algotronix ®

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

Choosing Configuration Options for the G3 AES Core

Purpose

- Provides an introduction to the various cryptographic modes of the AES algorithm
- Describes the various implementation tradeoff options provided and strategies
- Introduces implementation options which are available on request from Algotronix

Introduction

The G3 AES core provides a sophisticated range of configuration options which allow the user to customize area, performance, cryptographic mode support and keyschedule calculation to suit the requirements of their system. This application note provides an overview of the options available and provides some general guidance for users who are not specialists in cryptography on the security implications of choosing particular cryptographic modes.

This application note assumes that the reader is familiar with VHDL based design for FPGA chips and has knowledge of cryptographic terms related to the AES algorithm such as the modes of operation of the algorithm.

Cryptographic Modes

NIST Special Publication 800-38A describes several modes of use of the AES algorithm: ECB, CBC, OFB, CTR, CFB1, CFB8 and CFB128 and provides diagrams of the data flow in each mode. The G3 AES core can implement any combination of these modes. The choice of which cryptographic modes to implement will have a major impact on the area required to implement the core and some impact on the maximum clock frequency achievable. However the most important factor to consider is that the choice of cryptographic mode is critical to achieving a secure system and that each mode has particular strengths and weaknesses.

This discussion of potential security issues with various cryptographic modes is not comprehensive and Algotronix recommends that a security expert is consulted when choosing the cryptographic modes for a particular application.

ECB Mode

ECB or 'Electronic Code Book' mode is the simplest mode of operation of the Advanced Encryption Standard algorithm and is the basic building block on which the more complex modes are built. The FIPS 197 standard which describes the AES algorithm provides ECB mode operation.

Electronic Code Book refers to the fact that ECB mode can be viewed as being equivalent to a very large code book or 'one time pad'. Each 128 bit input vector is 'looked up' in the code book table and the corresponding 128 bit output vector is output and each possible cryptographic key corresponds to a different 'code book'. The code book can be considered to be 2^128 entry table of random numbers such that each possible number occurs exactly once. The characteristic of the AES algorithm ensures that there is no detectable correspondence between the input and output vectors i.e. knowledge of the input vector provides no information about the output vector unless you have

access to the code book. The advantage of the AES algorithm over a real 'code book' is that only a 128, 192 or 256 bit key is required rather than storage for 2^128 128 bit numbers or 2^135 bits of information.

Although AES in ECB mode can be viewed as approximating a 'perfect' one-time pad this is not in itself sufficient to provide a secure communications link in most applications. ECB mode must be used with extreme care. The critical deficiency of ECB mode is the basic property of a one time pad that the same input data will always encrypt to the same value. For example suppose the 128 bit input vector 0x00000000000000000000000000000000 encrypted to 0xA3 16 F4 92 37 22 98 21 EF 67 43 DD 19 56 AC 51 and occurred exactly once in the plaintext then there is no way to determine the plaintext from the ciphertext without knowledge of the key BUT if the zero input vector occurs more than once in the data to be encrypted then each occurrence will encrypt to the same value. This can be exploited in several ways by an attacker:

1. If the attacker has knowledge of the relative frequency of characters/symbols in the input then they can look at the relative frequency of various patterns occurring in the output and make deductions about what the likely input text was. Binary files such as program executables may have a small number of input values (e.g. instruction codes, or 0 values used default values in data arrays) which occur frequently. Particular phrases may occur repeatedly in an English text document resulting in repeated 128 bit text blocks.
2. Data such as images may have a particularly simple structure. For example, the gross structure of a black and white security camera image with only 0 and 1 pixels might even be recognizable after encryption.
3. If the system is designed to transmit commands (e.g. 'enable cash dispenser') and the attacker has knowledge of the structure of the data frame being transmitted and can monitor the link while observing the controlled system they could easily determine the encrypted value corresponding to the desired command.

For these reasons the main function of ECB mode is to provide a basis for more complex modes of operation – it is rarely suitable for use on its own.

CBC Mode

CBC or Cipher Block Chaining is the most commonly used of the AES simple modes. CBC adds an Initialisation Vector (IV) and feedback loop to the basic ECB mode. The feedback loop XOR's plaintext words with the previous ciphertext value before encryption, the first word in the chain (where there is no previous ciphertext) is XOR'd with the IV instead. The key benefit of the feedback loop is that the encrypted value output depends not only on the current plaintext word but also on every previous ciphertext word in the file and the IV. Thus a particular value such as 0 will encrypt to a different ciphertext each time it occurs in the file and the structure of the plaintext is completely hidden in the ciphertext.

When encrypting two files or streams of data with the same key in CBC mode without an IV the resulting ciphertext will be the same up to the point where the plaintext is different between the two files. In many applications files or streams of data are highly structured and it is likely that the initial data in two files may be the same e.g. if it contains header information. Thus, up to the point where the data starts to diverge CBC has similar weaknesses to ECB. To avoid this problem the IV is used to ensure that the very first block of ciphertext is guaranteed to be different independent of the actual data being encrypted.

There are two NIST recommended techniques for generating an IV. The first is to use a cryptographic random number generator. This is a good option where one is has been provided for other purposes in the system. Providing a hardware random number generator for the sole purpose of creating IVs would cost significant area and may not be the most cost effective option. The second recommended technique is to create a unique value or 'nonce' for each stream of data to be encrypted with a given key. Nonces are also useful to foil 'replay' attacks where an encrypted message is captured and resent – the value of this technique is apparent from the scenario where the encrypted message instructs a cash machine to supply money. This can be done using a counter and in some communications protocols may already be available as a packet number. The nonce is then encrypted using AES to create an unpredictable, unique 'random' value for the IV. The hardware implementation of the second technique is particularly efficient. The nonce is prepended to the front of the stream of data to be encrypted and the IV input to the core is connected to constant 0. Since IV=0 the first ciphertext output is the encrypted nonce which then acts as the IV and this is then transmitted to the receiver. With this technique the synthesis program may be able to delete logic in the core associated with the IV input so the core area may be reduced.

Although CBC mode as described above provides confidentiality (i.e. an attacker cannot decrypt data without knowledge of the key) it does not provide authentication (i.e. it does not provide a guarantee that the received data

has not been tampered with). Authentication is often a requirement, for example where the encrypted channel is used to control a device (e.g. an ATM) or to transmit information (such as a bank balance) where changing the value of the information in transit would compromise security. Wireless links are particularly vulnerable to 'man in the middle' attacks where an attacker listens to an encrypted communication and then retransmits it to the recipient with selected changes.

In the decrypt path for CBC mode the previous ciphertext value is XOR'd with the ECB decrypted current ciphertext to create the plaintext. This means that if an attacker changes a single bit in the previous ciphertext they will toggle the corresponding bit in the current plaintext. If the attacker understands the structure of the data being transferred (e.g. block 10 contains the bank balance) then they can control the bits in block 10 by selectively flipping bits in the ciphertext for block 9. A side effect is that the decoded value of block 9 will decrypt incorrectly and be completely random but corrupting other data in the stream may not be a concern to the attacker.

CBC mode can provide a mechanism called a Message Authentication Code or MAC for authenticating that the data has not been tampered with in transit. Any tampering with the ciphertext data between transmitter and receiver will result in a MAC calculated in the receiver being incorrect. CBC-MAC can be secure and it is often more area efficient to use the AES unit for both confidentiality and authentication than to use a separate unit implementing a hash algorithm for authentication. However, CBC-MAC must be implemented with care, there are several known attacks and it is important to take advice from an expert or research the cryptographic literature before selecting this approach. There are two particular pitfalls which are worth mentioning here:

1. If AES-CBC is used for encryption and AES-CBC-MAC for authentication then the two encryptions should use different keys. This means that twice as many AES encryptions are required for authentication plus privacy as for privacy alone.
2. The message being authenticated by CBC-MAC should have a fixed length. There are various attacks on the CBC-MAC implementations where variable length data is allowed. Schemes which try and resolve this weakness by including a data-length field in the message itself need to be designed with care.

For applications which require authentication and confidentiality the AES-CCM and AES-GCM products may be appropriate, these products implement NIST and IEEE standardized algorithms which have been subject to substantial scrutiny by the cryptographic community.

Both ECB and CBC mode require the 'forward' or encrypt AES function for encryption and the 'reverse' or decrypt AES function for decrypt. The other modes of AES, instead of passing data through the cipher use AES to generate 'random' numbers which are then XOR'd with the data. The advantage of this approach is that the encryption and decryption operations are symmetrical – XOR with a random number to produce ciphertext in the encryptor and XOR the ciphertext with the same random number in the decryptor to 'cancel it out' again. Thus both the encrypt and decrypt operations in these modes actually run the core AES algorithm in the forward direction. The algorithm for the forward direction of AES is not identical to that for the reverse direction and can be implemented in hardware more efficiently than that for the reverse direction. Thus an encrypt/decrypt circuit for ECB or CBC needs to include components for both forward and reverse AES algorithms which creates an area penalty compared to modes which use forward AES for both encrypt and decrypt.

The feedback loop in CBC mode complicates implementation of parallelization schemes where multiple encryptors are run in parallel to achieve very high performance.

CTR Mode

Counter (CTR) mode AES lends itself to parallel implementations and is commonly used in applications which involve high data rates. In CTR mode the output of a counter is fed to the AES encryptor to generate a sequence of unique 'random' 128 bit values. These values are then XOR'd with the plaintext to create ciphertext or with ciphertext to restore the plaintext.

The concern with CTR mode is to ensure that the sequence of values from the counter is used only once for each cryptographic key. For example, consider the case where the counter was set to 0 at the start of each stream of data and an attacker either knew ('known plaintext attack') or could control ('chosen plaintext attack') one particular stream of data provided to the encryptor. The attacker can eavesdrop on the communications link and obtain the ciphertext corresponding to the known plaintext. They then XOR the known plaintext with the ciphertext and this provides them with the sequence of random numbers generated by the encryptor which they store. If the system then encrypts a different data stream using the same key and the same initial value for the counter then the attacker

can simply XOR their saved file of random numbers with the ciphertext to obtain the corresponding plaintext. Thus CTR mode is completely insecure unless steps are taken to ensure that the counter values do not repeat for a given cryptographic key. Once the key is changed it is possible to start repeating counter values.

The AES block size is 128 bits but building a 128 bit counter in hardware is area expensive and slow compared with a shorter counter. In any practical system the total number of data blocks to be encrypted could be encoded in a much shorter field. Therefore in many hardware implementations, including the G3 core, the CTR field is split into a fixed register which can be loaded with a unique 'stream' or file identifier and a smaller counter which maintains a unique count within the stream. The number of bits of counter is set with a compilation parameter and must be chosen large enough to ensure that the counter will not 'wrap round' with the largest possible data set to be processed.

In CTR mode if an attacker flips a single bit in the ciphertext the corresponding bit in the plaintext will be flipped and no other data will be affected. There is no equivalent to the CBC-MAC mechanism to ensure data authentication thus, in practice, CTR mode is often combined with another data-authentication mechanism. The IEEE Wireless networking standards combine AES in CTR mode for confidentiality with AES in CBC-MAC for authentication. The GCM standard for high speed networking uses CTR mode in conjunction with a Galois Field multiplier hash function for authentication.

Designing an ad-hoc communications protocol using CTR mode should be done with extreme caution – standardized protocols such as AES-GCM or CCM have considered potential attack vectors in detail and offer a safer approach.

An interesting property of CTR mode is that bit positions in the ciphertext correspond directly with the corresponding bit position in the plaintext. In ECB or CBC mode changing one bit of plaintext will cause many, potentially all, bits of the corresponding ciphertext to change. In CTR only the corresponding bit will change. This makes it easy to deal with incomplete blocks at the end of a data transmission – normally the number of bytes to be transmitted does not map exactly into an integer multiple of 128 bit AES blocks. In CTR mode the 'additional' bits in the final incomplete block can simply be discarded.

OFB mode

OFB mode is a less commonly used mode with an Initial Value and feedback which could potentially be substituted for CBC mode.

CFB1, 8 and 128 bit mode

Cipher Feedback (CFB) mode can be implemented with input block sizes of 1, 8 and 128 bits. In CFB-1 mode only a single bit of ciphertext is produced for each AES operation on a 128 block. Similarly in CFB-8 mode only 8 bits of ciphertext are produced per 128 bit AES operation. Thus in computational terms the throughput of CFB-1 is 128 times less that most AES modes and the throughput of CFB-8 is 16 times less. The feedback circuitry in CFB-1 and CFB-8 is also relatively complex and costs significant area.

The particular advantage of the CFB1 and CFB8 modes is the ability to resynchronize the data in the event that ciphertext bits are inserted or deleted in transmission. All other modes of AES will lose synchronization completely and every decrypted block in the stream after the point where the insertion or deletion occurs will be corrupted. CFB1 or CFB8 may be reasonable choices for an application with a relatively low data rate where data corruption is likely - perhaps an encrypted UART - but in general they are less useful than the other modes.

Keyschedule Related Implementation Options

The 128, 192 or 256 bit key input to the AES unit is converted to a key schedule of 128 bit 'round keys' one for each round of the encryption operation. The number of rounds varies according to the input key size – with a 128 bit key there are 10 rounds. The algorithm to expand the cipher key to create the round keys is described in FIPS 197 and is called the keyschedule algorithm.

In AES the keyschedule is different for ECB encryption and ECB decryption. In decryption the order in which the round keys are applied is reversed. In the G3 core various optimizations mean that there are other differences between the ECB encryption and decryption round keys. This is less significant than it at first appears because all encryption modes except ECB and CBC actually run the basic AES ECB encryptor in Encrypt mode for both Encrypt and Decrypt operations. Thus only ECB and CBC mode require different keyschedules for Encrypt and Decrypt operations.

The keyschedule algorithm as specified in FIPS 197 operates in the forward direction calculating round keys in the order required by an Encryption keyschedule. To provide a decryption keyschedule the calculated keyschedule is stored in a buffer memory and read out in the opposite order. With an encryption keyschedule it is possible for the G3 hardware keyschedule unit to calculate round keys as needed – this is termed ONLINE mode. The alternative of pre-calculating a keyschedule and storing it in a memory to read out later is termed OFFLINE mode.

The relative advantages and disadvantages of the ONLINE and OFFLINE mode are as follows:

1. ONLINE requires slightly more area
2. ONLINE eliminates the latency associated with precalculating the keyschedule. Therefore it is a good choice if the key changes frequently and keyschedule latency is important to performance. ONLINE mode can be used with a 64 or 128 bit datapath in the keyschedule for high performance designs.
3. ONLINE cannot be used with ECB decrypt or CBC decrypt – therefore if either of these two modes is to be implemented the ONLINE option is not available
4. OFFLINE mode provides flexibility to run the keyschedule algorithm at a different data path width than the main data path. For example, keyschedule could be run at 32 bits and the main data path at 128 bits. In the present release of G3 only 8, 16 or 32 bit operation is available in the keyschedule in OFFLINE mode.
5. Power analysis attacks (i.e. attacks which attempt to determine the cryptographic key by analyzing noise on the power supplies to the chip) are a concern then ONLINE mode may be less suitable. Running the keyschedule continuously may make this attack more feasible.

In most cases OFFLINE is the most suitable keyschedule mode. OFFLINE is usually combined with the option to share SBoxes between the Keyschedule and main data path which results in a particularly area efficient implementation. However, opting not to share SBoxes can sometimes provide a slight performance benefit.

The Algotronix G3 core provides two other options for keyschedule generation. These options are less frequently used.

In USER mode, the hardware keyschedule unit is omitted although the keyschedule buffer memory is still provided. The user must download the full keyschedule into the keyshedule memory rather than simply providing the key. This option may be attractive when a microprocessor is available to calculate the keyschedule – although handling keys using software may be less secure than keeping them within the cryptographic hardware. The area savings from USER mode are not substantial when compared to OFFLINE mode with the keyschedule sharing the SBoxes in the main datapath. Therefore in most cases USER mode is not attractive.

In the OVERLAPPED modes: USER_OVERLAPPED and OFFLINE_OVERLAPPED the key schedule buffer memory is dual ported. This allows the new keyschedule to be loaded (in the case of USER_OVERLAPPED) or calculated while the main encryptor runs with the previous keyschedule. This can be used to 'hide' the latency of keyschedulecalculation when a key changes. This technique is area intensive and doubles the block RAM requirements for the keyschedule unit. Therefore, it is only suitable for applications where latency on key change is a paramount concern and ONLINE mode cannot be used.

Many systems require duplex operation – that is an encrypt and decrypt path operating simultaneously. A potential configuration is to share a Keyschedule unit between the Encrypt and Decrypt paths. This is possible but is less attractive than might at first appear because a stand alone keyschedule unit cannot easily share SBoxes with the

main data path in the encryptors. Algotronix has sample code for the duplex scenario with keyschedule sharing: this code is not released with the standard product but is tailored for particular application requirements and provided on request. In many cases the most area efficient approach to the duplex scenario is to build a fast encryption circuit and time share it between encrypt and decrypt paths.

Data Path Width Selection

Selection of the internal data path width is the parameter which provides the greatest control of both core area and performance. There are several important considerations:

1. The 32 bit data path width is particularly efficient. It is the 'natural' data path width for implementing the AES algorithm and gives the optimum area/performance tradeoff. The achievable clock frequency will generally be highest with a 32 bit data path width.
2. The 16 and 8 bit data path width options can be selected when the performance level required by the application is much lower than that provided by the core with a 32 bit data path. The main potential benefit of these options is saving of FPGA RAM blocks. Logic use may actually increase because some areas of the algorithm must still be implemented at 32 bits wide and multiplexing is needed to convert between the narrower and wider busses. With modern Xilinx and Altera FPGAs a single FPGA RAM block can implement sufficient SBoxes for the 16 bit data path option – therefore there is little benefit to selecting 8 bit data path. The 8 bit data path option may be useful for targeting ASICs or high end CPLDs or older FPGAs with very small RAM blocks.
3. For high performance designs 64 or 128 bit data path width can be selected. The main cost is additional RAM blocks to implement SBoxes and the wider keyschedule memory. Signal lengths within the encryptor will be increased by using a wider datapath so operating clock frequency is likely to be lower than in the 32 bit data path case. Thus overall performance is substantially increased but does not scale linearly with data path width.

The best strategy for selecting internal data path width is to try 32 bits initially. If this does not provide enough performance then try 64 or 128 bits. If it provides very much higher performance than is necessary then try 16 bits.

Algotronix has an example design for a parallelized AES unit with multiple 128 bit datapath encryptors operating in ECB mode to provide multi-gigabit performance. This code can be customized for a particular application and is supplied on request. It is used in Algotronix' AES-GCM product.

As well as providing the ability to set the data path width in the main encryption circuit using the internal_data_path_width parameter the G3 core also allows the external_data_path_width to be set. This affects the external bus width and hence the number of clock cycles it takes to load data into the encryptor and output data from the encryptor. The external data path width must be less than or equal to the internal data path width.

The external_data_path_width parameter sets the width of the logic within the modes circuitry. When some of the more complex modes are used the amount of logic cells used in the modes circuitry can be quite a significant fraction of the total logic used in the design. Choosing a lower value for external_data_path_width only increases the number of clock cycles required for I/O operations and the core implements many round calculations between each I/O operation. Setting the external_data_path_width lower than the internal_data_path_width can often provide a good area/performance tradeoff. Currently, if internal_data_path_width is lower than 32 bits then the external_data_path_width must be set equal to the internal_data_path_width – this restriction may be removed in future versions of the core.

Choosing Cipher Function and Key Length

The AES core provides three options for cipher function – Encrypt only, Decrypt Only or run time selectable Encrypt/Decrypt. In most cases the choice is clear from the application context. From a cost perspecitive ECB mode decrypt requires more area than ECB mode Encrypt and a combined Encrypt/Decrypt unit requires more area than Decrypt only. An Encrypt only design will also usually attain a higher clock frequency. This area overhead is only

relevant if ECB or CBC mode is required – all the other modes run the core ECB cipher in Encrypt mode for both Encrypt and Decrypt operations.

The FIPS 197 standard specifies three possible key lengths: 128, 192 and 256 bits. Algotronix provides a compilation option Max_Crypt_Size to specify the largest key length that the core is required to support. The compiled core can support any key length up to that value:for example if Max_Crypt_Size is 128 bits then only 128 bit keys are supported but if Max_Crypt_Size is 256 bits then all key sizes can be selected at run time. In an FPGA implementation there is a relatively modest area benefit to limiting the key size at compilation time.

The most significant effect of key size is on the number of clock cycles required per encryption operation and hence the throughput of the core. Longer keys require more rounds of AES processing and so significantly reduce throughput. It also takes additional clock cycles to load a long key and to calculate the key schedule.

Some government applications require the enhanced security provided by 192 or 256 bit keys.

Additional Implementation Options

There are a small number of additional implementation options provided through generic parameters on the core:

Implement_SBoxes_in_RAM – this parameter specifies that the AES SBoxes should be implemented using FPGA RAM blocks rather than as random logic. When targeting FPGAs this is almost always the correct choice – an exception might be if other parts of the design had used all the available RAM blocks. The capability to implement SBoxes in logic is useful to target non-FPGA technologies such as ASIC or CPLD.

Keyschedule_Shares_SBoxes – specifies that instead of creating extra SBoxes for use by the hardware keyschedule unit it will share the Sboxes in the main data path. In OFFLINE mode the keyschedule is pre-calculated and stored in a memory before encryption starts so the keyschedule and main encryption path do not need the SBoxes at the same time and can share them without affecting performance. Therefore, in OFFLINE mode this is almost always the best option - although a slightly higher clock frequency can sometimes be obtained by providing the keyschedule with its own SBoxes. In ONLINE mode and OFFLINE_OVERLAPPED mode SBox sharing is not practical because the main encryption data path is using its SBoxes at the same time as the keyschedule unit is operating and this option must be set to false.

Force_Output_Low_Until_Valid – specifies that the output_text bus should be held to constant zero except when outputting valid results. Without this option intermediate values from non-final rounds would be visible and these might be useful to an attacker. Therefore, if the output bus from the AES core is connected to an output pin of the FPGA ior might otherwise be visible to an attacker it is essential to set this parameter to TRUE. However, in the case where other elements of the system make sure that only fully encrypted data will be output area can be saved and performance improved by setting this parameter to FALSE.

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

Version Control Information	
Subversion Revision Number	**30**
Date	2010/01/04 15:13:29
Document	Choosing Configuration Options For The G3 AES Core
Status (blank field indicates OK/no warnings)	
	(Table auto-updates, do not edit field values by hand)