algotronix
130-10 Calton Road
Edinburgh EH8 8JQ
Phone: +44 131 556 9242

E-mail: cores@algotronix.com

URL: www.algotronix.com

AES-XTS Core for Data Storage Applications

Algorithm	**AES**
Modes	XTS
Test Method	NIST AESAVS
Clock Cycles/Encryption	Selectable
Performance/Area	Tradeoff via compilation options.
Deliverables	VHDL with testbench. Optimizations for Xilinx FPGAs.

Features

- Complies with IEEE 1619-2007 and NIST SP800-38E standards
- Performance selectable to meet or exceed USB 3.0 and SATA 3.0 (6 Gbps), even on low cost FPGA families
- Low area, implementation of AES-XTS suitable for data storage applications.
- Based on the NIST validated (Cert #953) AES-G3 implementation of FIPS 197 (November 2001) Advanced Encryption Standard
- Supports 128 bit keys as standard, with 192 and 256 bit key options available
- Targets all modern FPGA families from Xilinx.
- Supplied as easily customizable portable VHDL or Verilog to allow customers to conduct their own code review in high-security applications.
- Supplied with comprehensive test bench implementing XTSVS tests.

Potential Applications

- Government/Military Encrypted Data Storage
- SATA 3.0 compliant storage
- USB 3.0 compliant storage
- Encrypted disk drives
- Encrypted Solid State Disks for Servers
- Hardware encrypted Memory stick
- Proprietary Security applications

General Description

The AES-XTS core is a high performance pipelined implementation of IEEE 1619. The AES core is optimized for encryption of data storage devices and can use low cost FPGA families as well as advanced devices. This product can be parameterized to deliver gigabit throughput where required, or can be tuned by the customer to provide a minimum footprint where lower performance is required. This allows it to meet the throughput requirements of SATA 3.0 or USB 3.0 connected storage.

The Algotronix AES-XTS Core implements the XEX Tweakable Block Cipher with Ciphertext Stealing Counter mode of operation of the AES algorithm. This mode of operation is described in NIST Special Publication SP800-38E and is used in the IEEE 1619 - 2007 standard for encrypted data storage. The XTS algorithm is a subcase of Rogaway's XEX (Xor Encrypt Xor) tweakable cipher design. Earlier drafts of IEEE-1619 used a different variant of AES called

AES-LRW but this was dropped due to concerns raised in the review process. The terms AES-XTS and XTS-AES are interchangeable and refer to the same algorithm. Algotronix uses AES-XTS in order to keep a consistent naming scheme across its AES product range.

AES-XTS is tailored for the particular requirements and constraints of storage encryption and is significantly more complex than the simple standard modes of AES such as Cipher Block Chaining (CBC) which are provided by the AES-G3 product. A design constraint for the AES-XTS algorithm was that ciphertext had to be exactly the same length as the plaintext so that encryption of data on a disk could be done 'in place' with the ciphertext simply overwriting the plaintext. This made it impossible to provide strong authentication since adding a hash or integrity check value (as is done in AES-CCM and AES-GCM) would make the ciphertext longer than the corresponding plaintext.

The AES-XTS algorithm therefore represents a compromise which provides data confidentiality and allows for parallel implementation for performance. However, unlike AES-CCM and AES-GCM (which are also available from Algotronix), it does not provide strong authentication and cannot detect and report tampering with the encrypted data. Instead, it ensures that if the encrypted data is tampered with it is very likely to decrypt to random values unrelated to the original plaintext. Encrypting the same data on two different data-units (e.g. disk sectors) of the storage medium will result in completely different ciphertext. This prevents an attacker making targeted alterations to data on the storage medium by copying portions of another encrypted file with known values over it or guessing the plaintext by comparing the ciphertext to that created by encrypting a known file.

The AES-XTS core is supplied with a testbench which can generate random test vectors or read vectors in the file format specified in the NIST XTS Verification System (XTSVS) document and used by NIST approved test laboratories for qualification. Files containing standard known answer tests from the NIST documents are supplied. The AES-XTS testbench operates in regression mode to verify any changes you may make to the source code or can be used to process vector files supplied by a NIST approved test lab for validation purposes.

The Algotronix AES-XTS core is supplied as VHDL source code and can be configured using a number of VHDL generic parameters to select only those features which are required in order to conserve area. The synthesisable core can also be supplied in Verilog on request. The core can be configured as Encryptor, Decryptor or Encryptor/Decryptor and the maximum key length can also be selected. The core provides hardware key schedule generation. This level of flexibility makes it easy to experiment with area/performance/functionality tradeoffs and makes it likely that the core will be useful in multiple projects.

The AES-XTS core is an easy to use fully synchronous design with a single clock and an enable signal to allow the core to be started and stopped on a clock cycle by clock cycle basis to match up with external data sources. The core has been designed for efficiency in modern FPGAs and makes full use of FPGA specific features such as dual port memory blocks.

Performance and Area

The core provides many options to allow the user to trade off area against throughput and latency and the FPGA architecture and speed grade also has a strong bearing on the results achieved. In addition, both the core itself and the FPGA manufacturer design tools are regularly updated and this affects area and maximum clock frequency results.

For these reasons, rather than provide area and performance information in the data sheet Algotronix prefers to generate these estimates on demand for our customers. Simply contact us with the desired throughput, target FPGA family and speed grade, and we will work out the best core configuration and use the latest FPGA design tools and core source code to provide estimates of the attainable clock frequency and area in the required configuration.

Functional Description

This implementation of XTS-AES is designed to offer megabit through gigabit level performance on low cost and high performance FPGA devices through the use of pipelining.

The AES algorithm consists of an iterative application of a processing round, with a 128-bit key there are 10 rounds of processing required (14 rounds are required with a 256-bit key) so the basic throughput is 1 block in 10 clock cycles. When configured with two round processing units this core 'unrolls' the inner loop using multiple processing units so that two round operations can take place simultaneously. In the example shown in Figure 1 with two pipeline stages, there is a latency of 10 clock cycles to process an AES block and the throughput is one AES block every 5 clock cycles.

The core can also, optionally, include an additional pipeline register within each round processing unit. This inner loop pipelining breaks the critical timing path and allows the core to run at a higher clock frequency than would otherwise have been the case. With this pipelining, latency increases to 20 clock cycles and throughput remains at 5 clock cycles per block. Although throughput in terms of clock cycles per block is unchanged performance is increased because the clock frequency is higher. The inner loop pipelining option is particularly valuable on low cost or low power FPGA architectures with relatively slow interconnect and combinational logic.

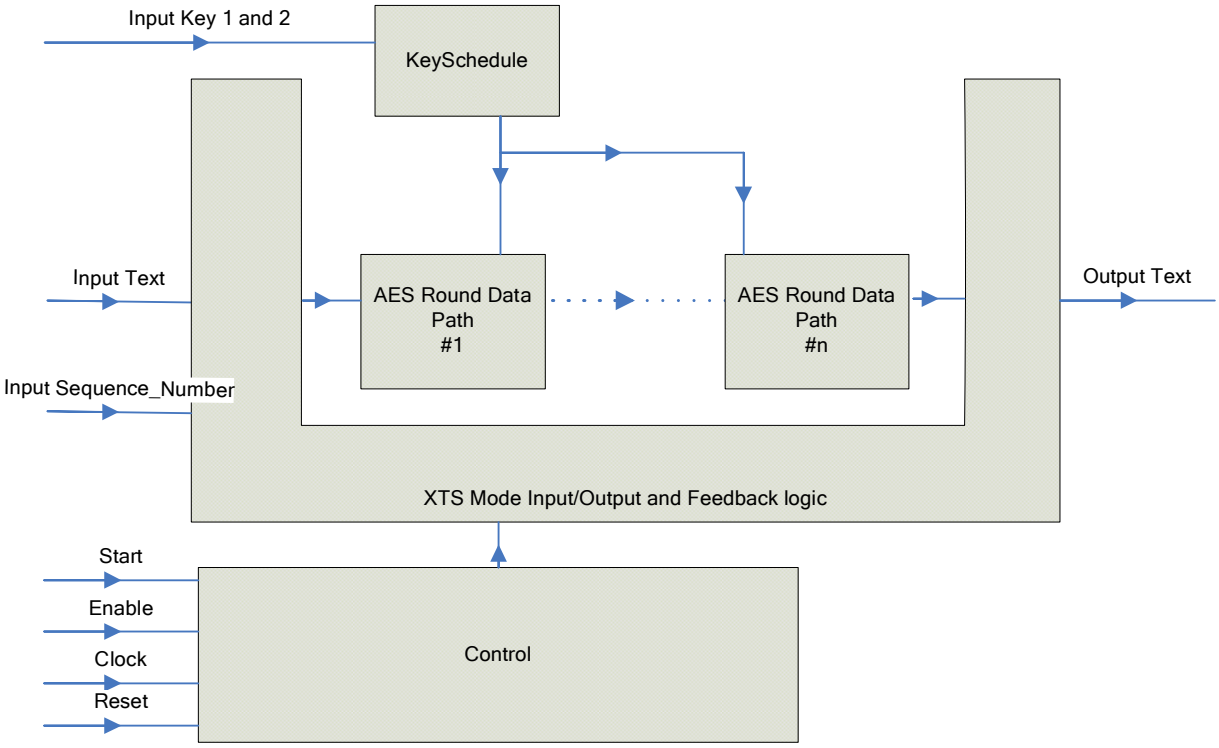

Figure 1, AES-XTS Encryptor Block Diagram

Compilation Options

The core can be configured easily using a set of VHDL generic parameters. Normally, it is unnecessary for users to modify the design source code although the code is supplied and they are free to do so if they wish. Algotronix can also customise the core as a service for users with particular requirements which are not met by the standard product.

- **cipher_function** - specifies whether an encryptor, decryptor or encryptor/decryptor is required. An encryptor will use less area than a decryptor which in turn will use less area than an encryptor/decryptor. For some FPGA architectures the area difference is quite small.
- **cipher_key_length** – this parameter is passed through to the AES unit within AES-XTS and specifies the AES key length the core should implement. The AES-XTS standard allows 128 bit or 256 bit AES keys. Somewhat confusingly the AES-XTS algorithm uses two AES keys and so the AES-XTS key length is described as 256 or 512 bits. This generic parameter is set according to the key length required in the AES encryptor within AES-XTS. Encryption with 256 bit AES keys requires significantly more computation than 128 bit AES keys and may need higher parallelism (and hence area) to achieve the desired throughput.
- **data_path_width** – In the present release of the code the data path width is fixed at 128 bits and this parameter must be set to the constant DP_WIDTH_128. Future versions of this product may support narrower data path widths for low performance applications.
- **sequence_number_width** – Length of the 'tweak' supplied to the AES-XTS algorithm in bits. This can be set to 128 to allow a random 128-bit tweak to be used but will normally be just wide enough to allow for the largest 'sector' number within the disk to be encoded.
- **data_unit_size_in_bytes** – The size of the unit to be encrypted, normally the size of a 'sector' on the disk. In the present code this must be a multiple of 16 bytes (128 bits).
- **number_of_aes_round_processors** – The number of parallel processing units to be provided. This value should be set in consultation with Algotronix because there are interactions between the AES key length, the data unit size and the number of round processors which can usefully be deployed.
- **maximum_ram_block_usage** - specifies the maximum number of FPGA RAM blocks to be used to implement AES SBoxes. This can be used to force the core to implement some or all SBoxes in logic rather than RAM blocks. It is especially useful when targeting FPGAs with limited amounts of block RAM or where other parts of the customer design have used most of the block RAM.
- **force_output_low_until_valid** – When true extra logic is added to hold output_text at zero except when output_valid is true. This prevents partially encrypted text being visible at the output of the core. Text which has not had the full number of encryption rounds applied may be useful to an attacker trying to deduce the key so this option is useful if the output of the core may become visible to an attacker. Usually there will be other circuits between the output of the core and signals which go off chip which prevent this, for example if the core output is registered within the user design when output_valid is true.
- **register_output_text –** When true an extra register is added on the output_text, advanced_output_valid and output_valid signals isolating combinational delays within the core and improving timing at the cost of one clock tick of latency and some area.
- **target_device** – Specifies the FPGA family to be targeted. This edition of the core targets FPGA families from Xilinx.

Core I/O Signals

The core I/O signals have not been fixed to specific FPGA device pins to provide flexibility for interfacing with user logic. Descriptions of all ports are provided in Table 2.

Signal	Signal Direction	Description
clock	input	Clock – active on rising edge
reset	input	Reset – active high. Usually connected to FPGA global reset. A global constant USE_ASYNCHRPNOUS_RESET with aes_package.vhd determines whether this is an asynchronous or synchronous reset. By default this constant is set to false because on Xilinx a synchronous reset normally provides better area and performance than an asynchronous reset.
enable	input	Module clock enable – 0: module is inactive, 1: module runs. This signal can be used for flow control to pause the core when the system is not ready to supply or to accept data.
do_encrypt	input	Specifies whether the core should operate in Encrypt or Decrypt mode. This input is only significant if the compilation option cipher_function is set to ENCRYPT_DECRYPT i.e. hardware for both encryption and decryption has been included.
start_data_unit	input	Starts a new block of encryption operations.
load_key	output	Load flag – high when the key is being loaded.
input_key [data_path_ width-1:0]	input	Bus to input the key for the AES encryptor. It will take 2 cycles to transfer two 128 bit keys for XTS-256 or four clock cycles to transfer two 256 bit keys for XTS-512.
load_text	output	Load flag – high when input_text is being loaded. The user may bring enable low during the clock cycle when load_text = 1 if the external system is not ready to provide more input text.
input_text [data_path_width-1:0]	input	Data input. Width is set by the data_path_width generic parameter
load_sequence_number	output	Load flag - high when the sequence number is being loaded.
input_sequence_number [sequence_number_width - 1 downto 0]	input	Data input to supply the sequence number. The sequence number must be unique for each data unit on the storage medium to be encrypted with the same key. This is essential to ensure that two data units with the same data will encrypt to different ciphertext.
output_valid	output	Valid flag – high when output_text is valid.
advanced_output_valid	output	High on the clock cycle immediately preceding output_valid.
output_text [data_path_width-1:0]	output	Data output: in the current version of the core this bus is 128 bits wide and the 128 bit block of text is transmitted in a single clock cycle.
waiting_for_next_data_unit	output	Indicates the core has finished processing the previous data unit and is idle waiting for the start_data_unit signal to go high..

Table 2: Core I/O Signals.

Description of Operation

AES-XTS operates on fixed size data units from a storage medium - these might correspond to sectors on a disk. Processing a data unit is initiated by bringing the start_data_unit signal high for one clock cycle. At this point the do_encrypt signal must be valid if the core is implementing both encrypt and decrypt functionality. If the core is implementing Encrypt only or Decrypt only then the do_encrypt signal is ignored and can be tied to constant 1 for encrypt or 0 for decrypt.

The core will then start processing by loading the key. AES-XTS uses two separate AES keys which are referred to in the standard documents as Key_1 and Key_2. Key_2 is used to encrypt the tweak and is loaded before Key_1 which used to encrypt data because the encrypted tweak must be available before data encryption can start. The tweak is then loaded through the input_sequence_number port and the tweak encryption begins. The term input_sequence_number is from the NIST XTSVS document, normally the tweak is a sequence number of the data-unit being encrypted within the storage medium (e.g. a sector number on a disk). The tweak encryption must conclude before the first data encryption can start which causes some latency at the beginning of a data-unit.

The AES-XTS core then loads Key_1 ready for the first data. If the operation is AES-XTS decrypt before starting to decrypt data it must calculate the keyschedule, for encrypt operations this can be done online with no additional latency. This difference between encrypt and decrypt timing is because AES-ECB decrypt uses round-keys in the opposite order from which they are generated by the keyschedule algorithm.

Once these preliminaries are accomplished the AES-XTS core sets the load_text signal high to indicate it is ready for data and loads a 128 bit data block through the input_text bus. If the AES-XTS core is pipelined multiple blocks of data will be loaded on successive clock cycles according to the pipeline depth. The AES-XTS core then processes the data, with a 128 bit data path this takes 1 clock cycle for every 'round' of AES. AES with a 128 bit key requires 10 rounds of processing and with a 256 bit key 14 rounds. When the processing is concluded the output_valid signal is brought high and output plaintext or ciphertext is presented on the output_text port. Simultaneously the next set of input data is loaded. This process is repeated until the data unit is completely processed. The present version of the AES-XTS core requires that the data unit length is a multiple of 16 bytes (128 bits) so it breaks evenly into AES blocks. This constraint is met by most data unit sizes on storage media. Algotronix will implement additional ciphertext stealing circuitry to remove this restriction on request.

In this description the sequence of activity has been described without stating the number of clock cycles between phases of activity. Timing charts for representative operations are available on request to Algotronix. It is recommended that the user circuit should synchronize to the AES-XTS core using the load_text, load_key and output_valid signals rather than assume a set number of clock cycles between various operations. Future updates to the core may have slightly different delays between processing phases as a consequence of enhancements to improve latency and throughput.

Verification Methods

The testbench includes a self-checking architecture of the top level entity in the VHDL design which uses the behavioral model of the AES-XTS algorithm to check the results from the synthesizable implementation code. This is implemented using the VHDL facility to provide multiple architecture definitions for a particular entity: the top level entity in the design has a 'self_test' as well as a 'synthesis' architecture defined. As shown in Figure 2, the self-checking architecture has an identical interface to the synthesizable architecture and instances the synthesizable architecture within itself but also contains behavioral code to capture all input and output signals and check their values against expected values computed using a behavioral model. When errors are detected assertions are triggered and the simulation is stopped with an error message.

The self-checking architecture of the AES-XTS core can be instantiated within the user's own simulations. This makes it easy to verify the core operates properly when connected to the user circuitry surrounding the core. In addition, the assertions within the self-checking code will detect and report many situations where the user design is not driving the core correctly simplifying the task of integrating the core with the larger user design.

The AES-XTS testbench supplied makes use of the self-checking configuration of the core. When configured for RANDOM_TEST (in aes_xts_parameters_package.vhd) the testbench stimulates the self-checking core with random sequence of data units (the number of data units is specified in aes_xts_parameters_package) and the self-checking core takes responsibility for detecting any errors.

As well as checking with randomly generated data the testbench can read vector files in the standard format specified in the NIST XTS Verification System document. This file format without the 'known answer' data to compare against will be used by NIST approved validation laboratories for algorithm validation testing. In the case of validation testing the output from the core would be sent to the laboratory to be checked. Algotronix supplies several sample test vector files with the core, these are taken from a ZIP file provided by NIST on their website, test examples from the standard documents and tests generated by a C program. The known-answer testing also serves to check the behavioral model of AES-XTS used in the random test mode.

For AES-XTS the random testing is usually more useful than the testing from pre-existing vector files because the Data Unit Length parameter could take many different values and pre-created tests may not have sufficient vectors for the exact configuration of interest.

Much of the code in AES-XTS is shared with other Algotronix AES products. The AES implementation building blocks are from the AES-G3 product and are also used in the AES-GCM cores. Thus many elements of AES-XTS are also tested in other contexts and with separate testbenches. The AES-G3 implementation is tested with NIST standard tests and versions of it have been validated by NIST test labs.

Verification of the AES-G3 core which forms the basis of the pipelined implementation of AES within AES-XTS is through a comprehensive VHDL testbench which supports the standard AESAVS test suite with additional vectors from the SP800-38A publication to test the various AES modes. The testbench allows simulation of the design source code and also post place and route timing simulation. The testbench can be used in Regression mode to confirm the functionality of the core against known 'golden' test vectors provided by Algotronix or in Qualification mode to generate response files from vectors supplied by a NIST approved certification laboratory.

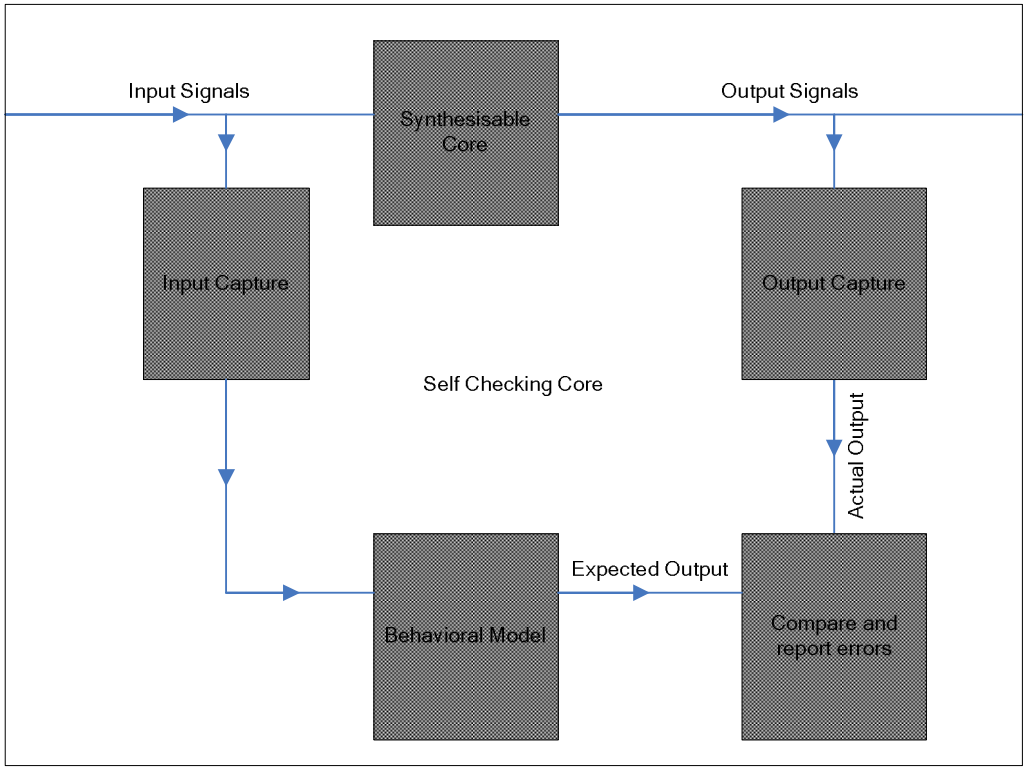

Figure 2, AES-XTS Self Checking Architecture

Recommended Design Experience

It is recommended that the user is familiar with the VHD language and with the Xilinx design flow and simulation tools. The coding standards used when creating the product allow automatic translation of the synthesizable core into Verilog without loss of readability and the core can be supplied in Verilog on request.

It is recommended that the user has a background in data security or takes appropriate advice when considering how to implement AES-XTS in a larger system.

Information on the XTS-AES Algorithm

AES Standards

The AES algorithm is standardized by the Computer Security Division, National Institute of Standards and Technology (NIST), Gaithersburg MD. The relevant standard to this implementation is FIPS 197 which specifies the AES algorithm. The FIPS 197 document provides an excellent and concise description of the processing involved in implementing AES and therefore this basic information on the structure of the AES algorithm is not repeated here.

NIST Special Publication SP800-38E describes the XTS-AES algorithm, this document is derived from IEEE standard 1619-2007. NIST also provides a compliance testing document: "The XTS Verification System (XTSVS)". This document specifies the test vector formats and tests which will be used by NIST approved test laboratories to verify implementations of XTS-AES.

These NIST documents can be downloaded free of charge from the NIST website ().

Customization Service

Algotronix can offer a cost effective customization service for this core in order to tune the implementation for easy integration into a larger system. It is also possible to produce variants with significantly higher performance at the expense of increased area and to create optimized variants of the core targeted at particular FPGA devices.

Algotronix Ltd.
130 - 10 Calton Road
Edinburgh EH8 8JQ
Phone: +44 131 556 9242
E-mail: cores@algotronix.com

URL: www.algotronix.com

Version Control Information	
Subversion Revision Number	58
Date	2014/11/09 00:02:53
Document	Aes Xts Data Sheet, Xilinx Edition
Status (blank field indicates OK/no warnings)	
	(Table auto-updates, do not edit field values by hand)