Encryption
Cores Overview v1.5

Introduction

This document describes the Ocean Logic encryption product line. DES, Triple DES and AES are discussed, together with their modes of operation and key expansion issues for AES. Hashing functions such SHA-1 and MD5 are also included. Examples of performances and possible applications are also given.

DES

The DES algorithm is the result of a joint effort of IBM and the NSA and was adopted as a federal standard in November 1976. It is a block cipher that encrypts and decrypts data in 64 bit blocks using a 56-bit key. The block diagram is shown below.

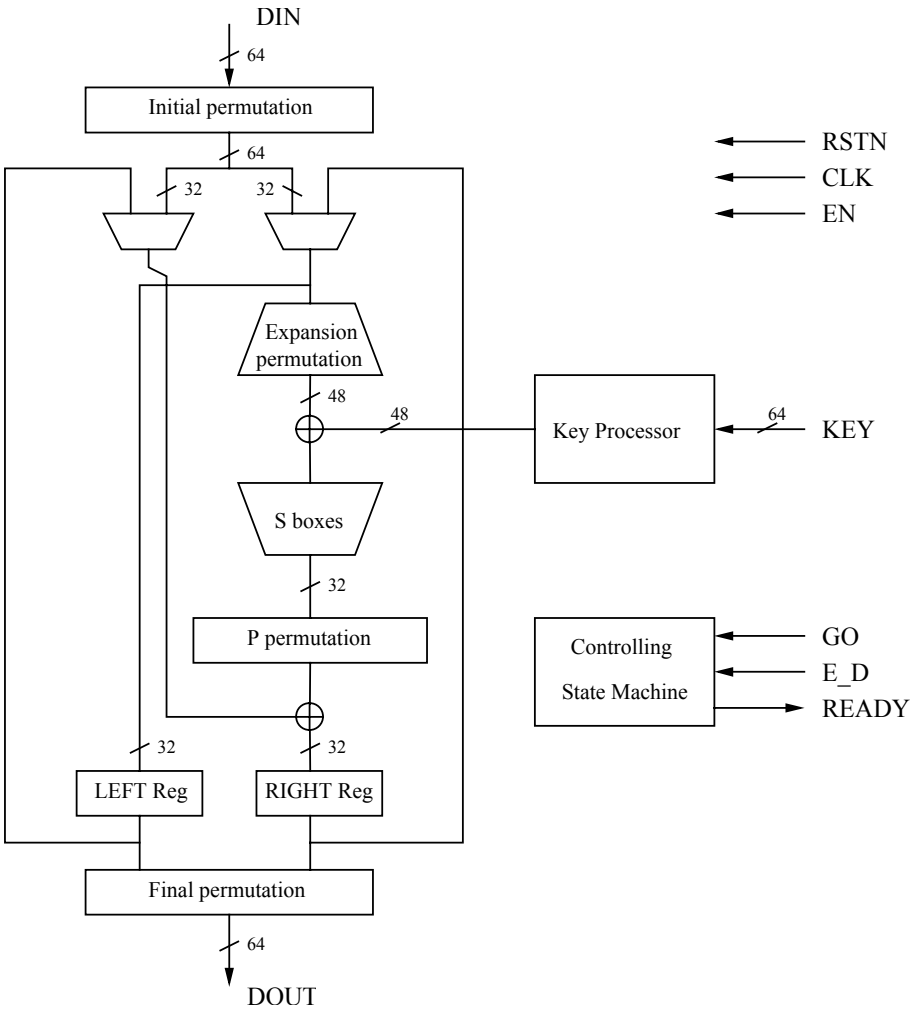

Figure 1 - Block Diagram for the DES Engine

After an initial permutation, the input data is split into two 32-bit words, left and right. This is followed by 16 rounds of identical operations.

The right word is processed with an expansion permutation and XORed with the processed key, followed by the S boxes substitution. The output of the S boxes is permuted and then XORed with the left word.

The result is used to update the right word register at the end of each round. Also, the previous right word is stored in the left word register. The processed key changes at each round as well, thanks to shifts and permutation operations.

At the end of the 16 rounds the left and right words are reassembled together and passed through the inverse of the initial permutation.

Although the security of DES has never been compromised, it is now considered relatively weak. Its short key allows exhaustive search with relatively inexpensive equipment and the algorithm should only be used for compatibility with legacy equipment.

The DES core is capable of performing a DES encryption/decryption operation in 16 cycles. Since DES operates on 64 bits data words, this translates in a sustained throughput of 4 bits/cycle.

At 100 MHz the core is therefore capable of processing 400 Mbit/s.

This performance can be sustained even when changing from encryption to decryption and while switching key.

A summary of performances of the DES core is below.

Number of cycles	Latency (cycles)	Throughput (bits/cycle)
16	16	4

Table 1 - DES core performance

Triple DES

The triple DES algorithm was proposed by IBM when it became clear that the security of the DES had been reduced by advances in computer technology.

Compared to the DES algorithm, the triple DES algorithm provides a much higher level of security.

Each triple DES encryption/decryption operation (as specified in ANSI X9.52) is a compound operation of the DES encryption and decryption operations.

A triple DES encryption operation consists in the transformation of a 64-bit block I into a 64-bit block O, defined as follows:

$$O = E_{K3}(D_{K2}(E_{K1}(I)))$$

Where the $E_K(I)$ and $D_K(I)$ represent the DES encryption and decryption of I using DES key K respectively.

Similarly, a triple DES decryption operation consists in the transformation of a 64-bit block I into a 64-bit block O, defined as follows:

$$O = D_{K1}(E_{K2}(D_{K3}(I)))$$

The standard specifies the following keying options for the keys (K1, K2, K3).

1. Keying Option 1: K1, K2, and K3 are independent keys;
2. Keying Option 2: K1 and K2 are independent keys and K3 = K1;
3. Keying Option 3: K1 = K2 = K3

In the last case, the triple DES algorithm coincides with the DES algorithm, providing backward compatibility.

There are two different versions of the DES3 core implementing the Triple DES algorithm.

The first performs each DES operation serially. Since each DES operation takes 16 cycles, a total of 48 cycles are required. This means that the sustained throughput of this version is ~1.3333 bits/cycle.

A partially pipeline version is also available, performing a Triple DES operation every 16 cycles with a sustained throughput of 4 bits/cycle. Only ECB mode is supported in this version of the core.

A summary of performances of the Triple DES core is below.

Type	Number of cycles	Latency (cycles)	Throughput (bits/cycle)	Relative size
Serial	48	48	~1.333	1
Partially pipelined	16	48	4	3

Table 2 - Triple DES core performances

AES

The AES algorithm was selected by NIST on October 20, 2000 amongst a group of competing algorithms. AES specifications are now part of the FIPS-197 specification. AES stands for Advanced Encryption Standard.

The algorithm chosen by NIST, Rijndael offers strong and secure encryption with the added flexibility of variable key block sizes.

Compared to the DES and the triple DES algorithms AES provides an even higher level of security.

An AES encryption operation consists in the transformation of a 128 bits block into a block of the same size.

The encryption key can be chosen among three different sizes: 128, 192 or 256 bit. The key is expanded during cryptographic operations.

A block diagram of the AES core is shown below.

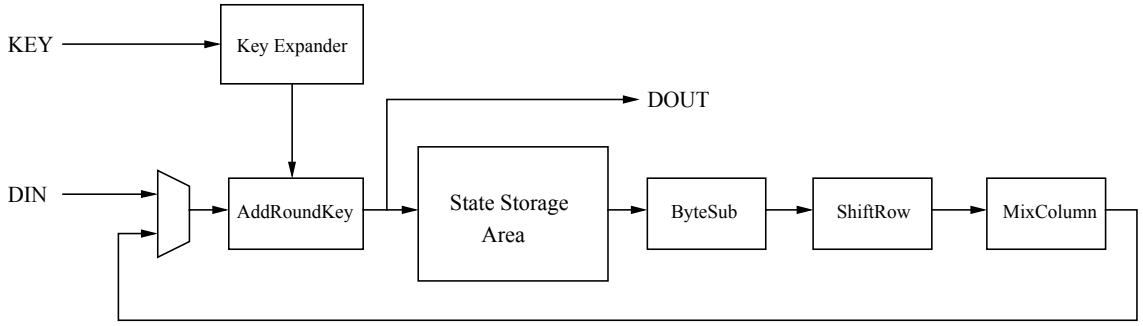

Figure 2 - AES core block diagram

The AES algorithm consists of a series of steps repeated a number of times (rounds). The number of rounds depends on the size of the key and the data block. The intermediate cipher result is known as state.

	KSIZE = 00	KSIZE = 01	KSIZE = 10
Rounds	10	12	14

Table 3 - Number of rounds as a function of key size.

Initially, incoming data and key are added together in the AddRoundKey module. The result is stored in the State Storage area. The state information is then retrieved and the ByteSub, Shiftrow, MixColumn and AddRoundKey are performed on it in the specified order. At the end of each round the new state is stored in the State Storage area. These operations are repeated according to the number of rounds.

The final round is anomalous as the MixColumn step is skipped. After the final round the cipher is output.

The AES core supports both encryption and decryption and all key sizes.

A fast version processes 128 bits at once for each round. A smaller but slower version processes 32 bits at once and it takes 4 cycles to complete a round.

The tables below summarize the performances of the AES cores.

Key size	128	192	256
Number of cycles	11	13	15
Latency (cycles)	11	13	15
Throughput (bits/cycle)	~11.63	~9.84	~8.53

Table 4 - Fast AES core performance.

Key size	128	192	256
Number of cycles	44	52	60
Latency (cycles)	44	52	60
Throughput (bits/cycle)	~2.9	~2.46	~2.13

Table 5 - Small AES core performance.

As it can be seen, the performance of the core depends on the key size as well.

The throughput of the fast core is 4 times the one of the small core. Its size is also approximately 4 times larger.

Fully pipelined AES implementations are also possible capable of over 25 Gbit/s in a modern ASIC process.

An encryption only core will be generally smaller than an encryption/decryption one.

There are many optimizations possible for AES cores that depend on the particular customer's requirement. It is important that all the requirements are discussed so that the best option is chosen.

Key expansion issues

According to the AES algorithm, a key must be expanded in order to be used during the encryption process.

Keys that are 128, 192 or 256 bits are expanded to 1408, 1664 or 1920 bits respectively.

An additional core can expand the key on the fly, as the encryption (or decryption) operation proceeds.

However, in applications where the key is not changed or not changed very often, a key can also be pre-expanded and stored in a memory.

In this case the additional key expander core is not required.

The separation of AES core and its key expander allows greater optimization and adaptability to customers' requirements.

Key expansion during decryption

During decryption (and only with ECB and CBC modes, see the "Modes of Operation" section in this document) the expanded key must be fed to the core backwards.

This can be achieved in a couple of ways. One way consists in expand the key, store it in a memory and then read it backwards. Another possibility is to use a forward/backwards key expander core.

This core can expand the key normally during encryption. However, during decryption, the core accepts the last 128, 192 or 256 bits of the expanded key and recreates all the other bits in reverse order, as required.

In the case of ASIC implementations the lack of the additional memory makes this implementation quite attractive.

In modern FPGAs, where RAM blocks are inexpensive, the other solution is often more appropriate.

A disadvantage of the Forward/backward Key Expansion method is that the last bits of the expanded key must be known. In fact, the core needs them to recreate all the other expanded key data backwards.

Even if last key bits are unknown, this core can still be used to eliminate the external memory. In fact, it is possible to fully forward expand the initial key, store only the last key data and then use these to recreate all the other expanded data backwards. Because only the last key bits are stored, the memory is no longer required.

The table below summarizes the advantages and disadvantages of both methods.

Key expansion method	Advantage	Disadvantage
Forward Key Expansion plus external memory	Smaller core.	Requires an external memory to store all the expanded key data. Bigger latency for the first key.
Forward/Backwards Key expansion	No external memory required to store all the expanded key data. Smaller latency if the last bits of the key data are known.	Larger core. The last bits of the key data must be known.

Table 6 - Pros and cons for the decryption key expansion

Backwards Key Expansion with Additional Memory in FPGAs

In FPGAs, the AES encryption/decryption cores are designed according to the "Equivalent Inverse Cipher" algorithm, as described in section 5.3.5 of the FIPS-197 specification.

Consequently, during decryption (again with ECB and CBC modes) the expanded key must be inverted. This is achieved with a small additional module (INVKEY) provided with the core.

As mentioned above, simple way of feeding the key backwards to the core is to expand it directly in an additional memory and then read it backwards when feeding it to the AES core.

The size of the additional memory will depend on the size of the key that must be expanded. As mentioned above, keys that are 128, 192 or 256 bits are expanded to 1408, 1664 or 1920 bits respectively.

For a fast core, processing 128 bits at the time, the memory will be 11x128 (total 1408), 13x128 (total 1664) and 15x128 (total 1920) respectively.

For a small core, processing 32 bits at the time, the memory will be 44x32(total 1408), 5x32 (total 1664) and 60x32 (total 1920) respectively.

A block diagram, showing one of the many possible ways of achieving this is shown below for the small AES core.

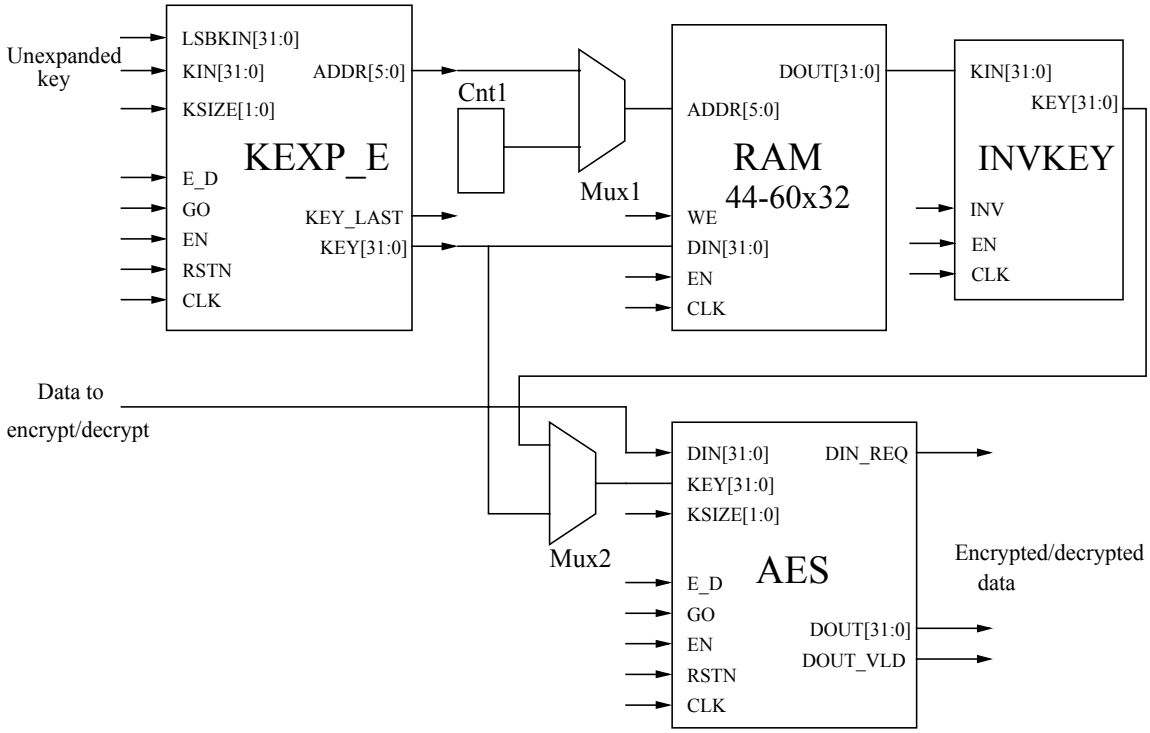

Figure 3 - Backwards key expansion with additional memory

Additional control logic is not shown. Basically, during encryption, the key is expanded on the fly by the key expander core and fed directly to the AES core.

Before decryption, the key is expanded and stored in the memory shown. During decryption the expanded key is read backward from the memory and fed to the AES core through INVKEY.

Key inversion is only required for the first and last 128 bits in the expanded key. Thus key inversion in INVKEY is controlled via the INV input.

Because of the required additional memory, this configuration is not recommended in an ASIC implementation. However, since memory blocks are readily available in modern FPGAs, this is the architecture of choice in this case.

It is worth noticing that the overall latency is increased in this case the first time a key is used. This is because the key must be fully expanded first and then fed backwards. Thus, when a new key is required, decryption cannot start until the new key has been fully expanded.

Encryption Modes of Operation

Each encryption core is available in 5 modes: ECB, CBC, CFB, OFB and CTR. These modes can be applied to either DES, Triple DES or AES.

An AES core supporting all the modes at the same time is also available. Additional modes are briefly discussed at the end of this section.

ECB Mode

ECB stands for Electronic CodeBook. In this mode, each plaintext is mapped to a unique ciphertext through encryption, like in a codebook. Each ciphertext is mapped to a unique plaintext through decryption.

This is the simplest mode and consists of the direct application of an encryption algorithm.

CBC Mode

CBC stands for Cipher Block Chaining. In this modes a feedback mechanism is introduced resulting is stronger security.

The dataflow for both encryption and decryption for CBC mode is shown below.

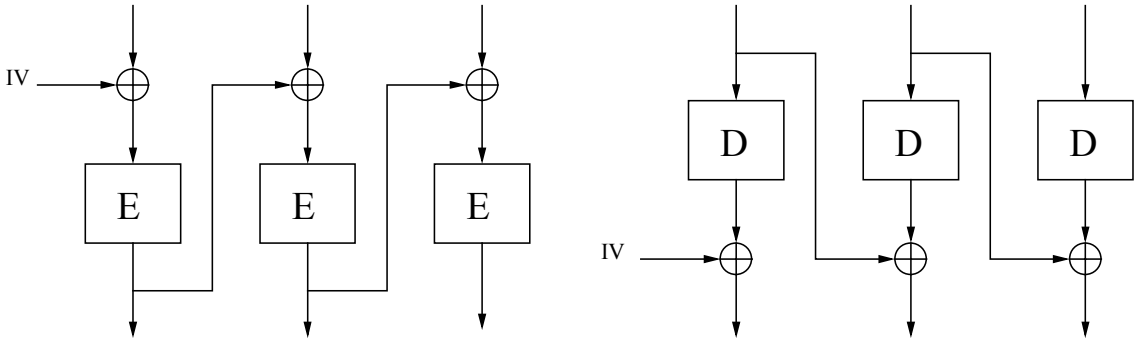

Figure 5 - CBC mode dataflow during encryption and decryption

In this mode the plaintext is XORed with previous ciphertext before being encrypted. Decryption simply reverses the encryption steps.

An Initialization Vector (IV) is required for the first cryptographic operation. IV doesn't need to be secret but it should be unique.

Unlike ECB mode, for a given key and different IVs, the same plaintext will be mapped to different a ciphertext. This represents an increase in security.

CFB Mode

CFB stands for Cipher FeedBack. Again, a feedback mechanism is added to increase security. A block diagram for this mode is shown below.

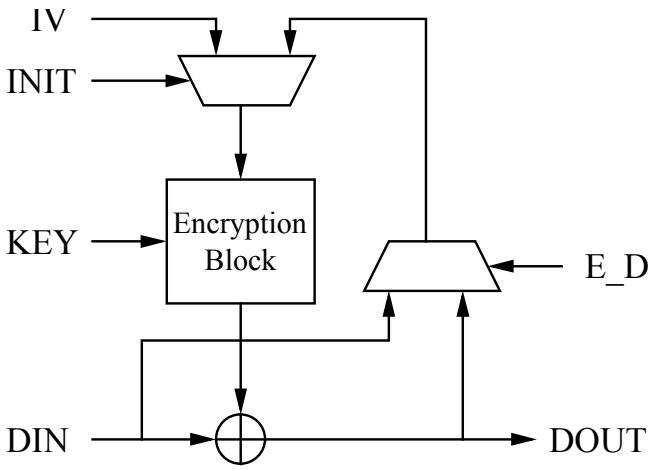

Figure 6 - CFB mode block diagram

Unlike CBC, the size of the feedback path can be chosen to fit the size of the plaintext. So, for example, in the case of CFB-8 (8 bit plaintext), only 8 bits are fed back to the encryption block input and N-8 bits are re-used. N is the encryption algorithm data block size (64 bits for DES and Triple DES and 128 bits for AES).

This has also the advantage to be able to encrypt plaintext whose size is smaller than the encryption algorithm data block size. For example, in the case of AES CFB-8, a byte can be encrypted immediately whereas for CBC and ECB modes, 16 bytes (total 128 bits) must be packed together before encryption is started.

As for CBC, IV is required. However, this **must** be unique for security reasons.

For AES CFB decryption neither backward key expansion nor INVKEY are required.

OFB Mode

OFB stands for Output FeedBack. Unlike CFB, in OFB mode the feedback path is represented by the output of the encryption block as shown in the block diagram below.

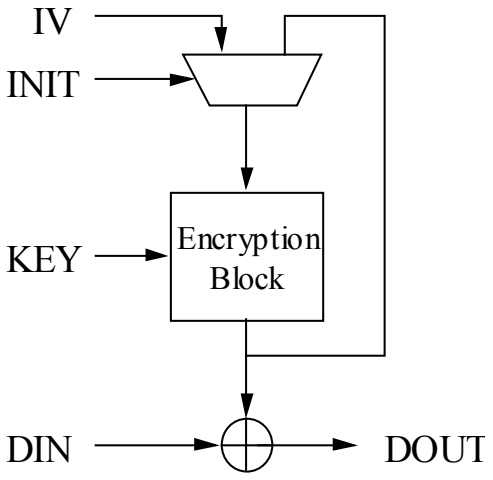

Figure 7 - OFB block diagram

Like CFB, OFB mode will allow to process plaintext whose size does not match the encryption block data size. This means, for example, that data blocks of 17 bits can be encrypted with AES which is a 128 bit data block cipher.

As for CBC and CFB, IV is required. This doesn't need to be secret but it should be unique.

For AES OFB decryption neither backward key expansion nor INVKEY are required.

CTR Mode

CTR stands for CounTeR. This mode is very similar to OFB and similar considerations are also valid here. The only difference is that the input to the encryption block is provided by a counter which is incremented with each cryptographic operation. A block diagram is shown below.

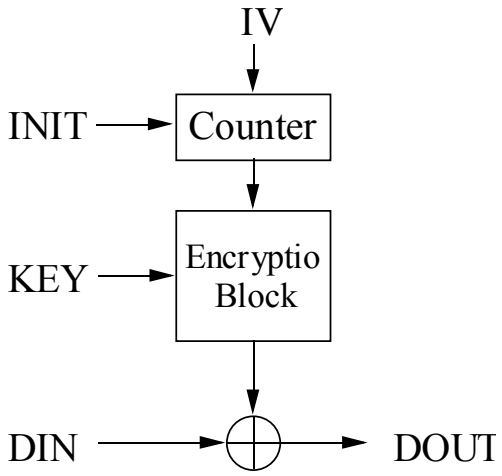

Figure 8 - CTR mode block diagram

For AES CTR decryption neither backward key expansion nor INVKEY are required.

Characteristics of the Modes of Operation

Each mode of operations has its advantages and disadvantages that make them suitable for particular applications.

The table below lists all the modes discussed and lists their advantages and disadvantages in regard to security, efficiency and fault tolerance. These considerations are valid for DES, Triple DES and AES.

Mode	Characteristic	Advantage	Disadvantage
ECB	Security		Plaintext patterns are not concealed
	Efficiency	Processing is parallelizable	
	Fault tolerance		Errors in the ciphertext affects a whole block. Synchronization errors are unrecoverable.
CBC	Security	Plaintext patterns are concealed by XORing with previous plaintext	
	Efficiency	Decryption is parallelizable	Encryption is not parallelizable
	Fault tolerance		Errors in the ciphertext affect a whole block and corresponding bits in the next block. Synchronization errors are unrecoverable.
CFB	Security	Plaintext patterns are concealed	
	Efficiency	Decryption is parallelizable	Encryption is not parallelizable

	Fault tolerance	Synchronization errors of full block size are recoverable. 1-bit CFB can recover	Errors in the ciphertext only affect corresponding bits in the plaintext and the next full block.
OFB	Security	Plaintext patterns are concealed	
	Efficiency	Processing is possible before the message is seen.	Processing is not parallelizable
	Fault tolerance	Errors in the ciphertext only affect corresponding bits in the plaintext.	Synchronization errors are unrecoverable.
CTR	Security	Plaintext patterns are concealed	
	Efficiency	Processing is parallelizable. Processing is possible before the message is seen.	
	Fault tolerance	Errors in the ciphertext only affect corresponding bits in the plaintext.	Synchronization errors are unrecoverable.

Table 8 - Pros and cons of the modes of operation

Additionally, from the efficiency standpoint, it is worth noticing that AES CFB, OFB and CTR are built using AES encryption only and therefore do not require backward key expansion or INVKEY.

Other Modes

Recently, new modes have emerged that combine encryption with authentication such as OCB, CCM or GCM. Many of these modes have not even been approved yet as standard. Most of them can be assembled relatively easily using the cores described. More information about these modes can be found in the references contained in the reference section.

One-Way Hash Functions

A one-way hash function H() operates on an arbitrary length message M, producing a fixed length hash h (h=H(M)).

The basic, desirable characteristics of a one-way has function are:

- Given M, it is easy to compute h
- Given h, it is hard to compute M such that H(M) = h
- Given M, it is hard to find another message M' such that H(M) = H(M')

The application of one-way hash functions varies from message verification and authentication to digital signatures.

The subsequent sections discuses the MD5, SHA-1 and SHA-256 hashing algorithms.

MD5

The MD5 algorithm is an improved version of MD4, created by Professor Ronald L. Rivest of MIT and is closely modeled after that algorithm. It is described in the RFC 1321 specification.

MD stands for message digest. It operates on message blocks of 512 bits for which a 128-bit (4 x 32-bit words) digest is produced.

Before processing, a message of arbitrary length is divided into blocks of 512 bits each. The message is padded so that is just 64 bits short of being a multiple of 512.

Corresponding 32-bit words of the digest from consecutive message blocks are added to each other to form the digest of the whole message. The block diagram of the core is shown below

The algorithm consists of 4 rounds, each counting 16 steps for a total of 64 steps.

A block diagram of the MD5 core is shown below.

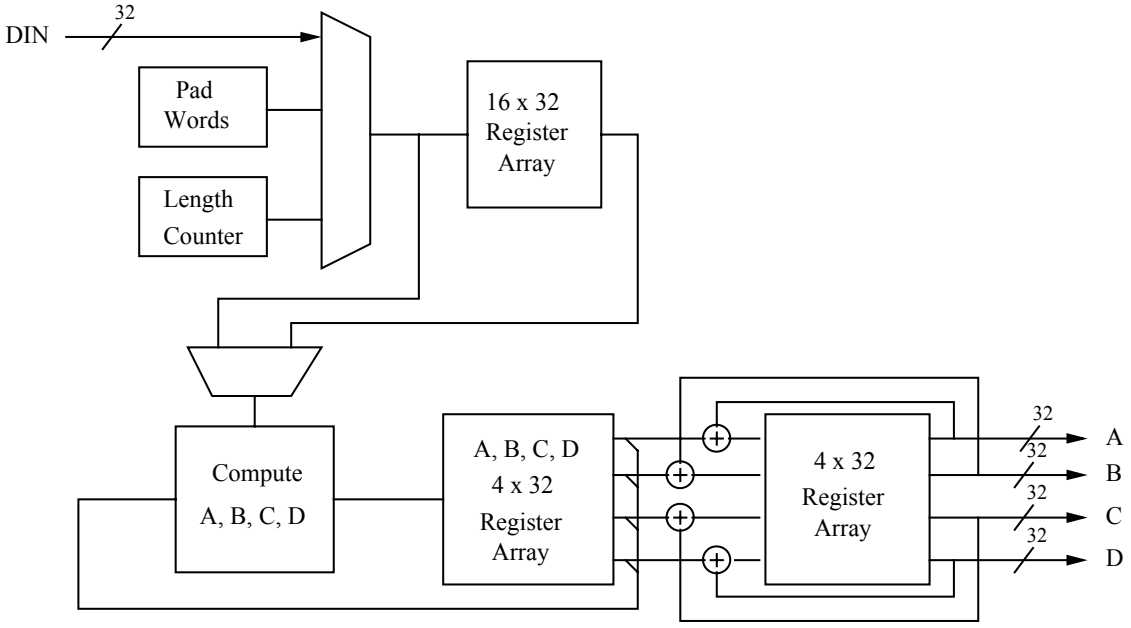

Figure 9 - Block Diagram for the MD5 processor

The core can process a block of 512 bits in 65 cycles. Thus the sustained throughput of the core is ~7.87 bits/cycle. At 100 MHz this core can digest ~787 Mbit/s.

The table below summarizes the performances of this core for a 512 bit block.

Number of cycles	Latency (cycles)	Throughput (bits/cycle)
65	65	~7.87

Table 9 - MD5 core performance

SHA-1

The SHA-1 algorithm is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 message digest algorithm, and is closely modeled after that algorithm.

SHA stands for Secure Hashing Algorithm. This algorithm is also used in the digital signature standard. It operates on message blocks of 512 bits for which a 160-bit (5 x 32-bit words) digest is produced. Padding is the same as in MD5.

Corresponding 32-bit words of the digest from consecutive message blocks are added to each other to form the hashing of the whole message.

The block diagram of the core is shown below.

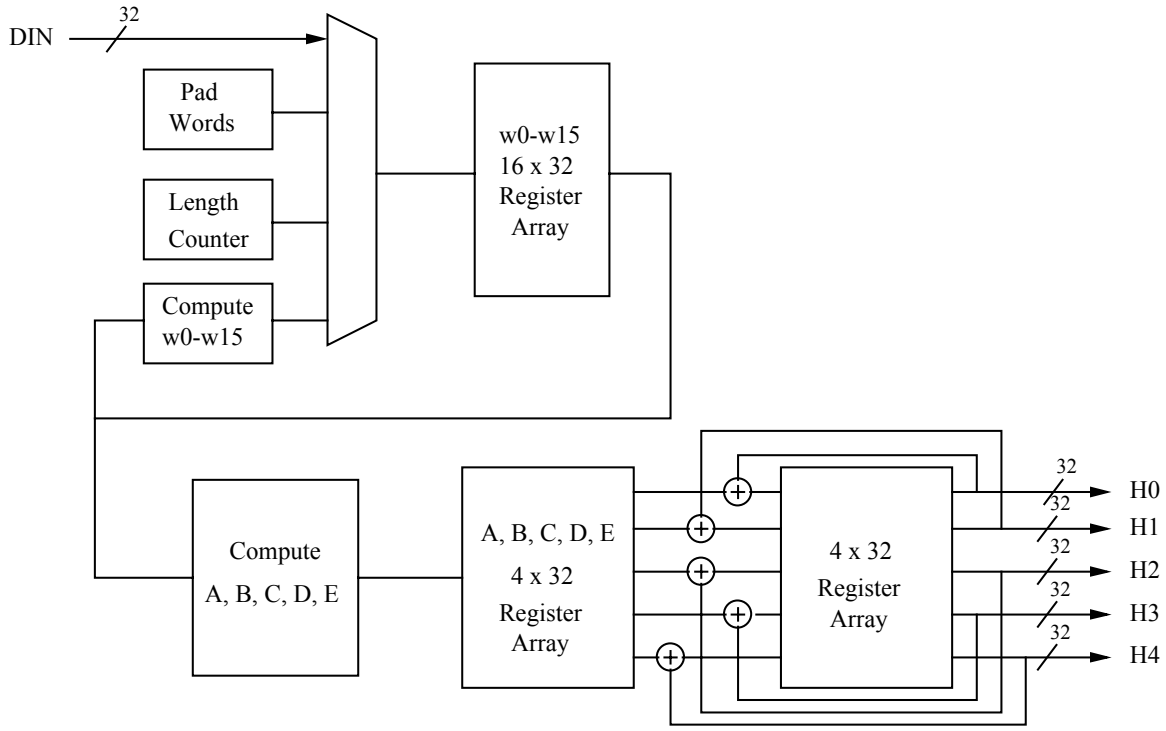

Figure 10 - Block Diagram for the SHA-1 processor

The algorithm consists of 80 steps. The core can process a 512-bit block in 81 cycles. Thus the throughput of the core is ~6.32 bits/cycle. . At 100 MHz this core can digest ~632 Mbit/s.

The table below summarizes the performances of this core for a 512-bit block.

Number of cycles	Latency (cycles)	Throughput (bits/cycle)
81	81	~6.32

Table 10 - SHA-1 core performance

SHA-256

The SHA-256 algorithm offer security improvements over SHA-1.

SHA stands for Secure Hashing Algorithm. This algorithm is also used in the digital signature standard. It operates on message blocks of 512 bits for which a 256-bit (8 x 32-bit words) digest is produced. Padding is the same as in SHA-1.

Corresponding 32-bit words of the digest from consecutive message blocks are added to each other to form the hashing of the whole message.

The block diagram of the core is shown below.

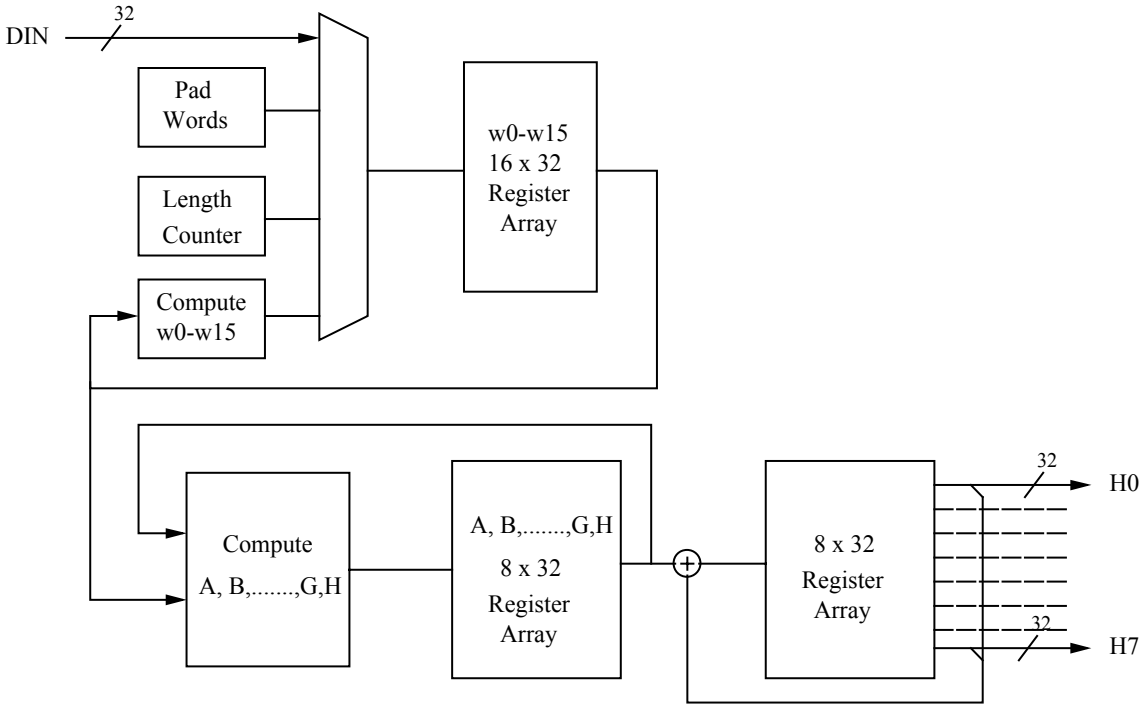

Figure 10 - Block Diagram for the SHA-256 processor

The algorithm consists of 80 steps. The core can process a 512-bit block in 65 cycles. Thus the throughput of the core is ~7.87 bits/cycle. At 100 MHz this core can digest ~787 Mbit/s.

The table below summarizes the performances of this core for a 512 bit block.

Number of cycles	Latency (cycles)	Throughput (bits/cycle)
65	65	~7.87

Table 11 – SHA-256 core performance

References

Bruce Schneier – Applied Cryptography – John Wiley & Son – ISBN 0-471-11709-9

FIPS 197 – Advanced Encryption Standard

FIPS 180-2 – Secure Hash Standard

Morris Dworkin - NIST Special Publication 800-38A – Recommendation for Block Cipher Modes of Operation

P. Rogaway, M. Bellare, J. Black and T. Krovetz - OCB:A Block-Cipher Mode of Operation for Efficient Authenticated Encryption

D. Whiting, R. Hously, N. Ferguson – Submission to NIST: Counter with CBC-MAC (CCM)

D. A. McGrew, J. Viega - The Galois/Counter Mode of Operation (GCM)

Contact Information

Ocean Logic Pty Ltd

PO BOX 768

Manly NSW 1655

Australia

E-Mail: contact@ocean-logic.com
URL: www.ocean-logic.com