AES Core G2

Product Description

April, 2006

Algotronix Ltd.
P.O. Box 23116
Edinburgh EH8 8YB
Phone: +44 131 556 9242
Fax: +44 870 052 5069
E-mail: cores@algotronix.com

URL: www.algotronix.com

Algorithm	**AES**
Modes	ECB, CBC, CFB, OFB, CTR
Test Method	NIST AESAVS
Clock Cycles	44 (AES 128, Encrypt)
Perform-ance/Area	Tradeoff via compilation options. See section 2.
Deliverables	VHDL with Testbench. optimizations for Xilinx, Actel and Altera FPGAs.

AES Core

Features

- NIST validated implementation (Cert #347) of FIPS 197 (November 2001) AES Encryption and Decryption
- Supports all modes of AES defined in SP800-38A: ECB, CBC, CFB1, CFB8, CFB128, OFB and CTR.
- Supports 128, 192 and 256 bit keys
- Targets all modern FPGA families from Xilinx, Actel and Altera.
- Compile as Encryptor, Decryptor or Encryptor/Decryptor
- Supplied as easily customizable portable VHDL to allow customers to conduct their own code review in high-security applications. Comprehensive compilation options to include only required features and save area
- Supplied with comprehensive test bench implementing all AESAVS tests plus additional vectors in FIPS197 and Special Publication SP800-38A.

Potential Applications

- Government/Military Communications
- Wireless Networks
- Internet Security: IPSec, SSL, TSL protocols
- Financial Security ANSI X9.53
- Gaming Machines
- Content Anti-Piracy (e.g. set top box)
- Proprietary Security applications

General Description

This is a complete validated implementation of AES: all standard modes of operation and key lengths are supported. The core is developed in accordance with Federal Information Processing Standards Publication (FIPS PUB 197) "*Advanced Encryption Standard* (AES)" and tested in accordance with the NIST document "*The Advanced Encryption Standard Algorithm Validation Suite* (AESAVS)", November 15, 2002. The modes of operation are developed in accordance with the NIST document SP800-38A. The core was validated by a NIST approved laboratory in March 2006 and received certificate number 347 from NIST.

As well as supporting all standard modes of operation the core is designed for flexibility both at compile time and during operation. An extensive set of compile time options are provided so that the user can save area by including only those components of AES which are necessary to support their application. The core can be compiled for 128,

192 or 256 bit keys. For maximum flexibility the 192 bit implementation can select between 128 or 192 bit keys during operation. Similarly, the 256 bit implementation can select between 256, 192 or 128 bit keys.

To support highly sensitive applications the Algotronix AES Core is supplied as portable VHDL allowing customers to carry out an internal code review to convince ensure no 'Trojan horse' circuitry has been added to the core which could compromise its security. As an example in a 'chosen plaintext' attack a core could be designed to communicate key information in the ciphertext after receiving a 'trigger' set of plaintext. Such circuitry is almost impossible to detect by applying test vectors to the pins of an IP core - which is the only method customers have for checking cores supplied in 'compiled' formats such as device specific netlists. Providing source code also allows for easy customization of the interface to the core (e.g. data bus widths) or performance tuning by increasing parallelism at the expense of area.

The core has been simulated using the Active HDL simulation environment and implemented using Xilinx XST, Altera Quartus II and Synplicity synthesis for Actel FPGAs. It has also been simulated using Modelsim XE. Implementation directories from the FPGA vendor tools are supplied. The core implements all standardized modes and key lengths of AES and can be compiled as an encryptor, decryptor or encryptor/decryptor according to the application requirements. The FPGA implementations are optimized to take advantage of FPGA resources such as RAM blocks.

The core is supplied with a comprehensive test bench implementing all the NIST Known Answer Tests, Monte Carlo Tests, and the Multi-block Message tests.

Performance and Area

The core provides many options to allow the user to trade off area against throughput and latency so it is difficult to provide a comprehensive set of performance figures here. The FPGA architecture and speed grade also has a strong bearing on the results achieved. The purpose of this section is to give a reasonable overview of the achievable performance to allow comparison with competitive products.

The figures below are for a 128 bit encryption core supporting the ECB mode only. The KEYSCHEDULE_SHARES_SBOXES compilation option is used. The IMPLEMENT_SBOXES_IN_RAM compilation option is used so the SBox logic is implemented in FPGA RAM blocks rather than CLBs. These figures are for a 'press the button' flow through the vendor tools with no timing or other implementation constraints specified. Clock frequency could be increased by specifying constraints.

The first encryption in a block of chained CBC mode encryptions has an additional four cycle delay while the key and initial plaintext is loaded.

When comparing with competitive implementations it is important to check the number of clock cycles required per encryption as well as area and clock frequency. Unfortunately this information is not always provided on standard FPGA vendor datasheets. Implementations which have significantly lower area will usually be using a narrower datapath and therefore require more clock cycles per encryption e.g. a 16 bit datapath would require 88 clock cycles per encryption.

	Slices	RAM Blocks	ECB Clock Cycles	Clock Frequency
Xilinx (XC3S200-5, ISE 8.1i)	321	3	44	97MHz
	Logic Elements			
Altera (Cyclone EP1C12F256C6, Altera Quartus II, 5.1)	572	3	44	83.4MHz

Functional Description

As the designated replacement for DES the AES algorithm is the most commonly used symmetric cryptographic algorithm in new systems. It finds use in all the main Internet security protocols as well as financial, government and proprietary applications. The Algotronix core is aimed at providing sufficient performance with good area efficiency for mainstream applications. A further design goal is that it should be straightforward for a customer to relate the implementation source code back to the algorithm description in the standards.

In most practical applications of the AES core ease-of-use, area efficiency and compatibility with the feedback mechanism of Cipher Block Chaining mode are more important considerations than raw performance. However, the cores are designed for easy customization either by the customer or by Algotronix and it is straightforward to trade-off area for increased parallelism if more performance is required.

Information on the 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 NIST also provides a compliance testing standard: "*The Advanced Encryption Standard Algorithm Validation Suite* (AESAVS)", November 15, 2002. This document specifies compliance testing for AES. A third NIST document SP-800A "Recommendation for Block Cipher Modes or Operation" specifies the standard modes of operation for AES. This core implements these standards in their entirety, however compilation options are provided so that only circuitry required for a particular application need be included.

These NIST documents can be downloaded free of charge from the NIST website () and are also supplied with the core

The FIPS 197 standard document is easily downloaded and 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.

Textbook

Algotronix recommends the book "Applied Cryptography", Bruce Schneier, Second Edition, John Wiley and Sons ISBN 0-471-12845-7, as an introductory reference to cryptographic algorithms and their modes of operation.

Detailed Description

Core Interface

The interface to the core is specified as a VHDL package in the file aes_parameters_package.vhd. This package along with the package aes_package.vhd must be included in your design using the VHDL 'use' statement. Together these packages make available declarations of the various datatypes required and the component definition of the AES core itself.

The VHDL component declaration AES core (encryptor) is shown below.

```
-- definition of the aes unit
component aes_cipher_wrapper
  port(clock        : in  std_logic;
       clear        : in  std_logic;
       reset        : in  std_logic;
       enable       : in  std_logic;
       mode         : in  crypt_mode;
       KeyLength    : in  key_length;
       do_encrypt   : in  std_logic;
       start        : in  std_logic;
       load_text    : out std_logic;
       load_key     : out std_logic;
       input_text   : in  word; -- word is std_logic_vector(31 downto 0)
       initial_value : in  word;
       input_key    : in  word;
       output_valid  : out std_logic;
       advanced_output_valid : out std_logic;
       output_text   : out word);
end component;
```

The following table summarizes the functions of the various pins.

Pin	Direc-tion	Function
clock	input	Clock – active on rising edge
clear	input	Synchronous clear of controller state and most registers (Some shift registers do not use clear to allow a more area efficient implementation).
reset	input	Asynchronous reset – active high. Usually connected to FPGA global reset.
enable	input	Module clock enable – 0: module is inactive, 1: module runs
mode	input	Mode signal – specifies which mode of AES is to be implemented. See also the omit_* compilation options in the section below. If compilation options have specified that logic for a particular mode should be omitted then incorrect behaviour will result if that mode is selected.
KeyLength	input	Specifies the length of the key that is being used – 128, 192 or 256 bits. See also the max_crypt_size compilation option. Only keys up to the size specified in max_crypt_size may be specified e.g. if max_crypt_size generates hardware for a 192 bit key then KeyLength may be 128 or 192 bits but not 256 bits.
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 EncryptDecrypt i.e. hardware for both encryption and decryption has been included.
Start	input	Starts a new encryption operation or block of operations in the chained modes. The control signals Mode, KeyLength and Do_Encrypt are sampled and the parameters fixed for the next operation. The key is assumed to have changed and the keyschedule is recalculated (or loaded if the compilation option User_Calculates_Keyschedule is active).
load_text	output	Load flag – high when input_text is being loaded
Load_key	output	Load flag – high when the key is being loaded
Output_Valid	output	Valid flag – high when output_text is valid.
Advanced_Output_Valid	output	High on the four clock cycles immediately preceding output_valid. This signal gives advanced warning that the core is about to input and output data and can by external control circuitry to stop the core using the enable signal until the system is ready to provide new input data and accept output data.
Input_Text[31:0]	input	Data input: current 32-bit word of the 128-bit plain text
Output_Text[31:0]	output	Data output: current 32-bit word of the 128-bit cipher text
initial_value [31:0]	input	current 32-bit word of the 128-bit initial value for the chained modes of operation (ECB mode does not use the initial value).
Input_Key[31:0]	input	current 32-bit word of the key – it takes 4, 6 or 8 clock cycles to load a complete key. If the compilation option User_Calculates_Keyschedule is specified the entire keyschedule is input through this signal.

The input_text, output_text and initial_value busses transfer 128 bit AES block values over 32 bit wide busses (these 32 bit busses are declared as datatype word in the above code fragment) using four successive clock cycles. The input_key bus transfers 128, 192 or 256 bits of key information over a 32 bit bus using 4, 6 or 8 clock cycles respectively. Words are transferred in order starting with the most significant word.

Synchronising between the user design and the core is relatively straightforward. The user controls when the core processing starts using the 'start' signal. When the core samples 'start' as being high on a rising edge of the clock it prepares for a new chain of encryptions and on the following clock cycle starts to load key and plaintext data. When the key length is 192 or 256 bits, an initial two or four clock cycles respectively are required to load key data before the final four clock cycles in which both key and text data are loaded.

Once the initial data is loaded the core begins processing. The number of clock cycles required depends on the key length. Four clock cycles before the core is ready to output results the signal advanced_output_valid is brought high. The user design can use the enable signal to implement flow control if it is not ready to accept the output data – bringing enable low will stop the core processing.

Assuming that enable is left high, four clock cycles later the core will start outputting the processing plaintext or ciphertext (depending on whether this is an encryption or decryption operation). Simultaneously, if this is a chained operation it will load the next block of plaintext or ciphertext for processing.

The best way to get an understanding of the timing of the interface signals to the core is to simulate the core using the testbench provided and examine the waveforms on the signals in the table above during operation in the mode of the cipher you wish to use.

Parameterising the Core for Efficiency

The core supports a wide variety of compilation (generic) parameters which allow the user to configure it for area efficiency making sure that no unnecessary circuitry is included. The parameterisable version of the core is declared in the file aes_cipher.vhd. The compilation parameters are then specified in the file aes_parameters_package and applied in the file aes_cipher_wrapper – the result is a fully parameterized AES core with no generic parameters at the top level of the design hierarchy. The advantage of having a non-parameterised cell at the top of the design hierarchy is that the top level cell of the 'flat' post-synthesis design has exactly the same interface as the top level cell of the pre-synthesis design. This allows the same testbench to be used for both pre-synthesis and post synthesis simulation.

The most expensive sub-units in area terms within the AES design are the substitution or S-boxes. In an encryptor/decryptor there are three sets of S-boxes – S-boxes in the key-scheduler, the encryption data path and inverse S-boxes in the decryption data path. Many of the compilation parameters are concerned with efficiently implementing the S-boxes.

In AES the encryption and decryption operations are not identical. In the AES decryptor it is necessary to pre-calculate a set of round-keys and store them in a large memory (44, 32 bit words for AES 128) before decryption begins. In the encryptor it is possible to calculate the round keys on-line as they are required – eliminating the memory and the start-up delay associated with pre-calculation.

If the key-schedule is pre-calculated then one can use the same set of S-boxes in the Key Schedule unit as the encryption/decryption data path since they are not required at the same time. This is always a good option in a decryptor design and is usually a good tradeoff in an encryptor/decryptor or encryptor design.

In FPGAs using a block RAM to implement the Sboxes is usually the most efficient approach. However, if the block RAMs are required by other circuitry or if the FPGA does not have suitable block RAMs the S-boxes can be implemented as logic gates.

The following compilation options are specified by editing constant definitions in the aes_parameters_package file. This is the only file in the AES core release which will normally be edited by the user.

- **Cipher_function** - specifies whether an Encryptor, Decryptor or Encryptor/Decryptor is required.
- **Max_Crypt_Size** – specifies the maximum key length the core should implement. The user can select any key length up to and including this using control signals. For example, if Max_Crypt_Size is aes256 then the core would deal with 256, 192 and 128 bit keys.
- **Implement_SBoxes_in_RAM** – specifies that FPGA RAM blocks rather than logic gates should be used to implement SBoxes and Inverse SBoxes. This is the most efficient option if RAM blocks are available after mapping the remainder of the user design.
- **Omit_ECB_Mode, Omit_CBC_Mode, Omit_OFB_Mode, Omit_CTR_Mode, Omit CFB1_Mode, Omit_CFB8_Mode, Omit_CFB128_Mode** - Used to request that logic to support cipher modes that will not be required is omitted from the design. The CTR and CFB modes require quite large amounts of additional logic.
- **User_Calculates_Keyschedule** – specifies that the Keyschedule datapath should be omitted. Rather than providing a key the user will load a complete key schedule (i.e. all round keys) into the design. This can be a useful way to save area if the user's system has a microprocessor available to calculate the keyschedule and the key changes relatively infrequently so the time taken to calculate the keyschedule is not an issue.
- **Keyschedule_Shares_Sboxes** – specifies that the same SBoxes are used for the encryption datapath and the keyschedule unit. When this option is selected the encryption key schedule must be pre-calculated in the same way as decryption keys causing additional latency when the key is changed.
- **Target_Device** – Specifies one of Xilinx, Altera, ActelProASIC, ActelProASIC3, ActelAxcelerator FPGA families as the target device.

- **Force_output_low_until_valid** – When true the core will hold the output low at all times when valid output data is not present. When this signal is false the circuitry to hold the output zero will be omitted, saving some area. In this case the core output 'output_text' will show the values at intermediate rounds of the cipher as well as the final round. This data is not fully encrypted and, if available to an attacker, could compromise security of both the key and data. Therefore, this parameter should only be set to false if the user's design which contains the core can guarantee that an attacker will not be able to monitor the core output directly.

Test bench Information

The main component of the test bench is the test vectors specified in the NIST AESAVS and SP800-38A documents. These include Known Answer Tests (KAT) and Monte Carlo Tests (MCT).

The NIST Known Answer Tests (KAT) provide comprehensive coverage of all components of the AES core by applying a set of plaintext and keys to the core and checking that the correct ciphertext is generated (or in the case of the decoder supplying ciphertext and checking the plaintext). The various KAT tests are intended to stress different elements of the implementation. The KAT tests run quickly and are good diagnostics after any changes to the core Electronic Code Book function.

The Monte Carlo Tests (MCT) run the cipher iteratively and are very computation intensive. NIST specifies a method of iteratively running the AES algorithm starting from a supplied initial set of plaintext, initial value and key and automatically generating new test inputs based on the outputs of the previous test. Unlike the KAT tests NIST does not provide a complete set of Known Answers. When an AES implementation is sent for qualification by a NIST approved laboratory the starting condition of the MCT tests is not known in advance. NIST does provide some cut-down example vectors to check the basic operation of the MCT test implementation. The Monte Carlo Tests run a very large number of these tests – each test involves 100,000 encryption/decryption operations and there are several tests for each mode of the cipher. PC based simulators are unable to run complete MCT tests in a reasonable amount of time. The cut down versions still run tens of thousands of encryptions and provide a very high level of confidence in the core.

The Algotronix testbench runs in one of two basic modes:

- In REGRESSION mode the testbench automatically compares the results of test operations against known answers provided in the test file. These files have the extension .gld for 'golden' because they contain known good results. Most customers will only require this mode of operation.
- In QUALIFICATION mode the testbench processes vector files with extension .req for 'request'. These files do not contain correct answer data. In this mode the testbench generates output files with extension 'res' for 'result'. The result files would be required if the AES core was submitted to a NIST approved laboratory for qualification.

To use the testbench in qualification mode the user needs to purchase a set of test vectors from a NIST approved test laboratory. This package is then unzipped to form a directory tree in a convenient location. VHDL has very limited file handling abilities compared to a normal programming language and so some user intervention is required to set the qualification vectors up for the testbench. Firstly, the string constant 'qual_test_directory' in the file aes_testbench.vhd must be set to point at the vectors obtained from the test laboratory. Secondly, the user must provide a list of all the qualification tests in the file 'test_list.txt' within this directory. The testbench will read the test_list.txt file and run all the specified request files and produce the '.res' files for sending to the test laboratory. Algotronix recommends that the user contacts us for support if they wish to use the qualification features of the testbench.

Regression Mode

In regression mode the test vectors are stored in the subdirectory 'kat' of the design. The testbench reads the file 'test_list.txt' in the KAT directory to determine which tests to apply. The default list does not include the Monte Carlo tests to keep simulation runtime relatively short. The regression mode file names have a standard format (e.g. KATCBC-AESAVS-192.gld), which is explained below.

The first three letters are KAT or MCT and indicate whether this is a Known Answer Test or a Monte Carlo Test.

Following the KAT or MCT indication the next block of letters specify the mode of the cipher which is being tested: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback with 1, 8 and 128 bit variants (CFB1, CFB8 and CFB 128), Output Feedback (OFB) and Counter (CTR).

The next block of letters specifies the source of the test vectors:

- AESAVS – the NIST AESAVS document
- SP800 – the NIST SP800 document on modes of use of the cipher.
- SELECT – the original Rijndael submission to the NIST selection procedure (when candidate algorithms were submitted for selection as the new Advanced Encryption Standard).

The final three numbers in the file name specify the AES cipher key length that is to be tested – 128, 192 or 256 bits.

The testbench can determine from the filename the key length and mode of AES that is being tested, it then compares this against the compilation options for the core and only applies the test if appropriate (e.g. CTR tests would not be run against a core compiled for ECB only). Only the base case (e.g. KATCBC-SP800 is specified in the test_list.txt file – the testbench will run the tests at all key lengths supported by the core's compilation options (e.g. a core compiled for 256 bit keys will support 128, 192 and 256 bit keys but a 128 bit core only supports 128 bit keys). Normally the best choice is to specify all the tests except the MCT tests in the test_list.txt file and allow the testbench to select the appropriate ones based on the compilation parameters. The MCT tests make the run time very much longer but are highly unlikely to detect any problems not found by the simpler tests.

The KATECBCipher test uses the worked example from the FIPS197 standard and is only available with a 128 bit key, this will result warning messages when it is included in the test_list.txt file.

Post Layout Simulation

Some FPGAs require specific support from the testbench for initialization. For example the Actel FPGAs require a period after power on to initialize the RAM blocks used to implement SBoxes. In post implementation simulations the Xilinx FPGAs require some dead time after power on for the chip initialization to complete. For this reason you may see that there is a delay between the start of simulation time and the point at which vectors start to be applied to the core.

In post-implementation simulations it is quite likely that the delay on the implemented clock net will be longer than the delay between the chip pins and the first register for one or more data signals. At the system level the clock delay would usually be cancelled with a phase locked loop but this is not provided in the simulation. Instead a small delay is added by the testbench to all data signals to ensure that changes in data will not occur on chip before the clock edge that caused them.

The timing parameters aes_clock_high, aes_clock_low and aes_data_hold are set in the aes_parameters package to reasonable defaults. If your simulation encounters set up and hold problems post layout you may need to vary these parameters.

Limitations on NIST Supplied MCT tests

The Known Answer Data for Monte Carlo Testing supplied by NIST in the AESAVS document only includes results from the first three blocks of encryptions (MCT testing for qualification is specified to run 100 blocks of encryptions). Nevertheless, since each block requires 1,000 chained encryption operations and the entire block would fail if any of these gave an incorrect response this is still a strong proof of correctness. It also has an acceptable run time.

NIST does not specify any known answers for MCT testing in decrypt mode. In ECB mode the MCT test is symmetrical and one can make use of the encryption vectors in the opposite direction (i.e. start with the ciphertext and obtain the plaintext) but MCT in the chained modes of operation is not symmetrical.

Although it may look that the MCT tests have been cut back it is important to realize that the core of the cipher is the same in all operating modes. The testbench contains files for many different operating modes and key lengths each of which will test the basic ECB cipher logic. Since the AES algorithm is designed to amplify the effects of a single bit change (i.e. changing one bit in the input plaintext or key results in a completely unrelated ciphertext) in practice any problems in the encryption datapath are detected very quickly.

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 products.

Algotronix Ltd.
P.O. Box 23116
Edinburgh EH8 8YB
Phone: +44 131 556 9242
Fax: +44 131 556 9247
E-mail: cores@algotronix.com

URL: www.algotronix.com

Document	Revision	Date
AES-CORE-PD	1.0	February 2002
AES-CORE-PD	2.0	March 2003
AES-CORE-PD	2.1	March 2004
AES-CORE-PD	3.0	November 2004
AES-CORE-PD	3.1	March 2005
AES-CORE-G2-PD	1.0	January 2006
AES-CORE-G2-PD	1.1	April 2006