AES128 CORE SPECIFICATION

Revision History

Rev	Date	Author	Description
0.1	14/11/2024	Vincent Yang	First draft release

References

Title	Reference
Specification for the ADVANCED ENCRYPTION STANDARD (AES)	https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

Contents

List of figures

List of tables

1. Introduction

1.1. Principles and areas of application

This document provides a comprehensive specification for the implementation of the Advanced Encryption Standard (AES) algorithm in VHDL. The AES algorithm is a widely used symmetric encryption standard that ensures data security through a series of defined transformations. This specification outlines the design, functionality, and integration of the AES module.

This specification covers the following aspects of the AES encryption module:

- Detailed descriptions of all input and output signals.
- Finite State Machine (FSM) state transitions and operations performed.
- The process of generating round keys from the initial encryption key.
- Implementation details of core AES functions.

1.2. Features

The AES encryption module is designed to process 128-bit plaintext data using a 128-bit encryption key, producing 128-bit ciphertext. The module operates synchronously with a clock signal and includes a reset mechanism to initialize the system. The design ensures efficient and secure data encryption, adhering to the AES standard.

Features :

- Implements the AES-128 encryption and decryption standard as defined by the National Institute of Standards and Technology (NIST).
- Supports 128-bit plaintext and ciphertext data.
- Supports 128-bit encryption keys.
- Utilizes a FSM to manage the encryption and decryption process.
- Includes a key expansion function that generates round keys from the initial encryption key, supporting up to 11 round keys for AES-128.
- Implements SubBytes, ShiftRows, MixColumns and AddRoundKey functions.
- Operates synchronously with a clock signal.
- Includes a reset signal to initialize the module to a known state, clearing all internal signals and resetting the FSM.
- Start and finish signals to indicate AES operation beginning and end.
- Cipher input signal to do an encryption or decryption.

1.3. Hardware target

TODO

2. Overall description of AES128 module

2.1. Architecture

Entity aes128 is composed of one FSM acting as a control unit to manage AES execution steps.

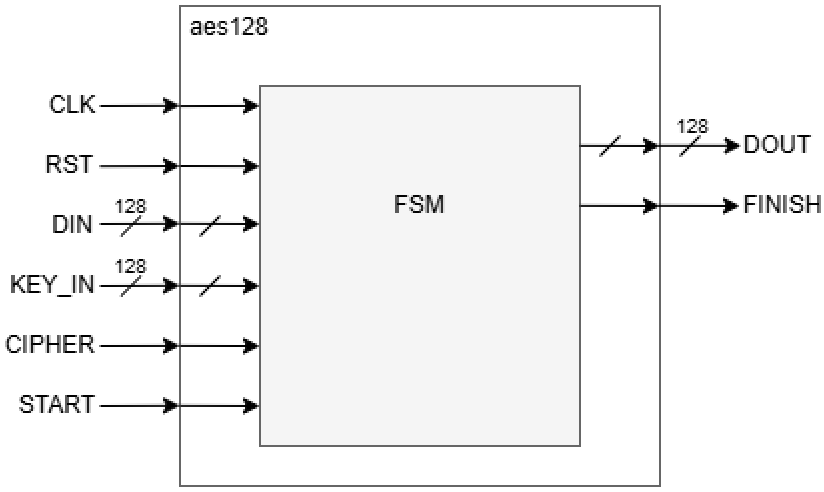

Figure 1: Block diagram of the AES128 core

2.2. FSM diagram

The FSM ensures that each stage of the encryption or decryption process is carried out in the correct sequence and manages the transitions between these stages. The FSM handles loading the initial key and the SubBytes, ShiftRows, MixColumns and AddRoundKey operations. It iterates through the necessary rounds of transformation with the round_counter internal signal.

2.2.1. Cipher

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])            // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                        // See Sec. 5.1.1
      ShiftRows(state)                       // See Sec. 5.1.2
      MixColumns(state)                      // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 2: Pseudo code for the Cipher

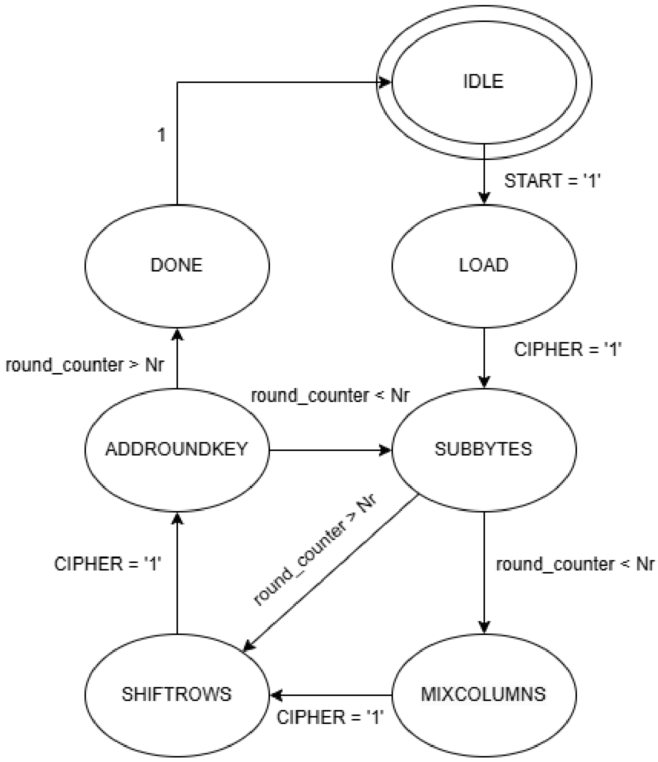

Figure 3: FSM diagram when ciphering

Figure 2 gives pseudo code for the cipher and Figure 3 gives the FSM used to cipher.

2.2.2. Decipher

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Figure 4: Pseudo code for the Decipher

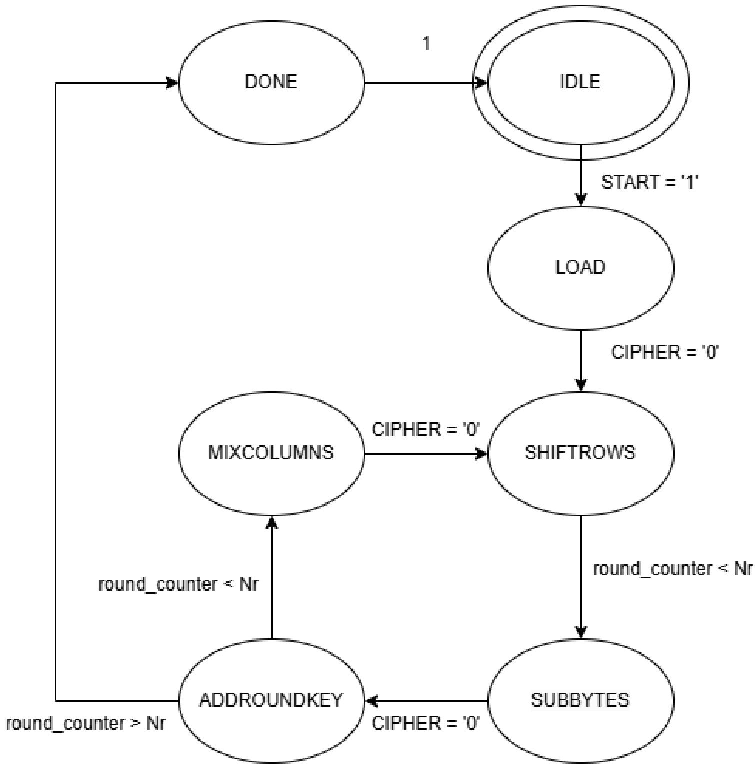

Figure 5: FSM diagram when dechipering

Figure 4 gives pseudo code for the decipher and Figure 5 gives the FSM used to decipher.

Notes that same states that cipher operation are used, the difference in the actions and transitions done by these states lies in current_phase signal value.

Current_phase is set to :

- CIPHER_PHASE when input signal CIPHER = 1.
- DECIPHER_PHASE when input signal CIPHER = 0.

2.2.3. FSM states

State	Action	Transition
IDLE	Resets finish, round_counter , status_array, and dout. If start is '1', sets current_phase based on CIPHER and moves to LOAD.	If start is '1', moves to LOAD. Else, stays in IDLE
LOAD	Loads the initial key and status array based on current_phase	If CIPHER_PHASE, moves to SUBBYTES. If DECIPHER_PHASE, moves to SHIFTROWS.
SUBBYTES	Applies the subbytes_f function to status_array.	If CIPHER_PHASE, increments round_counter, updates key_w, and moves to SHIFTROWS. If DECIPHER_PHASE, moves to ADDROUNDKEY
SHIFTROWS	Applies the shiftrows_f function to status_array.	If CIPHER_PHASE and round_counter is less than Nr, moves to MIXCOLUMNS. Else, moves to ADDROUNDKEY. If DECIPHER_PHASE, increments round_counter, updates key_w, and moves to SUBBYTES
MIXCOLUMNS	Applies the mixcolumns_f function to status_array.	If CIPHER_PHASE, moves to ADDROUNDKEY. If DECIPHER_PHASE, moves to SHIFTROWS.
ADDROUNDKEY	Applies the addroundkey_f function to status_array using key_w.	If round_counter is less than Nr, moves to SUBBYTES if CIPHER_PHASE, or to MIXCOLUMNS if DECIPHER_PHASE Else, moves to DONE
DONE	Outputs the final status_array to dout, resets key_w and expanded_ key, sets finish to '1'.	Moves to IDLE
OTHERS	Defaults to IDLE	Moves to IDLE

Table 1: FSM state action and transition

2.3. AES module specification

2.3.1. Data types

In this design, signal din[127:0] is converted into a 4x4 byte matrix format (defined type block_4x4_array) to handle AES operations more easily in SubBytes, ShiftRows etc. functions.

The functions vector_to_matrix and matrix_to_vector, represented in Figure 6, handle this conversion:

1. **vector_to_matrix**: Transforms the 128-bit input vector din[127:0] into a 4x4 matrix format (each matrix element being 8 bits) required for AES processing.
2. **matrix_to_vector**: Converts the 4x4 byte matrix back into a 128-bit vector after processing by AES functions, facilitating further data handling or storage.

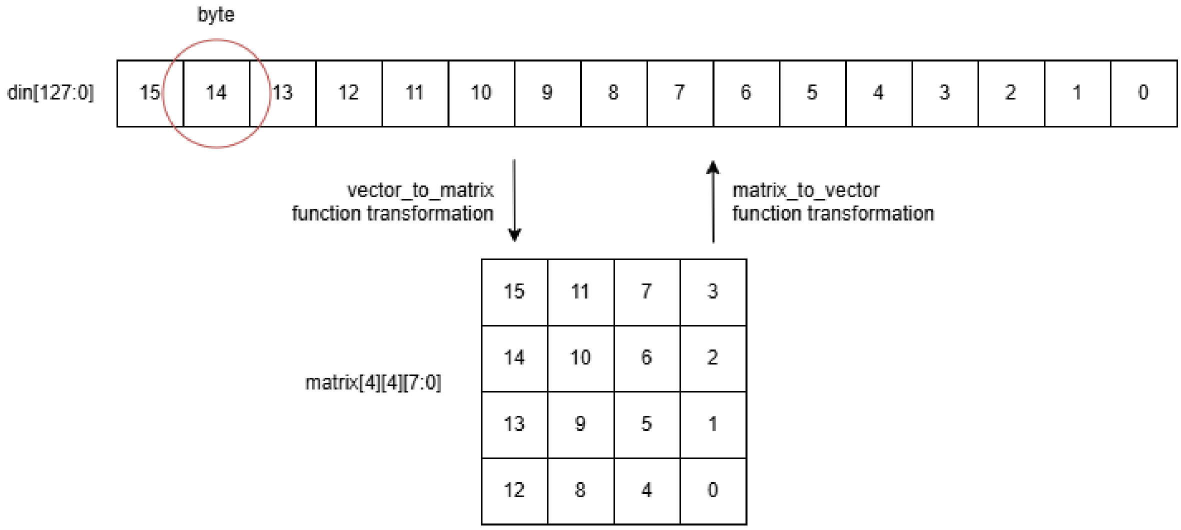

Figure 6: Vector and matrix conversion diagram

2.3.2. AES functions

Functions are defined library aes128_lib file.

Table 2 gives main AES functions used in the design.

Function	Purpose	Input	Output
subbytes_f	Substitutes each byte in matrix_in with a corresponding value from the sbox or inv_sbox constant.	matrix_in: 4x4 byte matrix phase: Specifies encryption/decryption	4x4 matrix with substituted values
shiftrows_f	Shifts rows of matrix_in left or right based on phase.	matrix_in: 4x4 byte matrix phase: Specifies encryption/decryption	4x4 matrix with shifted rows
mixcolumns_f	Mixes columns of matrix_in.	matrix_in: 4x4 byte matrix phase: Specifies encryption/decryption	4x4 matrix with mixed columns
addroundkey_f	Adds the round key to matrix_in by performing an XOR with key_w.	matrix_in: 4x4 byte matrix key_w: 128-bit round key	4x4 matrix after XOR with key_w
key_expansion	Generates round keys from the initial key key_in for each AES round.	key_in: Initial 128-bit encryption key	Expanded round key (1407 bits)

Table 2: Main AES functions name and purpose

2.4. Module interface (I/O ports)

Port	Width	Direction	Description
CLK	1	Input	Clock, actions are performed on its rising edge.
RST	1	Input	Synchronous reset, asserted when high. It resets the FSM and signals to its initial state.
START	1	Input	Start signal. When asserted, it indicates that the input data is ready to be processed. This signal triggers the beginning of the encryption or decryption process.
DIN	128	Input	Data input signal. It carries the 128-bit plaintext data that needs to be encrypted.
KEY_IN	128	Input	Key input signal. It carries the 128-bit encryption key used for the AES algorithm.
CIPHER	1	Input	Indicates which operation is performed. If set to 1, it's an encryption; if set to 0, it's a decryption
DOUT	128	Output	Data output signal. It carries the 128-bit ciphertext result after the encryption process is completed.
FINISH	1	Output	The finish signal. When asserted, it indicates that the encryption process is complete and the output data is valid.

Table 3 : I/O Interface

3. Testbench

A simple directed testbench aes128_tb.vhd has been made with plain text and key examples from the AES specification. Complete verification has been made in a SV/UVM test environment.

Figure 7 and Figure 8 give start and end of an encryption operation. Start of operation is signaled with the assertion of start signal. End of operation is signaled with the assertion of finish signal.

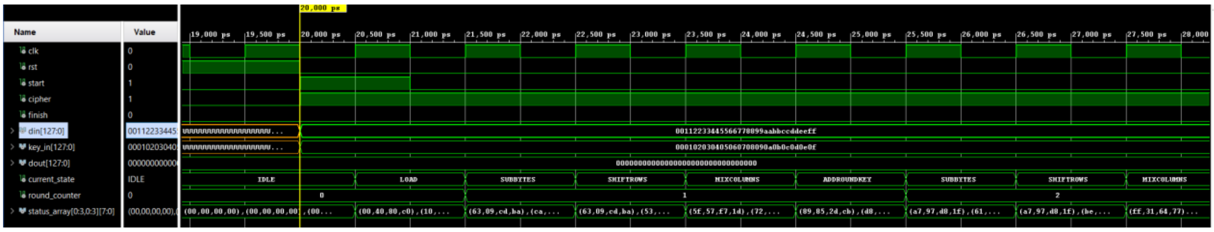

Figure 7: AES start of encryption waveform

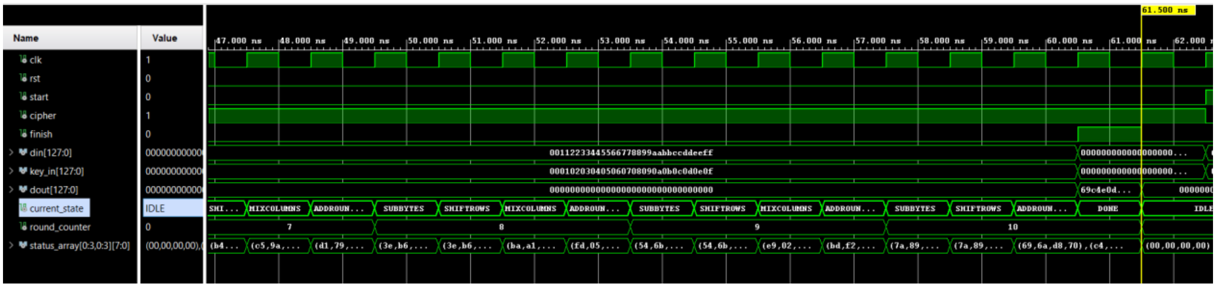

Figure 8: AES end of encryption waveform

One operation (encryption or decryption) takes 42 clock cycles to be executed:

- 1 clock cycle to initializes the FSM, load the plain text and key.
- 40 clock cycles to execute round 0 to 10.
- 1 clock cycle to output the cipher text.

Appendix

N/A