AES GCM Core, Xilinx Edition

August 1, 2008 Product Specification

Algotronix®

PO Box 23116
Edinburgh, Scotland
United Kingdom, EH8 8YB
Phone: +44 131 556 9242
Fax: +44 870 052 5069
E-mail: cores@algotronix.com
URL: www.algotronix.com

Core Facts

Provided with Core	
Documentation	User Manual
Design File Formats	VHDL or EDIF Netlist
Verification	Test Bench, Test Vectors
Instantiation templates	VHDL
Simulation Tool Used	
Model Tech ModelSim XE, Aldec Active HDL	
Support	
Support provided by Algotronix	

Features

- Full Implementation of AES Galois Counter Mode (GCM) Proposal as specified in IEEE 802.1 and NIST draft SP800-38D
- Based on Algotronix' G3 AES core, can implement simple AES modes as well as GCM
- Wide range of configuration options allowing efficient implementation of applications with throughputs from 100Mb/sec up to around 2Gbit/sec
- Supplied as VHDL source code to allow security review

Applications

- Wired, Optical and Wireless Networking
- IEEE 802.1 MAC standard
- Government/Military
- Satellite

General Description

The Algotronix AES-GCM Core implements the proposed Galois Counter Mode of operation of the AES algorithm. This mode of operation is described in a proposal by Cisco to the National Institute of Standards and Technology (NIST) and has been adopted by the IEEE for the 802.1 MAC standard. NIST has published Special Publication SP800-38D describing GCM mode making it an officially endorsed mode of operation of the AES cipher.

AES-GCM is significantly more complex than the simple standard modes of AES such as Cipher Block Chaining (CBC) specified in NIST special publication SP800-38A. Unlike simple modes of the cipher which provide only confidentiality, GCM provides both confidentiality and authentication. Authentication is the ability to detect tampering with the encrypted message as it passes between the sender and receiver and in most applications is essential for security. GCM mode also provides a standard method for processing data streams whose length is not a multiple of the 128 bit AES block size. GCM mode is based on the counter (CTR) mode of AES which can be parallelised to achieve very high throughput. In contrast, the Counter with CBC MAC (CCM) mode of AES specified in NIST publication SP800-38C and various IEEE wireless networking standards, which also provides authentication, uses the CBC mode of AES which has a feedback loop which prevents parallelisation. Thus GCM is the most suitable of the standard modes of AES to provide both authentication and confidentiality for very high speed

networks. A further advantage of AES-GCM is that it can operate 'on-line' processing information as it arrives, AES-CCM by contrast needs to know the length of the packet to be processed before processing begins which may require storing the packet before starting to encrypt it.

The Algotronix GCM core is an 'add-on' to the G3 AES core and uses the G3 core to implement the AES part of the AES-GCM algorithm with all standard key sizes (128, 192 or 256 bits). The GCM code extends G3 by providing a lookup table based 128 bit Galois Field multiplier to implement the Galois hashing function and the additional control circuitry to implement the AES-GCM protocol. The GCM core implements the complete set of functionality from the GCM proposal including Additional Authenticated Data (AAD). AAD is passed through the hash function in AES-GCM so that any unauthorised changes to the authenticated data can be detected but is not encrypted. The Algotronix GCM core is designed to minimise the latency associated with changing keys at the beginning of new GCM packets making its throughput less sensitive to GCM packet size.

The core is supplied with a testbench which implements the complete set of vectors from the GCM proposal. The testbench also includes a set of several thousand random test vectors generated from a software implementation of GCM. These additional vectors have randomly chosen IV, AAD and text lengths and cover several important 'corner' cases not included in the vectors in the GCM proposal. The comprehensive testbench for the G3 core can be used to verify the AES functionality. The GCM testbench operates in regression mode to verify any changes you may make to the source code.

The Algotronix AES-GCM 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 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.

The AES-GCM core also optionally provides access to the internal AES-G3 core to implement standard AES modes (ECB, CBC, OFB, CTR, CFB).

This level of flexibility makes it easy to experiment with area/performance/functionality tradeoffs and makes it highly likely that the core will be useful in multiple projects.

The AES 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.

Table 1: Example Implementation Statistics for GCM, 'Push Button' flow with Fmax specified by clock constraint

Family	Example Device	Fmax (MHz)	Slices[1]	IOB[2]	GCLK[2]	BRAM	MULT/ DSP48	DCM	Throughput (MBit/sec)	Design Tools
Low Area – 16 bit data path, Encrypt only, 128 bit key										
Spartan-3E™	XC3S250E-5	100	1696	78	1	2	0	0	100	ISE 10.1
Medium Area – 32 bit data path, Encrypt only, 128 bit key										
Spartan-3E™	XC3S250E-5	90	1695	126	1	4	0	0	180	ISE 10.1
Medium Area, additional capabilities - 64 bit data path, Encrypt and Decrypt, 256 bit key										
Spartan-3E™	XC3S12000E-5	74	2854	222	1	6	0	0	338	ISE 10.1
High Performance – 128 bit data path, Encrypt only, 128 bit key										
Virtex-4™	XC4VLX40-12	133	3362	412	1	10	0	0	1700	ISE 10.1
Virtex-5™	XC5VLX50-3	148	1602	412	1	5[3]	0	0	1920	ISE 10.1

Notes:

1) Actual slice count dependent on percentage of unrelated logic – see Mapping Report File for details

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.

3) Virtex 5 RAM blocks contain two 18K RAMs and are approximately twice the capacity of those in the other FPGA families.

Functional Description

The main functional blocks as shown in Figure 1 .

AES-G3

This block implements the ECB mode of the AES Algorithm for use by GCM. Although GCM is described as using AES in counter (CTR) mode it requires specific capabilities from the counter implementation which are not available in standard AES counter mode therefore it is built on the G3 implementation of ECB with its own customised implementation of the counter rather than using the G3 implementation of CTR mode. The GCM core can be configured to provide access to the internal AES-G3 core to provide standard AES modes as well as GCM.

GF Multiply

The GCM algorithm requires a 128 bit x 128 bit Galois Field (GF) multiply operation for each AES encryption. The Algotronix core provides a highly configurable GF multiplier with parameterisable data path width and digit size. Data path width is selected to match that chosen for the AES unit and the digit size is chosen to match the Galois Field multiply throughput with the throughput of the AES core. Wide digit sizes require more area but compute the multiply in fewer clock cycles. The Algotronix implementation of the GF multiplier was chosen to minimise latency following a change of cryptographic key so that the core can efficiently support minimum sized packets with a potential key change on each packet.

GCM Mode Logic

This block contains the datapaths required to route data between the various computational units within the core under the control of the GCM-Control block.

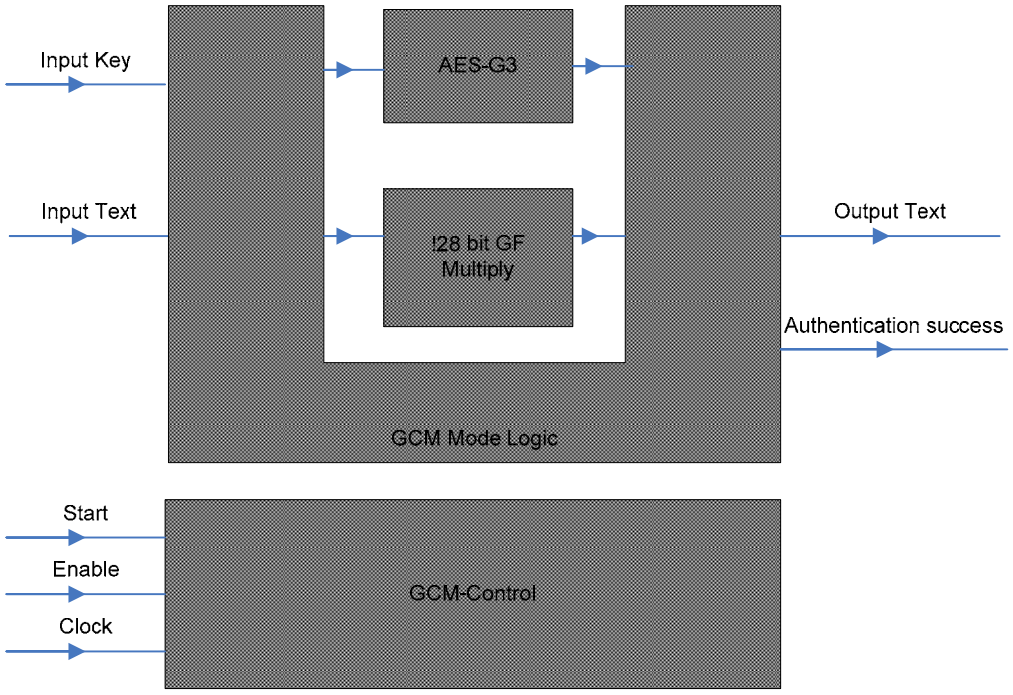

Figure 1, AES-GCM Core Block Diagram

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.

- **Cipher_function** - specifies whether an Encryptor, Decryptor or Encryptor/Decryptor is required. In the case of GCM there is little area advantage to specifying Encryptor or Decryptor rather than the more general Encryptor/Decryptor.
- **Max_Crypt_Size** – specifies the maximum key length the core should implement. The user can select any key length up to and including this maximum during operation using the key_length signal. For example, if **Max_Crypt_Size** is aes256 then the core would deal with 256, 192 and 128 bit keys. Longer key sizes require additional processing rounds in the AES algorithm and may increase overall processing time (if the increased AES processing time becomes larger than the processing time for the GCM part of the algorithm). There is also a small area advantage to specifying a shorter maximum key lengths.
- **Data_Path_Width** – Width of the data path within the AES-GCM core and the external buses supplying input and output data to the core. Data path widths of 128, 64, 32 and 16 bits are supported. Wider data path widths require significantly more area but offer a near linear reduction in the number of clock cycles required to process data i.e. with a 128 bit data path the core will require about half the number of clock cycles to process a particular GCM block it would with 64 bit datapath. Scaling is not completely linear because there are a small number of overhead cycles which are independent of data path width. Maximum clock frequency is also generally lower in designs with wide data paths because wire lengths are increased.

- **CTR_Mode_Counter_Width** – Length of the counter used for the AES-CTR component of the GCM algorithm in bits. This is usually 32 bits which matches with the 96 bit IV which is normally used in AES-GCM. This parameter is also specifies the counter width to be used when CTR mode is specified as an additional simple AES mode (see the description of the **Omit_Simple_AES_Modes** generic parameter below).
- **Assume_Fixed_96_Bit_IV** – This parameter specifies that the user supplied IV will always be exactly 96 bits long. A fixed 96 bit IV is specified in some standards which make use of AES-GCM even though the GCM standard itself allows more flexibility. When it is known that the IV will always be 96 bits the core can eliminate some circuitry which could save some area and increase performance in some cases. This generic parameter is ignored by the present core design but will be used in future releases.
- **Digit_Width** – This parameter specifies the size of the digit used in the GF multiplier. With a 128 bit data path width it takes 128/**digit_width** operations to calculate the GF multiplier product - so wider digits reduce the number of clock cycles per multiply. The number of clock cycles also depends on the **data_path_width** – with a 32 bit data path width it takes 4 word operations per 128 bit block. So an 8 bit digit size with a 32 bit data path results in (128/8)*4 = 64 clock cycles per multiply. In contrast a 64 bit digit size with a 128 bit data path width would compute the multiply in 2 clock cycles. Longer digit sizes require substantially more area than short digit sizes and can have longer combinational delays resulting in reduced clock frequency. The general goal is to choose the digit width so that throughput through the GF multiplier balances with throughput through the AES core. There is no advantage to having the GF multiplier run faster than the AES core can provide data or to have the AES core provide data faster than the GF multiplier can process it. **Digit_Width** cannot exceed **data_path_width**. Possible values for digit width are 8, 16, 32, 64 and 128 bits.
- **Force_output_low_until_valid** – When true the AES core will hold the output_text bus 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 design which contains the core can guarantee that an attacker will not be able to monitor the core output directly. The circuits which provide this function are on the critical timing path and require some additional area so in designs which stress area or performance and in which it is certain there is no security problem with partially encrypted values appearing on the core output this parameter should be set to false.
- **Optimise_for_area** – When true the AES core calculates the keyschedule using the AES G3 core's KC_OFFLINE option. This shares block RAM between the keyschedule unit and the main encryption datapath to save area. As a consequence the keyschedule calculation must be complete before data processing begins and there is increased per-packet latency, throughput once data processing begins is not affected. In the future additional area/performance tradeoffs may be triggered by setting this parameter to true.
- **Omit_Simple_AES_Mode_Operation** – When false the GCM core provides access to the G3 AES core contained within it to implement standard AES modes. The Omit_Mode array is used to determine which standard modes are provided. The Initial Value (IV) for the standard AES modes is loaded over the input_text bus before the key rather than over a separate IV port as on the G3 core.
- **Omit_Mode** – This array is only significant when **Omit_Simple_AES_Mode_Operation** is false. The array specifies which of the AES simple modes should be implemented. Since the G3 core is used with an ONLINE keyschedule (see the G3 core data sheet for more information on ONLINE and OFFLINE keyschedule calculations) inside GCM, the standard ECB and CBC modes cannot implement decryption operations although ECB and CBC encrypt is available. All other modes offer full functionality. This restriction can be removed on request but will result in additional latency in GCM mode for keyschedule calculation.

- **Generate_Advanced_Output_Valid** - The advanced_output_valid signal gives a one AES block (i.e. 1 clock cycle with a 128 bit data path width, or 4 clock cycle with a 32 bit data path) warning that the GCM core will wish to output data. The user design can then take the core's enable signal low if it is unable to accept this data. The GCM protocol makes it difficult to determine in advance when the core will output data so this functionality is provided by delaying the generated output data inside the core and thus increases latency and requires additional area. Therefore it is recommended that **Generate_Advanced_Output_Valid** is set to false unless the advanced_output_valid signal is absolutely required.
- **Target_Device** – In the Xilinx edition of the GCM core only Xilinx FPGAs are supported and target device must be set to 'xilinx'.

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
clear	input	Synchronous clear of controller state and most registers (Some shift registers do not use clear to allow a more area efficient implementation). Normally not used and tied to constant '0' so that synthesis will optimize the associated circuitry out to save area. See comment on 'reset' signal.
reset	input	Asynchronous reset – active high. Usually connected to FPGA global reset. Synthesis options can be used to convert this to a synchronous reset – on Xilinx this normally provides better area and performance than an asynchronous clear. If reset is made synchronous the separate synchronous clear signal on the AES-GCM core is redundant and can be tied to '0'.
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.
key_length	input	Specifies the length of the key that is being used – 128, 192 or 256 bits. See also the key_datapath_size compilation option. Only keys up to the size specified in key_datapath_size may be selected e.g. if key_datapath_size generates hardware for a 192 bit key then KeyLength may be 128 or 192 bits but not 256 bits.
use_GCM_mode	input	Specifies whether the core should implement GCM or the simple AES mode specified by aes_core_simple_mode. '1' selects GCM. This signal is only significant when the generic parameter Omit_AES_Simple_Mode_Operation = false.
aes_core_simple_mode	input	Specifies the simple mode of AES to be implemented. This signal selects between the available mode options compiled into the hardware. The compilation parameter array omit_mode specifies which mode logic is compiled into the core and is available for selection.
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 block of encryption operations. The key and the control signals KeyLength and Do_Encrypt are sampled and these parameters are fixed for the next block of operations. The key is assumed to have changed and the hash key H required by the GCM algorithm is recalculated.
input_text_kind[1:0]	input	Specifies whether the current 128 bit block of input text is an Initial value (TEXT_IV), additional authenticated data (TEXT_AAD), plaintext/ciphertext for encryption or decryption (TEXT_TEXT) a marker indicating the end of the plaintext/ciphertext data stream (TEXT_FINISHED)) or an authentication tag to be checked (TEXT_TAG). TEXT_TAG only occurs on the input in decryption.
input_text_width[4:0]	input	Specifies the number of bytes in the current 128 bit block which are actually used. In GCM more significant bits are filled first – so on a partial block it is the least significant bits which will be empty. Unlike the simpler modes of AES, GCM can deal with data streams which are not a multiple of 128 bits. All 128 bits will be used on every block except the last block in a stream of IV, AAD or TEXT blocks to be processed.
input_text [data_path_width-1:0]	input	Data input. Width is set by the data_path_width generic parameter. With narrower bus widths multiple clock cycles are required to transfer a 128 bit block of text. The most significant word is transferred first.
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.
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 AES round immediately preceding output_valid. The number of

		clock cycles per round depends on the internal data bus width – with a 128 bit databus this signal will go high for a single clock cycle. This signal gives advanced warning that the core is about to input and output data and can be used for flow-control. External circuitry can stop the core using the enable signal until the system is ready to provide new input data and accept output data. Advanced_output_valid is only generated when the generic parameter generate_advanced_output_valid is true – see the comments on this generic parameter.
output_text_kind	output	Specifies whether the current 128 bit block of output text is an Initial value (TEXT_IV), additional authenticated data (TEXT_AAD), plaintext/ciphertext for encryption or decryption (TEXT_TEXT a marker indicating the end of the plaintext/ciphertext data stream (TEXT_FINISHED)) or a calculated authentication tag (TEXT_TAG). The calculated tag will appear on the output in both encrypt and decrypt operations.
output_text_width[4:0]	input	Specifies the number of bytes in the current 128 bit block which are actually used. Unlike the simpler modes of AES, GCM can deal with data streams which are not a multiple of 128 bits. All 128 bits will be used on every block except the last block in a stream of IV, AAD or TEXT blocks.
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.
input_key keyschedule_data_path_ width-1:0]	input	Bus to input the key for the AES encryptor. It may take multiple clock cycles to load the 256, 192 or 128 bit key over the bus. The most significant data word is transferred first. In the case of a 128 bit input_key bus and 192 bit key only the most significant 64 bits on the input_key bus are used on the second clock cycle. This bus is the same width as the input_text bus and may be connected directly to input_text to save pins since the two busses never transfer data in the same clock cycle. This approach is particularly valuable when the buses are 128 bits wide.
authentication_success	output	On decrypt indicates whether the hash function has successfully authenticated the data stream. This signal is valid on the clock cycle following the cycle with output_valid = '1' and output_text_kind = GCM_TAG.

Table 2: Core I/O Signals.

Description of Operation

Processing a packet of GCM data is initiated by pulsing the start signal high for one clock cycle. The control signals which set the operating mode of the core for this packet must be valid at this time: do_encrypt, key_length, use_gcm_mode, aes_core_simple_mode. Some of these signals are latched internally during start and changes during period while the core is processing data will have no effect.

In this description the sequence of activity is described but the number of clock cycles between phases of activity is generally not documented. The user circuit should synchronise to the GCM core using the load_text, load_key and output_valid signals rather than by assuming a set number of clock cycles between various operations. Future updates to the core may have slightly different delays between phases as a consequence of changes to improve latency and/or throughput.

Simple AES Mode

If the core has been instructed to implement a simple AES operation (use_gcm_mode = 0, and the compilation generic parameter provide_simple_mode_operation is true) then the core will carry out the following sequence of operations:

1. load_text will be brought high and the 128 bit Initial Value (IV) for the simple AES operation will be loaded over the input_text bus. This may take multiple clock cycles depending on the data_path_width parameter which sets the input bus width.

2. load_key will be brought high and the 128, 192 or 256 bit key value will be loaded over the input_key bus. Again, this may require multiple clock cycles.

3. load_text will be brought high and the first 128 bit text (plaintext or ciphertext depending on whether an encryption or decryption operation was specified) will be loaded over input_text.

4. The AES core will carry out the encryption operation. This will take a variable number of clock cycles depending on the key length and data_path_width.

5. The AES core will bring load_text and output_valid high and will simultaneously output the newly processed 128 bit block of plaintext/ciphertext using the output_text bus and input the next text block for processing using the input_text bus.

6. The core will process the new data and loop back to step 5 to output the results and input the next block.

In simple AES mode the GCM core is providing direct access to the G3 AES core within it and the timing, apart from the loading of IV over the input_text bus before the operation starts is the same as for G3 AES. GCM signals such as input_text_kind are not significant in this mode and the special value GCM_TEXT_FINISHED is not used to indicate the end of the data stream. Instead, the core will continue to process a stream of data with the same key until the start signal is brought high again. As with AES G3, flow control is achieved by setting enable = 0 when the core indicates using the load_text and output_valid signals that it wishes to carry out a data transfer and the external circuitry is not ready.

GCM Operation

This section describes operation of the core when implementing the AES-GCM mode. In this configuration there is a single encryptor and data_path_width may be 16, 32, 64 or 128 bits. Code to provide multiple AES encryptors for even higher throughput can be supplied on request but is not included in the standard product. A more detailed description with timing diagrams is provided in the "Getting Started with the AES-GCM core" application note.

As with simple mode, operation is triggered by the start signal being brought high for one clock cycle. The core then carries out the following sequence of operations.

1. load_key is brought high and the AES key is loaded. This may take multiple clock cycles depending on data_path_width and key length.
2. The AES unit is used to encrypt the constant value 0 with the key to calculate the hash key H and the Galois Field (GF) multiplier is initialised. This stage must complete before any further activity is started because subsequent stages involve using the GF multiplier to compute hash values and require H. The number of clock cycles to perform this AES encryption depends on the data_path_width and the key length and can be significant.
3. The load_text signal is brought high and the first word of the IV is loaded. There must always be IV data in a valid GCM packet. The IV is supplied over the input_text bus, the input_text_width signal specifies how many bytes of the IV are valid and the input_text_kind signal must indicate TEXT_IV. If the IV is exactly 96 bits long (as determined by the input_text_width signal) then it is immediately loaded into the IV register. If it is less than 96 bits long it is processed using the GF multiplier to create a 128 bit hash value. This processing can take many clock cycles depending on how the GF multiplier is configured (data_path_width and digit_size).

4. If the initial block of IV was 128 bits long then it is processed using the GF multiply unit and the core goes on to load possible subsequent words of IV data. The core continues to load blocks of IV (setting load_text = 1) until it receives a text block whose type is not TEXT_IV. Each block is processed with the GF multiplier as it is loaded and added into the running hash computation.
5. If the IV was not 96 bits long and is being calculated using the GHASH algorithm the final component processed by the GF multiplier and included in the GHASH value is the length of the IV in bytes. Thus where there are n, 128 bit blocks of IV (with the last block possibly not complete) it takes n+1 GF multiply operations to compute the GHASH algorithm and determine the final IV. This makes the processing latency for non-96 bit IVs substantially higher than for 96 bit IVs.
6. At this point the core has loaded and processed the IV and also has loaded the next block of input text. If the packet includes an AAD the next block will have type TEXT_AAD. If there is no AAD but there is text to be processed it will have type TEXT_TEXT, otherwise if the packet is empty it will have type TEXT_FINISHED. Packets with no AAD, no text, or neither text nor AAD are legal in GCM and are handled correctly by the core.
7. The core now moves on to process the AAD (if an AAD is present in the GCM packet). Blocks of AAD text are loaded and processed by the GF multiplier until a block whose type is GCM_TEXT or GCM_FINISHED is loaded. Each block of AAD is also passed through to the GCM core output with a delay of one block (i.e. one clock cycle with a 128 bit data bus, 4 clock cycles with a 32 bit data bus). The final block of AAD may be incomplete as indicated by the input_text_width signal. If the non-AAD block which terminates the AAD is of type TEXT_TEXT the core moves on to process plaintext/ciphertext, otherwise it moves directly to calculate the final GHASH and TAG values for the GCM packet (step 9 below).
8. The core now processes the plaintext/ciphertext component of the packet. On encrypt the GHASH calculation involves encrypting the input text and applying the GF multiplier to the resulting ciphertext. On decrypt the GF multiplier is applied to the input ciphertext. The calculated ciphertext or plaintext is passed to the core output_text bus and the running GHASH value is updated after each block of text. The final block of text may be incomplete as indicated by the input_text_width signal. The end of the text is marked by loading a block with type GCM_FINISHED. This marker is used for both encrypt and decrypt. The core then moves on to complete the GHASH calculation and the GCM authentication tag.
9. The final part of the GHASH calculation is to include the length of the text and AAD fields in the hash value. This requires an additional GF multiply operation. The calculated GHASH value is then encrypted by the AES core and the resulting value is the authentication tag. The calculated authentication tag is output on the output_text bus with type TEXT_TAG. In a decrypt operation the core also loads the expected tag value over the input_text bus for comparison with the calculated value.
10. In a decrypt operation only, one clock cycle after the calculated tag is output the core will set the value of authentication_success to indicate whether the input expected tag matched the calculated tag. Note that output_valid is not set to 1 when authentication_success is valid and no plaintext/ciphertext is input or output. The one clock cycle delay is necessary to ensure there is no combinational dependency between the authentication_success signal and the input_text bus which would complicate the timing when using the core.

Verification Methods

Algotronix supplies a comprehensive VHDL testbench for the G3 core 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. The AES testbench is supplied along with the GCM product so that customers can check the AES implementation thoroughly.

In the GCM product a second testbench is provided which tests the GCM specific functionality using the vectors provided in the GCM proposal document – all the test cases in the GCM proposal are implemented. Algotronix also produced a C implementation of GCM based on the well known open-source implementation of AES by Brian Gladman. This C program can generate GCM test vectors with random data and random numbers of bytes in the IV, AAD and TEXT fields in order to thoroughly test the core control logic. This testbench supplied with the core includes a file with several thousand random vectors generated by this program.

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-GCM 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 additional 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, the original GCM proposal to NIST ("The Galois/Counter Mode of Operation (GCM)" by David McGrew and John Viega) and the NIST special publication SP800-38D document describing GCM mode 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	**7**
Date	2008/08/01 15:10:16
Document	Aes Gcm Data Sheet, Xilinx Edition
Status (blank field indicates OK/no warnings)	
	(Table auto-updates, do not edit field values by hand)