AES-P

Programmable Mode AES Encrypt/Decrypt Core

The AES-P encryption IP core implements Rijndael encoding and decoding in compliance with the NIST Advanced Encryption Standard. It processes 128-bit blocks, and is programmable for 128-, 192-, and 256-bit key lengths.

Two architectural versions are available to suit system requirements. The **Standard** version (AES-P-S) is more compact, using a 32-bit datapath and requiring 44/52/60 clock cycles for each data block (128/192/256-bit cipher key, respectively). The **Fast** version (AES-P-F) achieves higher throughput, using a 128-bit datapath and requiring 11/13/15 clock cycles for each data block. It can be programmed to use any of the following cipher modes: ECB, CBC, OFB, CFB and CTR.

The core works with a pre-expanded key, or with optional key expansion logic.

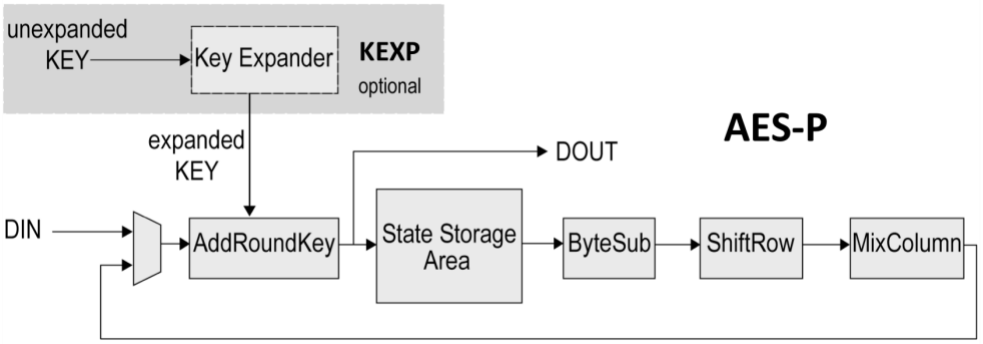
The AES-P cores are fully synchronous designs, have been evaluated in a variety of technologies, and are available optimized for ASICs or FPGAs.

Applications

The AES-P can be utilized for a variety of encryption applications including:

- Protected network routers
- Electronic financial transactions
- Secure wireless communications
- Secure video surveillance systems
- Encrypted data storage

Block Diagram



FEATURES

- Encrypts and decrypts using the AES Rijndael Block Cipher Algorithm
- Implemented according to the Federal Information Processing Standard (FIPS) Publication 197 from the US National Institute of Standards and Technology (NIST)
- NIST Certified
- Processes 128-bit data in 32-bit blocks
- Employs user-programmable key size of 128, 192 or 256 bits
- Two architectural versions:
  - Standard is more compact:
    32-bit data path size
    Processes each 128-bit data block in 44/52/60 clock cycles for 128/192/256-bit cipher keys, respectively
  - Fast yields higher transmission rates:
    128-bit data path
    Processes each 128-bit block in 11/13/15 clock cycles for 128/192/256-bit cipher keys, respectively
- Supports Cipher Block Chaining (CBC), Cipher Feedback (CFB), Counter (CTR), Electronic Codebook (ECB) and Output Feedback (OFB) modes
- Works with a pre-expended key or can integrate the optional key expansion function
- Simple, fully synchronous, reusable design
- Available as fully functional and synthesizable VHDL or Verilog, or as a netlist for popular programmable devices
- Complete deliverables include test benches, C model and test vector generator

Functional Description

An AES encryption operation transforms a 128-bit block into a block of the same size. The encryption key can be chosen among three different sizes: 128, 192, or 256 bits. The key is expanded during cryptographic operations.

The AES algorithm consists of a series of steps repeated a number of times (rounds). The number of rounds depends on the size of the key and the data block. The intermediate cipher result is known as state.

Initially, the incoming data and the key are added together in the AddRoundKey module. The result is stored in the State Storage area.

	KSIZE = 00	KSIZE = 01	KSIZE = 10
Rounds	10	12	14

Number of rounds as a function of key size

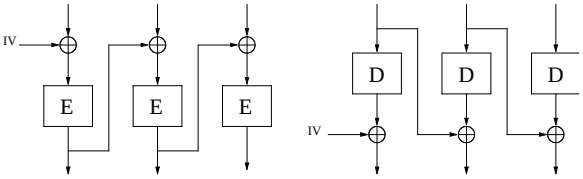
The state information is then retrieved and the ByteSub, Shiftrow, MixColumn and AddRoundKey functions are performed on it in the specified order. At the end of each round, the new state is stored in the State Storage area. These operations are repeated according to the number of rounds. The final round is anomalous as the MixColumn step is skipped. The cipher is output after the final round.

Supported modes

This AES-P core supports both encryption and decryption in ECB, CBC, CFB, OFB and CTR modes. ECB stands for Electronic CodeBook mode. This is the basic AES algorithm as described in the FIPS 197 documentation.

CBC stands for Cipher Block Chaining mode. Chaining adds a feedback mechanism to a block cipher. The result of the previous encryption operation is XORed with the incoming data. An initialization vector IV is used for the first iteration. Decryption reverses encryption operations.

The figure below shows the data flow during encryption (left) and decryption (right) in CBC mode.

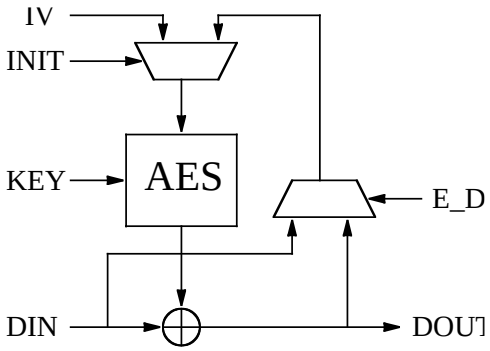


Encryption and decryption data flow in CBC mode

CFB stands for Cipher FeedBack mode. In this mode, the output of an encryption operation is fed back to the input of the AES core. An initialization vector IV is used for the first iteration.

Input data is encrypted by XORing it with the output of the encryption module. Decryption reverses encryption operations.

The figure below shows the block diagram of the AES in CFB mode.

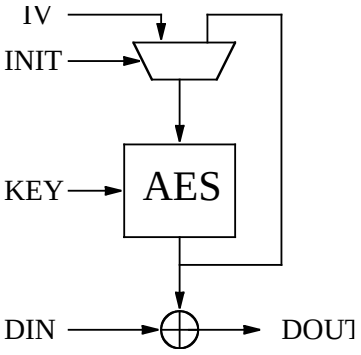


AES CFB core block diagram

OFB stands for Output FeedBack mode. In this mode, the output of an encryption operation is fed back to the input of the AES core. An initialization vector IV is used for the first iteration.

Input data is encrypted by XORing it with the output of the encryption module. Decryption reverses encryption operations.

The figure above right shows the block diagram of the AES in OFB mode.

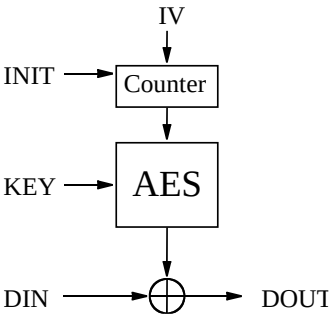


AES OFB mode block diagram

CTR stands for Counter mode. In this mode, the output of counter is input of the AES core. An initialization vector IV is used to initialize the counter.

Input data is encrypted by XORing it with the output of the encryption module. Decryption reverses encryption operations.

The figure below shows the block diagram of the AES in CTR mode.



AES CTR mode block diagram

Key Expansion

The AES algorithm requires an expanded key for encryption or decryption. The KEXP AES key expander core is available as an AES-P core option.

During encryption, the key expander can produce the expanded key on the fly while the AES-P core is consuming it. For decryption, though, the key must be pre-expanded and stored in an appropriate memory before being used by the AES-P core. This is because the core uses the expanded key backwards during decryption.

In some cases a key expander is not required. This might be the case when the key does not need to be changed (and so it can be stored in its expanded form) or when the key does not change very often (and thus it can be expanded more slowly in software).

Implementation Results

The AES-P can be mapped to any Intel FPGA device (provided sufficient silicon resources are available). The following are sample Intel results with all core I/Os assumed to be routed on-chip and throughput for a 128-bit key size.

AES-P Standard Core Intel Implementation Results

Family	Logic	RAM bits	Freq. (MHz)	Throughput (Mbps)
Arria 10 GX (-2)	756 ALMs	32	80	233
Stratix V (-1)	735 ALMs	0	100	291
MAX 10 (-7)	2,421 LEs	16	50	145

AES-P Fast Core Intel Implementation Results

Family	Logic	RAM bits	Freq. (MHz)	Throughput (Mbps)
Arria 10 GX (-2)	2,679 ALMs	0	80	931
Stratix V (-1)	2,763 ALMs	0	100	1,164
MAX 10 (-7)	8,432 LEs	0	50	582

The provided figures do not represent the higher speed or smaller area for the core. Please contact CAST to get characterization data for your target configuration and technology.

Related Products

The related KEXP key expander core is available as an option.

AES in CBC, CCM, CFB, CTR, ECB, GCM, LRW, OFB and XTS modes are also available as stand-alone cores.

Export Permits

This core implements encryption functions and as such it is subject to export control regulations. Export to your country may or may not require a special export license. Please contact CAST to determine what applies in your specific case.

Support

The core as delivered is warranted against defects for ninety days from purchase. Thirty days of phone and email technical support are included, starting with the first interaction. Additional maintenance and support options are available.

Verification

The core has been verified through extensive synthesis, place and route and simulation runs. It has also been embedded in several products, and is proven both in ASIC and FPGA technologies.

Deliverables

The core is available in ASIC (RTL) or FPGA (netlist) formats, and includes everything required for successful implementation. The Intel version includes

- Targeted Intel FPGA netlist
- Sophisticated HDL Testbench (self-checking)
- C Model & test vector generator
- Simulation script, vectors & expected results
- Synthesis script
- User documentation