**AES Engine CPS**

# Major Characteristics

## Main Features

* Implements an AES (Advanced encryption standard) offload engine conforming to the IEEE 1619 standard for 2 modes ECB and GCM
* Supports compatibility with Amba AXI4 interface
* Supports configuration from the CPU over the AXI bus
* Supports a periodic self-test
* Supports encryption and decryption for all modes
* The core will utilise the devices distributed LUT rams, and self-test mode to defend against side channel attacks

## Main functionality

Diagram

Description automatically generatedFor context, the AES engine will be used within the system shown below.

### AES Engine

* Graphical user interface, application, Word

  Description automatically generatedBelow is a block diagram showing the steps that will be used for the encryption and clock crossings
* The encryption methods will support AES128/192/256 which is configurable using a generic
* 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 = 57.6Gb/s and Lo = 4.0Gb/s approx. via a generic. This is an estimated figure using a clock rate of 450 MHZ
* The data will pass through input and output asynchronous FIFO’s as 128-bit AXI-Stream
* Each round of encryption and the key expansion will be pipelined to meet timing requirements
* The engine will carry out self-testing using a key of all 0’s at a pre-defined period to test the engine is functioning correctly and to avoid temperature fluctuations which can be used to tell an attacker the system is encrypting data.
* The following AXIS signals will be utilised:
  + TKEEP, this will pad out any unused bytes with 0 so the data going to the engine is still in 128-bit blocks
  + TREADY, this will be used by the engine to tell the master it is ready to receive data
  + TVALID, will be used to signal new and valid data which will be utilised for the configuration data
  + TLAST, will be utilised to signify when the final block of data arrives
* The engine will process the key expansion logic **1.2.2.2**

#### Modes

The Engine will support GCM and ECB mode which will be set by the processor using one of the fields in the configuration data. Two additional modes will also be supported called run-through and self-test.

##### ECB

* ECB mode provides encryption and decryption
* The ECB mode supports the following inputs:
  + Plain text (PT): 128-bit blocks
  + Key: 128/192/256-bit widths
* It will support the following outputs:
  + Cipher text (CT): 128-bit blocks

##### GCM

* GCM mode provides encryption, decryption and authentication.
* GCM incorporates AES encryption method with some additional measures to improve the security and authentication abilities
* The GCM mode supports the following inputs:
  + Plain text (PT): 128-bit blocks
  + Additional authenticated data (AAD): 262140 bytes
  + Initialisation vector (IV): 96 bits
  + Key: 128/192/256-bit widths
* It will support the following outputs:
  + Cipher text (CT): 128-bit blocks
  + Tag: 128, 120, 112, 104, 96, 64 and 32 bits

##### Run-through

* The run-through mode is used to send data through the engine without any encryption

##### Self-Test

* The self-test mode provides a method to test and self-check the engines integrity and functionality on a periodic basis. If an error occurs this will be visible in the status register.
* The self-test will make use of the reserved key address in the BRAM which will be address 0x000
* It can be used to keep temperature and power fluctuations to a minimum which could be used in a side channel attack.

### Side channel attack defence methods

* Critical elements such as S-boxes can be realized as Distributed LUTRAM that are dynamically randomized to scramble to content.
* A close-up of some writing

  Description automatically generated with low confidenceAs stated in section **1.2.1.1.4** the self-test mode can be used to keep the power consumption constant and make it more difficult to conduct any power analysis.
* A second encryption engine could be used running on the falling edge clock to reduce the ability of the attacker to decipher what encryption algorithm is being used. For example, this is a simple power analysis of AES128:

If two engines were running one on each clock edge, then this voltage graph would look more like a straight line.

* NOTE: Configurable Look up tables (CFGLUT) may be utilised to enable fast and dynamic logic reconfiguration during runtime which help against reverse engineering and HTH insertion. The CFGLUT and the FPGA bitstreams will be stored in separate secure memory’s, this is known as obfuscation to prevent an attacker reverse engineering the FPGA.

### Key

The Key BRAM is an asynchronous dual port ram containing keys that will be used during encryption

#### Key handling

* The keys will be stored in plain text form inside a BRAM large enough to store 1000 keys.
* The keys can be written by the processor over the AXI bus
* 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.
* It will contain the user keys and a key of all 0’s at the first address which will be used for self-testing mode.
* The keys stored by the processor will be monitored by software to know the length of time they have been used for and each key will have an expiry date. (This is not handled by the FPGA as this can be done in software very easily).

#### Key expansion

* The key Expansion block generates keys from the root key that can be used across all rounds.
* Key expansion shall support 128/192/256-bit keys
* The key expansion shall utilise pipelining for each round of key expansion in order to meet the timing requirements.

#### Key change

* The engine supports key changes mid operation
* When a key change is detected, the engine will allow the current pipelined encryption process to finish before allowing the new key and data to be used for the next encryption cycle. It does this using the TREADY signal. This takes a pre-defined number of clock cycles depending on the current mode:
  + 128 = 21 clock cycles
  + 192 = 23 clock cycles
  + 256 = 25 clock cycles

### Configuration block

The configuration makes use of AXIS signals and a predefined set of fields to configure the engine

|  |  |  |
| --- | --- | --- |
| 128 bits | | |
| 0’s(127:100) | IV(99:3) | MODE(2:0) |

* The configuration block takes the first 128bit block of data when rising edge of TVALID and extracts the fields from it which then gets applied to the engine. The field can be seen below:
* This block shall output status information into a readable register, it includes:
  + Error during self test

### MISC

Miscellaneous 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
* The {**i\_sys\_reset**} input will enter the device asynchronously and be synchronised in the device using 2FF
* The {**sys\_reset**} signal will be de-asserted synchronously after 3 clock cycles. Synchronous Resets are present at each pipeline stage. If they weren’t then the reset would be held high for longer than 3 clock cycles

#### 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