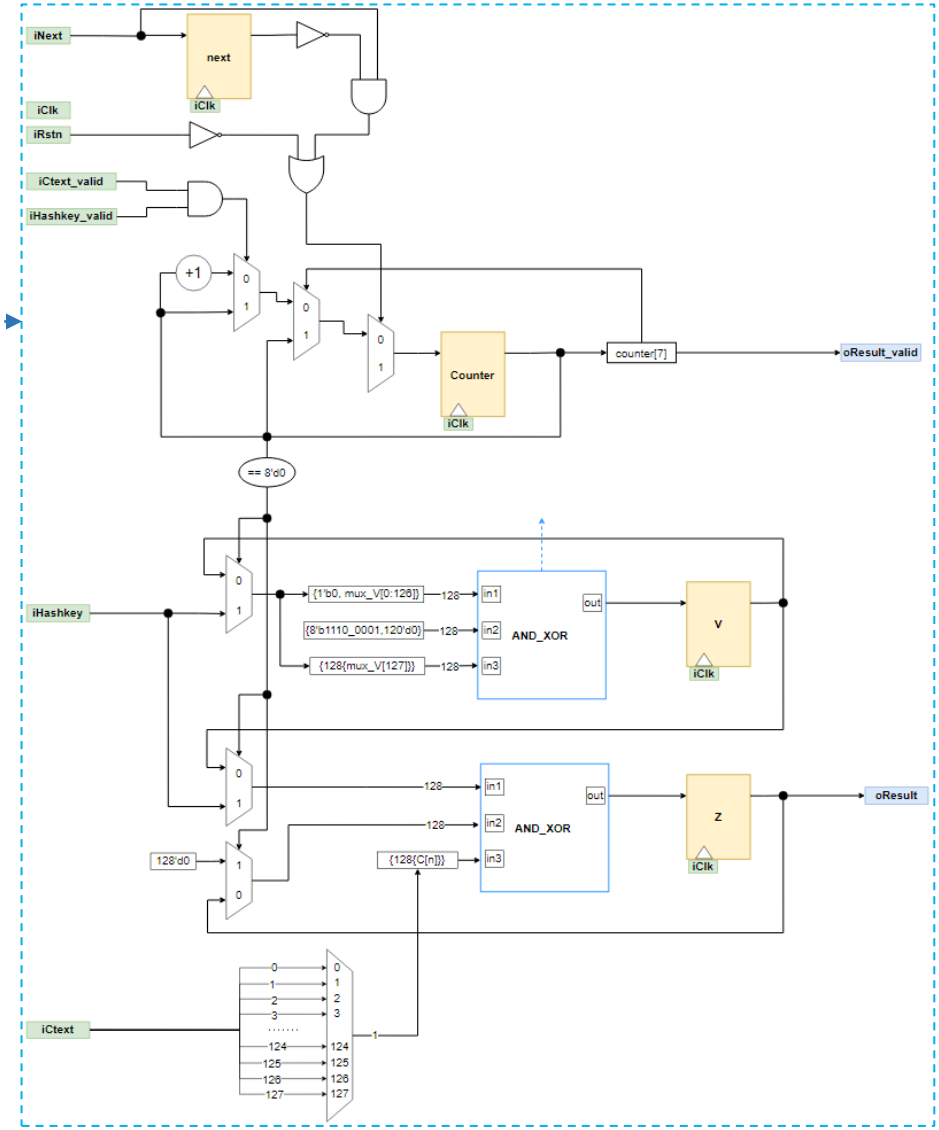


Submodule	File
AES	aes_core.v

Name	GCTR_BLOCK
File	gctr_block.v



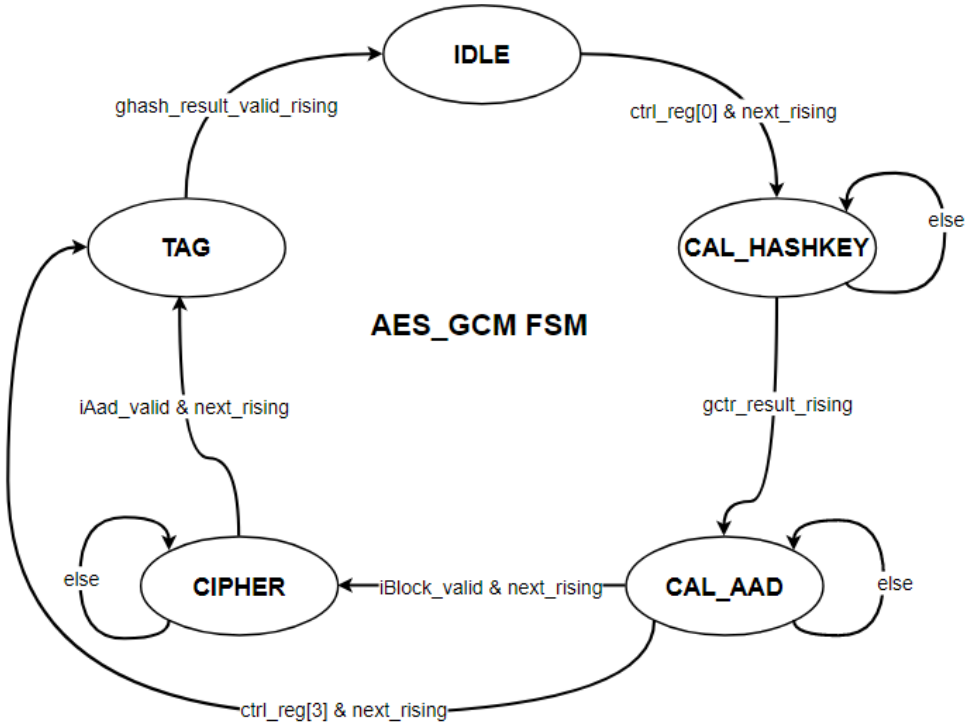


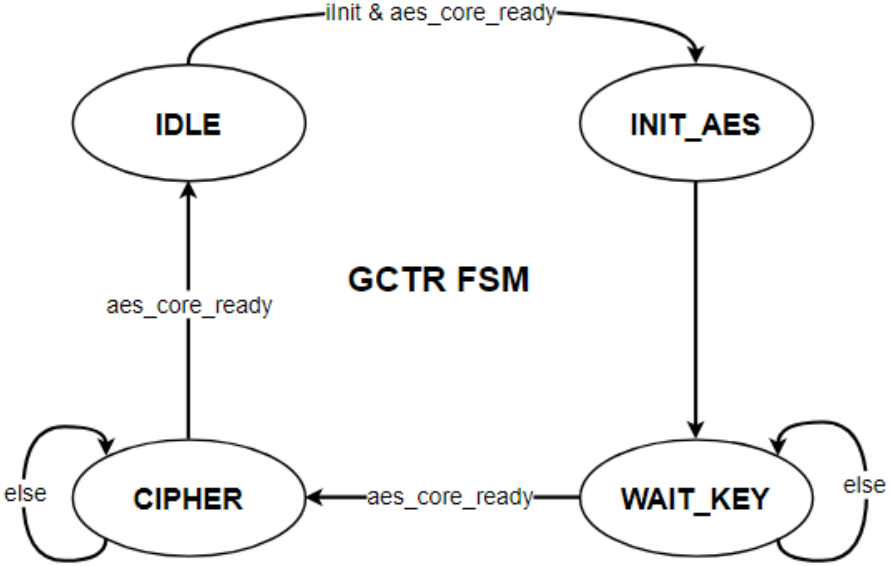


Submodule	File
GF_MUL	gfmul.v

Name	GHASH_BLOCK
File	ghash_block.v

STATE MACHINEs


IDLE
ghash_result_valid_rising
ctrl_reg[0] & next_rising
TAG
CAL_HASHKEY
else
AES_GCM FSM
iAad_valid & next_rising
gctr_result_rising
else
CIPHER
iBlock_valid & next_rising
CAL_AAD
else
ctrl_reg[3] & next_rising


iInit & aes_core_ready
IDLE
INIT_AES
GCTR FSM
aes_core_ready
else
CIPHER
aes_core_ready
WAIT_KEY
else

AES_GCM FSM

IDLE

```
gctr_init = 1'b0
gctr_hashkey_compute = 1'b0
gctr_y0_compute = 1'b0
ghash_next = 1'b0
ghash_input_signal = 2'b00
ghash_input_valid = 1'b0
ghash_key_wen = 1'b0
ghash_key_valid = 1'b0
ghash result_wen = 1'b0
y0_wen = 1'b0
```

CAL_HASHKEY

```
if(gctr_result_valid)
      gctr_init = 1'b0
else
      gctr_init = 1'b1
gctr_hashkey_compute = 1'b1
gctr_y0_compute = 1'b0
ghash_next = 1'b0
ghash_input_signal = 2'b00
ghash_input_valid = 1'b0
if(gctr_result_valid)
      gctr_key_wen = 1'b1
      gahsh_key_valid =
1'b1
else
      gctr_key_wen = 1'b0
      gahsh_key_valid =
1'b0
ghash_result_wen = 1'b0
y0_wen = 1'b0
```

CAL_AAD

```
if(next_rising &
(iBlock_valid |
ctrl_reg[3]))
      gctr_init = 1'b1
else
      gctr_init = 1'b0
gctr_hashkey_compute = 1'b0
gctr_y0_compute = 1'b0
ghash_next = next_rising
ghash_input_signal = 2'b01
ghash_input_valid =
iAad_valid
ghash_key_wen = 1'b0
ghash_key_valid = 1'b1
if(ghash_result_valid_rising
)
      ghash_result_wen =
1'b1
else
      ghash_result_wen =
1'b0
y0_wen = 1'b0
```

CIPHER

```
if(next_rising)
      gctr_init = 1'b1
else
      gctr_init = 1'b0
gctr_hashkey_compute = 1'b1
gctr_y0_compute = 1'b0
ghash_next =
gctr_result_rising
if(ctrl_reg[2])
      ghash_input_signal =
2'b00
else
      ghash_input_signal =
2'b10
ghash_input_valid =
gctr_result_valid
ghash_key_wen = 1'b0
ghash_key_valid = 1'b1
if(ghash_result_valid_rising
)
      ghash_result_wen =
1'b1
else
      ghash_result_wen =
1'b0
y0_wen = 1'b0
```

TAG

```
gctr_init = 1'b0
gctr_hashkey_compute = 1'b0
gctr_y0_compute = 1'b1
ghash_next =
gctr_result_rising
ghash_input_signal = 2'b01
ghash_input_valid =
gctr_result_valid
ghash_key_wen = 1'b0
ghash_key_valid = 1'b1
if(ghash_result_valid_rising
)
      ghash_result_wen =
1'b1
else
      ghash_result_wen =
1'b0

if(gctr_result_rising)
      y0_wen = 1'b1
else
      y0_wen = 1'b0
```

GCTR FSM

IDLE

counter_en = 1'b0

block_wen = 1'b0

IV_wen = 1'b0

key_wen = 1'b0

hashkey_wen = 1'b0

y0_wen = 1'b0

aes_core_init = 1'b0

aes_core_next = 1'b0

gctr_result_valid =

aes_core_output_valid

INIT_AES

counter_en = 1'b0

block_wen = 1'b1

IV_wen = 1'b1

key_wen = 1'b1

hashkey_wen = 1'b1

y0_wen = 1'b1

aes_core_init = 1'b1

aes_core_next = 1'b0

gctr_result_valid = 1'b0

WAIT_KEY

counter_en = 1'b0

block_wen = 1'b0

IV_wen = 1'b0

key_wen = 1'b0

hashkey_wen = 1'b0

y0_wen = 1'b0

aes_core_init = 1'b0

aes_core_next = 1'b0

gctr_result_valid = 1'b0

CIPHER

counter_en = 1'b1

block_wen = 1'b0

IV_wen = 1'b0

key_wen = 1'b0

hashkey_wen = 1'b0

y0_wen = 1'b0

aes_core_init = 1'b0

aes_core_next = 1'b1

gctr_result_valid = 1'b0

Multiplication in $GF(2^{128})$

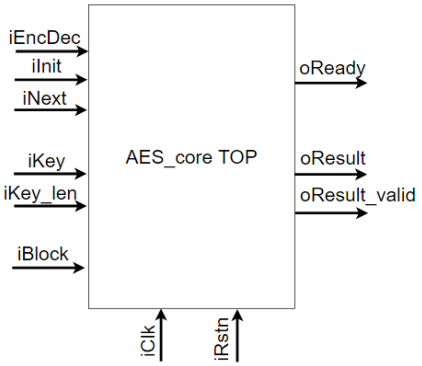
Each element is a vector of 128 bits. The i^{th} bit of an element X is denoted as X_i. The leftmost bit is X_i and the rightmost bit is X_{127}. The multiplication operation uses the special element $R = 1110001||0$, and is defined in Algorithm 1. The argument **rightshift()** moves the bits of its argument one bit to the right. More formally, whenever $W = \text{rightshift}(V)$, then $W_i = V_{i-1}$ for $1 <= i <= 127$ and $W_0 = 0$.

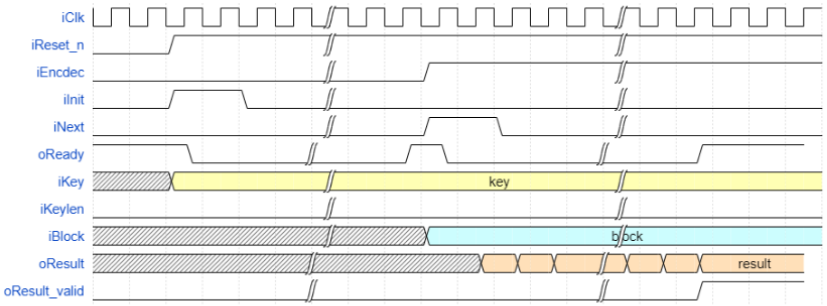
What we want to compute: oY = gf_mul(iHashKey, iCtext ^ iY)

Algorithm 1 Multiplication in $GF(2^{128})$. Computes the value of $Z = X \cdot Y$, where X, Y and $Z \in GF(2^{128})$.

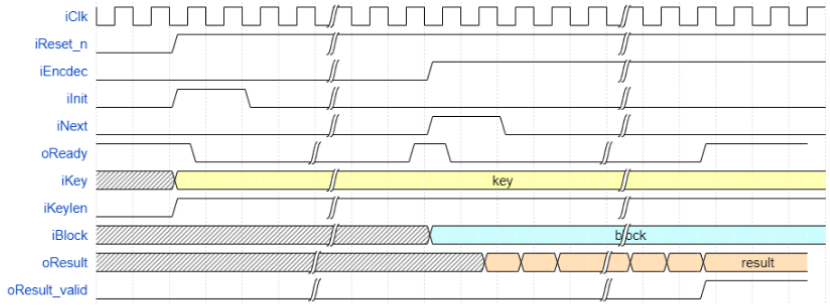
$Z \leftarrow 0, V \leftarrow X$
for $i = 0$ to 127 **do**
  if $Y_i = 1$ **then**
    $Z \leftarrow Z \oplus V$
  end if
  if $V_{127} = 0$ **then**
    $V \leftarrow \text{rightshift}(V)$
  else
    $V \leftarrow \text{rightshift}(V) \oplus R$
  end if
end for
return Z

URL: [tee-hardware/hardware/teehw/optvsrc/AES at master · uec-hanken/tee-hardware (github.com)]

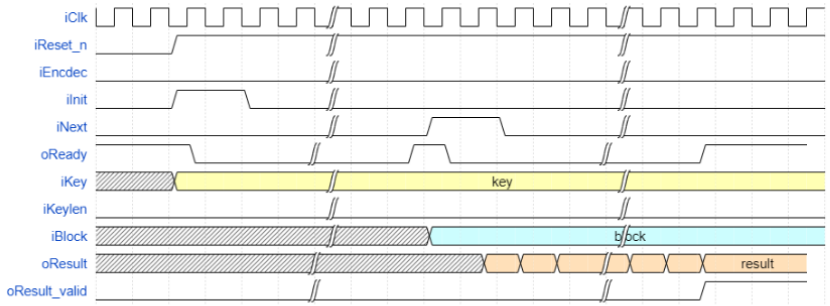




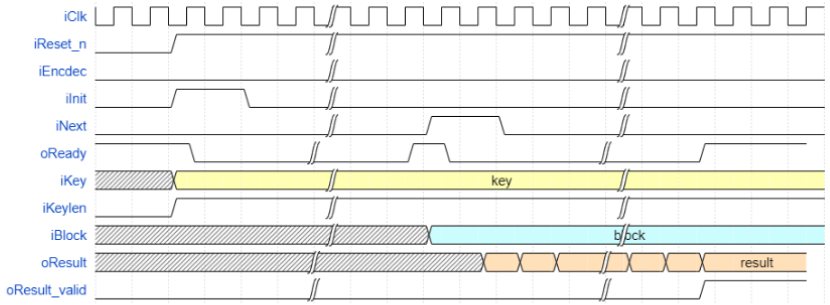
AES Encryption 128b Key



AES Encryption 256b Key



AES Decryption 128b Key



AES Decryption 256b Key