**AES offload CPS**

# Major Characteristics

The AES offload engine IP core implements an AES encryption module that conforms to the IEEE 1619 specification.

The AES offload engine will use 128-bit AXI interfaces.

Some auxiliary blocks will be developed inline with the AES offload engine. Such as a configuration block and a Miscellaneous(misc) block containing information about the build as well as clock and reset management.

## Main Features

* Implements an AES (Advanced encryption standard) offload engine conforming to the IEEE 1619 standard for 2 modes ECB and GCM
* Implements an Amba AXI4 interface
* Supports configuration and status checks to and from the CPU over the AXI bus
* Supports communication between a ZYNQ processor over the AXI bus

## Main functionality

The AES offload engine implements the blocks shown within the grey on the Programmable logic (PL) side. But have included details for the Misc block.

Diagram

Description automatically generated

### AES Block

* The encryption methods will support AES128/192/256
* The AES block has modes that can be selected ECB, GCM. This gets processed by the config block in section **1.2.3**
* The AES block’s data rate is configurable with Hi = 25.6Gb/s and Lo = 1Gb/s via a generic. This is an estimated figure depending on timing for a 200 Mhz clock

Below is the functionality of the AES ECB encryption standard that will be used or See slide 2 [Top.pptx](https://entrustdatacardcorp-my.sharepoint.com/:p:/g/personal/jack_okeefe_entrust_com/EQCdKEnFb-1NifCWGI_XPMoBRO10fIXlp9oShY8OpvgPwQ?e=2Fw7fW)

A picture containing graphical user interface

Description automatically generated

#### ECB

* The encryption performs various functions which are done over a series of rounds depending on the mode
  + AES128 = 12 rounds
  + AES192 = 14 rounds
  + AES256 = 16 rounds
* The functions are shown below:
  + Add Key: bitwise XOR operation of the input state bits and the bits of the current Expanded key (128 bits / round + initial round)
  + Substitute Bytes: performs a byte-wise substitution of the state using a substitution box (S-box).
    - S-box uses indexing according to its row and column number. Below is a miniature example but in reality, it will contain 256 bytes. Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | 63 | 7C | 77 | 7B | F2 |
| 1 | CA | 82 | C9 | 7D | FA |
| 2 | B7 | FD | 93 | 26 | 36 |

* + - The MSB’s represent the row value and the LSB bits represent the column value so with an input value of 04 = F2
    - Decryption S-box uses the inverse of this
  + Shift Rows: Each byte of a row of the state is cyclically shifted to the left by the index of the row (zero-based). Therefore, the first row does not change, the bytes of the second row are rotated one byte to the left, and so on.
  + Mix columns: This operation can be understood as a column-by-column multiplication (this function is omitted for the last round)
* Each round before the add key operation the data will be registered to create a pipelined design.
* Decryption is the same process but with each step inversed except the add round key this stays the same and the cipher text is the input
* AES parameters

Table

Description automatically generated

#### GCM

* GCM incorporates AES encryption method with some changes to improve the security

Diagram

Description automatically generated

* As can be seen in the above figure. The authenticated encryption operation takes Initialization Vector (IV) 96bits, Additional Authenticated Data (AAD) which is sometimes 128bit of zero’s, secret key and plaintext as an input in 128-bit and gives a 128-bit ciphertext and XOR’s with the authentication tag. The AES-GCM algorithm encrypts or decrypts with 128-bit, 192-bit or 256- bit of cipher key
* The hash key is generated by encrypting a block of all zeros with the secret key K. In step 2, the pre-counter block (J0) is generated from the IV. When the length of the IV is 96 bits, then the padding string 0^31 || 1 is appended to the IV to form the pre-counter block. Otherwise, the IV is padded with the minimum number of 0 bits, possibly none, so that the length of the resulting string is a multiple of 128 bits (the block size); this string in turn is appended with 64 additional 0 bits, followed by the 64-bit representation of the length of the IV, and the GHASH function is applied to the resulting string to form the pre-counter block. Thus, GCM is based on the CTR mode of operation and adds a MAC that authenticates both the message and additional data that requires only authentication.
* Description of GHASH: Let X be a concatenation of unencrypted authenticated data, CTR-encrypted ciphertext, and padding. This data is split into m 128-bit blocks Xi : X = X1 || X2 || · · · || Xm. AES is used to derive the root authentication key H = EK(0). The same AES key K is also used as the data encryption key. H is unknown to the attacker as the scheme would be otherwise trivially breakable. GHASH is A picture containing clock

  Description automatically generatedbased on operations in the finite field GF(2^128).

The authentication tag is T = Ym + EK(IV || 0 31 || 1), assuming that a 96-bit Initialization Vector (IV) is used. The IV value must never be repeated as that would lead to the “forbidden attack”

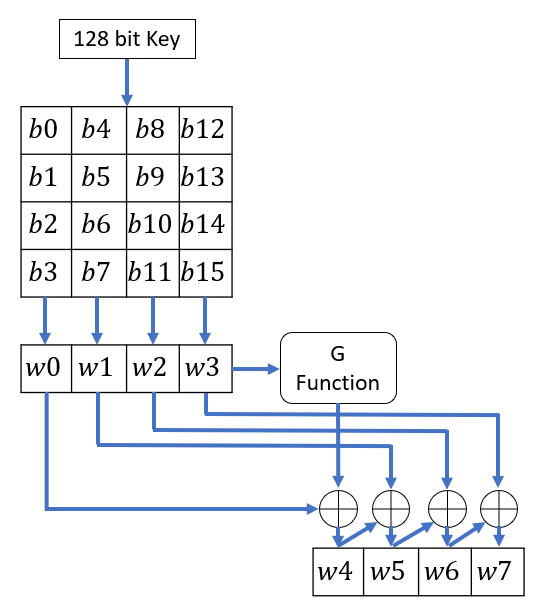
### Key handling

* The keys will be stored in plain text form inside a BRAM large enough to store 10 thousand keys.
* The Zynq will send a handle to access a certain key within the BRAM
* There will be an even number of keys stored for each AES key size. 3333 keys.

### Key expansion

The key Expansion block generates keys from the root key that can be used across all rounds.

#### Key Expansion

* Figure 1 shows the key expansion behaviour
* The Key expansion block **shall**produce a linear array of 44 words (4 bytes = 1 Word) needed for the number of rounds (Nr) + 1.
* The 128-bit input key **shall** be arranged into a 4x4 matrix which can be seen below.
* Each column **shall**be arranged into 4 words.
* First 4 words **shall**be used for the pre round 'AddRoundKey' function
* Column 4's word is passed into the G function. See Section 1.2.3.2
* Post G function, the key **shall**be subject to bitwise XOR with the G function output and 3 words of the expanded key w4-6.
* The resulting output w4-w7 **shall**be used for the first round.
* This behaviour will continue for each round. In this case 9 more rounds, see Table 1 for result.
* Table 1: Round output words

|  |  |
| --- | --- |
| Round | Word |
| Pre round | w0    w1    w2    w3 |
| 1 | w4    w5    w6    w7 |
| 2 | w8    w9    w10    w11 |
| …… | …… |
| 10 | w40  w41   w42   w43 |

#### G Function

* The G function **shall**take an input word Wi+3, i represents Nr \* 4.
* The word **shall**be shifted left once
* The 4 shifted bytes **shall** be substituted using the S-box substitution method.
* The resultant bytes **shall** then be XOR'd with the Round constant (RCj) Word.
* The RCj part or the most significant Byte Changes each round, the rest of the word is always 0's. See Table 2 for RCj values.
* Table 2: Round Constant values

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| RC[j] | 01 | 02 | 04 | 08 | 10 | 20 | 40 | 80 | 1B | 36 | 6C | D8 | AB | 4D | 9A | 2F |

* The G function **shall**output 1 word which will be XOR'd with Wi.

### AXIS config block

* This block routes and controls all the data being fed into and out of the AES engine
* This block controls the mode selected
* This block shalluse AXI Stream interface to send and receive data between the AES engine and the FIFO’s.
  + Using the tvalid, tlast and tready signals, the block will control data transfer
* This block shall output status information to the AXI registers block
  + Status includes
    - Error
    - Done
    - Mode
    - Speed (Hi or Lo)
* This block shall receive control data from the AXI stream Tdata word 1 and 2 after Tlast falling edge
* The control data will be extracted using a statemachine or If statement. These are then applied to the core
  + Config data consists of
    - Mode
    - Initial Value
* Config logic
  + Diagram

    Description automatically generated

### AXI ctrl registers

* This block is located inside the config block and contains read registers containing the status and any other necessary information regarding the AES offload engine
* This block contains 32-bit registers with 8-bit granularity, big endian (JC).
* This block interfaces with the Zynq processor to read information from the AES engine.
  + - Status
    - mode

### MISC

The Miscellaneous block contains the clock and reset management block it also contains the identification block.

#### Clock and reset management

* The reset management will ensure the reset is held high long enough to set all registers back to their defined initial state and flush out any pipelines in the design
* The {**i\_sys\_reset**} input will enter the device asynchronously and be synchronised in the device
* The {**sys\_reset**} signal will be de-asserted after a defined number of clock cycles (TBD) to ensure a proper reset
* The clock management will explicitly tell the tool that the clock signal is a global clock using the general clock buffer (BUFG) from the Xilinx HDL library.

#### Identification (ID)

* The ID block will contain information related to the build
* The ID block contains read only registers which will be available to the processor via AXI
* The ID contains the following information
  + Vendor
  + Vendor info/project name
  + Version
  + Date
* The ID block will be updated every time implementation is run through the build script