AES G2 Core, Xilinx Edition

February 10, 2015 Product Specification

Core Facts

Provided with Core	
Documentation	User Manual
Design File Formats	VHDL
Verification	Test Bench, Test Vectors
Instantiation templates	VHDL
Simulation Tool Used	
Model Tech ModelSim, Xilinx ISim	
Support	
Support provided by Algotronix	

Algotronix®
130-10 Calton Road
Edinburgh, Scotland
United Kingdom, EH8 8JQ
Phone: +44 131 556 9242
E-mail: cores@algotronix.com
URL: www.algotronix.com

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.
- 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.

Applications

- Wired, Optical and Wireless Networking
- Fire Control
- Gaming Machines
- Government/Military Communications
- Test Equipment

General Description

AES G2 is a mature 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. The core has been widely used in defense applications due to its ease of use, source code access and NIST certification.

The most recent version of the core was released in February 2015 and brings AES-G2 up to date with the design techniques used in Algotronix' more complex products. The core now uses the same Sbox implementation code as the AES-G3 product allowing it to fully take advantage of the most modern FPGA families. The testbench has been updated to use assertions and constrained random testing as well as running NIST validation vectors and the coding standards have been brought in line with Algotronix newer products.

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 VHDL source code allowing customers to carry out a code review to convince themselves no 'Trojan horse' circuitry has been added to the core which could compromise its security.

The core is supplied with a comprehensive test bench implementing a self testing version of the synthesisable core which compares its outputs with a behavioral model of AES, constrained random testing and the NIST Known Answer Tests, Monte Carlo Tests, and Multi-block Message tests.

Table 1 shows example implementation statistics for the core in the simple configuration most commonly used for comparison purposes. There are many implementation options available, some of which have a significant effect on area and performance. The AES-G2 core can be targetted at all Xilinx FPGA families, including older devices not listed in the table below. Algotronix will supply implementation results for the particular core configuration required in your application on request. The AES G2 product uses a fixed 32 bit data path width where Algotronix' more advanced AES-G3 product allows the data path width to be configured by the user. Algotronix can also supply pipelined AES implementations to further increase throughput. The AES-G3 product should be considered for applications which require higher performance than that available from G2.

Table 1: Example Implementation Statistics for AES-G2, 'Push Button' flow with Fmax specified by clock constraint. ECB mode, Encrypt Only, 128 Bit Key

Family	Example Device	Fmax (MHz)	Slices[1]	IOB[2]	GCLK[2]	BRAM	MULT/ DSP48	DCM	Throughput (MBit/sec)	Design Tools
Spartan-6	XC6SLX4-3	150	152	105	1	3	0	0	436	ISE 14.7
Zynq	XC7Z020-3	192	217	105	1	3	0	0	558	ISE 14.7
Kintex Ultrascale	XCKU035-1	250	291 (LUT)	105	1	1.5	0	0	727	Vivado 2014.1
Spartan-3	XC3S50A-5	108	413	105	1	3	0	0	314	ISE 14.7

Notes:

1) Actual slice count dependent on percentage of unrelated logic – see Mapping Report File for details. LUT count reported for Kintex Ultrascale.

2) IOB count when all core I/Os and clocks are routed off-chip, which is **not** the intended usage. The core interface is designed to provide flexibility inside a larger FPGA design. GClk signal is normally shared with user's design.

Functional Description

The main functional blocks as shown in Figure 1 .

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 AES-G2 core is aimed at providing ease of use 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.

ECB Data Path

This block implements the ECB mode of the AES Algorithm. All other modes of AES are built on top on this basic encryption operation.

Key Schedule

This block calculates the round keys for each stage of the AES algorithm based on the key supplied by the user. A compilation option allows the user to omit this unit to save area in which case the core must be supplied with the complete keyschedule. This option may make sense if the key changes relatively infrequently and there is a microprocessor available elsewhere in the system to calculate the keyschedule.

AES Mode Logic

This block contains the feedback paths and additional logic required to implement the more complex modes of AES – CBC, OFB, CFB1, CFB8, CFB128 and CTR. Compilation parameters are supplied so only the logic for those modes which are required by the user will be instantiated.

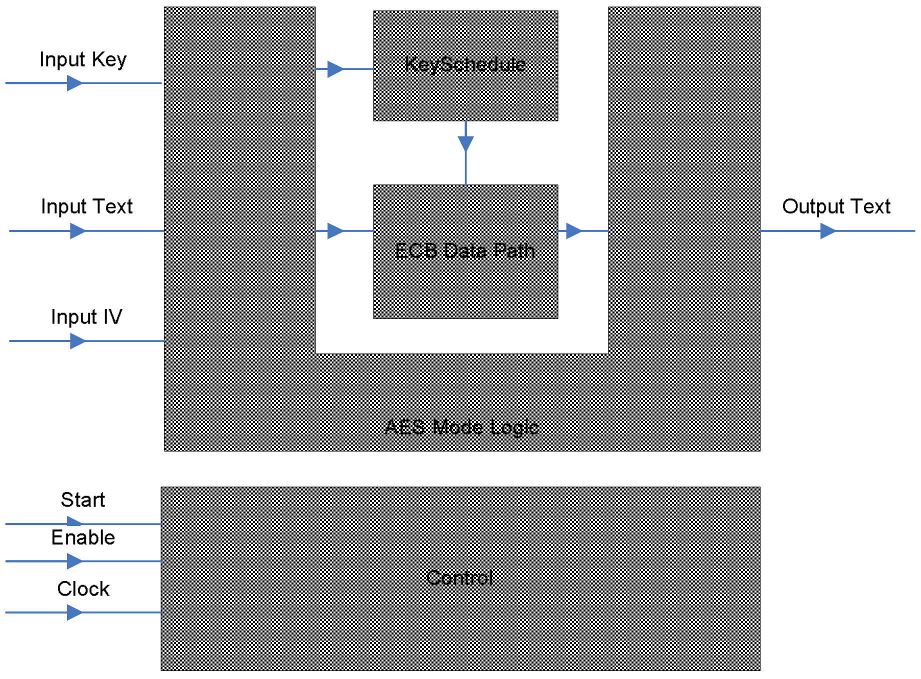

Figure 1, AES G2 Core Block Diagram

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.

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-parameterized 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. Algotronix can also supply a wrapper to allow the core to be used within a Verilog design.

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.

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.
- **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.

- **target_device** – In the Xilinx Edition version of the core only Xilinx devices are supported so target device should be set to XILINX for older devices or XILINX_VIRTEX_5 for newer devices with 6 input LUTs.

Core I/O Signals

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

Signal	Signal Direction	Description
clock	input	Clock – active on rising edge
reset	input	Reset – active high. A global constant in aes_package.vhd USE_ASYNCHRONOUS_RESET determines whether this is a synchronous or asynchronous reset. On Xilinx FPGAs a synchronous reset is usually more efficient.
enable	input	Module clock enable – 0: module is inactive, 1: module operates
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.
key_length	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 key_length 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, key_length 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 port.

Table 2: Core I/O Signals.

Description of Operation

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 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. Algotronix also provides a 'Getting Started' application note which includes timing diagrams for the core and demonstrates the core in operation on a Xilinx evaluation board.

Verification Methods

The testbench includes a self-checking configuration of the top level entity in the VHDL design which uses a behavioral model of the AES-G2 algorithm to check the results from the synthesisable 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_checking and a synthesis architecture defined. As shown in Figure 2, the self checking architecture has an identical interface to the synthesisable architecture and instances the synthesisable 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.

This self-checking configuration of the AES-G2 core can also 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 easier.

The AES-G2 testbench supplied with the core also makes use of the self checking configuration of the core for random testing. The testbench stimulates the self checking core with a random sequence of packets, writes of key information and key activations and the self checking core takes responsibility for detecting any errors.

The AES behavioral model used in the self checking core it is also operational in regression and qualification testing so that it is validated along with the synthesisable logic by the NIST test vectors.

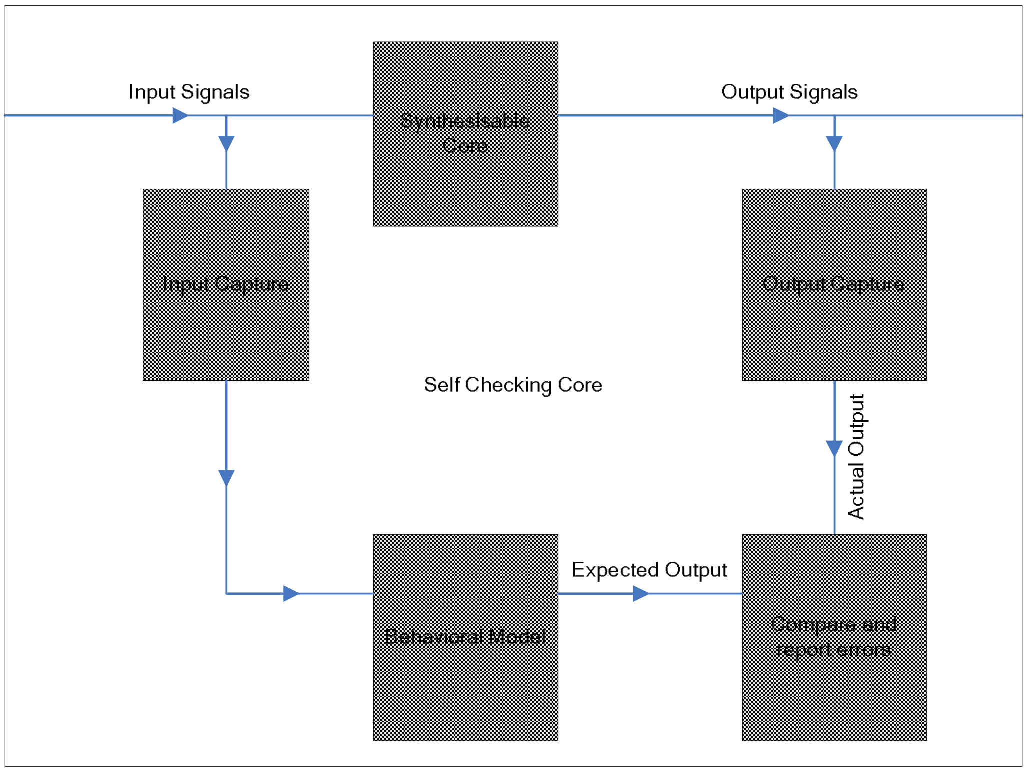

Figure 2, AES-G2 Self Checking Architecture

The AES-G2 testbench runs in one of three basic modes:

- In RANDOM mode the testbench generates random test vectors and applies them to the core. Results from the core are checked against a behavioral model. This is a less structured and more comprehensive test than the NIST test vectors as it will randomly swap many parameters such as cipher mode, encrypt or decrypt and key length between vectors where the NIST tests group many vectors for the same cipher set up.
- 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.
- 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.

As well as random testing using the self-checking version of the core in RANDOM mode the test bench has access to the test vectors specified in the NIST AESAVS and SP800-38A documents in REGRESSION and QUALIFICATION modes. 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.

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.

Recommended Design Experience

It is recommended that the user is familiar with the VHDL language and with the Xilinx design flow and simulation tools. The core can also be instantiated inside a wrapper to allow use with a Verilog design flow.

It is also recommended that the user has a background in data security or takes appropriate advice when considering how to implement AES-G2 in a larger system.

Ordering Information

This product is available directly from Algotronix under the terms of the SignOnce IP License. Please contact Algotronix for pricing and additional information about this product using the contact information on the front page of this datasheet. To learn more about the SignOnce IP License program, contact Algotronix or visit the web:

Email:	commonlicense@xilinx.com
URL:	www.xilinx.com/ipcenter/signonce

Export Control

Strong encryption technology such as AES is the subject of international export regulations. Algotronix is located in the United Kingdom and export of this core is regulated by the UK government.

The core is freely available within the European Union and in addition can be supplied immediately to the following countries: United States, Australia, New Zealand, Canada, Norway, Switzerland, Japan.

Export to other countries requires an export licence. The UK Department of Business, Enterprise and Regulatory Reform publishes information on their website (www.berr.gov.uk) which gives an indication of average export licence processing times for various countries and the percentage of licence requests which are granted. For many countries obtaining an export licence can be done relatively quickly and with only a small amount of paperwork.

It is the the responsibility of the customer to comply with all applicable requirements with respect to re-export of products containing the AES technology.

Related Information

Industry Information

The AES standard documents FIPS197, SP800-38A and AESAVS are available from the National Institute of Standards and Technology, Computer Security Resouce Center website (www.csrc.nist.gov).

Xilinx Programmable Logic

For information on Xilinx programmable logic or development system software, contact your local Xilinx sales office, or:

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124

Phone:	+1 408-559-7778
Fax:	+1 408-559-7114
URL:	www.xilinx.com

URL: www.algotronix.com

Version Control Information	
Subversion Revision Number	**66**
Date	2015/03/12 16:25:03
Document	Aes G2 Data Sheet, Xilinx Edition
Status (blank field indicates OK/no warnings)	
	(Table auto-updates, do not edit field values by hand)