FPGA Implementation for Real Time Encryption Engine for Real Time Video

Jayashri E. Patil[1], Dr. A.D. Shaligram[2]
[1] IsquareIT, ESD & VLSI department, Pune, India
Email: jayashripatil3@yahoo.com
[2] Pune University, Electronics Science Deprtment, Pune, India
Email: adshaligram@gmail.com

***Abstract*—The paper titled FPGA Implementation of Real Time Encryption Engine for Real Time Video Encryption is an attempt to implement crypto cores for three different algorithms viz. AES, 3DES and Twofish so that to achieve real time encryption and decryption of video data from a real time source. The crypto core encryptor feeds the decryptor with an encrypted data with a private key cryptographic algorithm so that uninterrupted video streaming performance can be achieved with the system. The user key for cryptographic algorithms remains fixed for both encryptor and decryptor. The output is observed for a Digital Camera VGA picture input on a VGA monitor. The multiple implementations of the crypto cores add to the flexibility in terms of future modifications to the design. Each of the private key encryption algorithms was considered separately for an HDL implementation. The implementation was tested for synthesizable components. The system was then migrated to internal pattern encryption and decryption for the fixed available pattern and was tested for the same. The last stage consisted of applying the same to the external video signal available from the Camera unit interface. Each stage was complemented by the simulation of the behavioral model. Thus six different systems were developed which form the encryption decryption pairs for each possibility with the respective algorithm in hand.**

***Index Terms*— 3DES, AES, Cryptography, Decryption, Encryption, Twofish**

I. Introduction

Cryptography is a method of storing and transmitting data in a form that only those it is intended for can read and process. It is a science of protecting information by encoding it into an unreadable format. Cryptography is an effective way of protecting sensitive information as it is stored on media or transmitted through network communication paths. Although the ultimate goal of cryptography, and the mechanisms that make it up, is to hide information from unauthorized individuals, most algorithms can be broken and the information can be revealed if the attacker has enough time, desire, and resources. So a more realistic goal of cryptography is to make obtaining the information too work-intensive to be worth it to the attacker.

Encryption is a method of transforming original data, called plain text or clear text, into a form that appears to be random and unreadable, which is called cipher text. Plaintext is either in a form that can be understood by a person (a document) or by a computer (executable code). Once it is converted into cipher text, neither human nor machine can properly process it until it is decrypted. This enables the transmission of confidential information over insecure channels without unauthorized disclosure. When data is stored on a computer, it is usually protected by logical and physical access controls. When this same sensitive information is sent over a network, it can no longer take these controls for granted, and the information is in a much more vulnerable state.

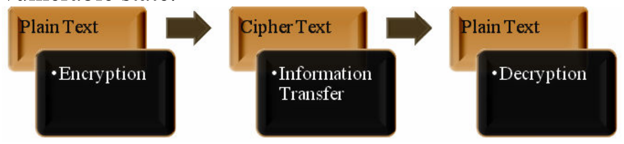


Figure1. Encryption Decryption Overview

A. Cryptosystem

A system that provides encryption and decryption is referred to as a Cryptosystem and can be created through hardware components or program code in an application. The cryptosystem uses an encryption algorithm, which determines how simple or complex the process will be. Most algorithms are complex mathematical formulas that are applied in a specific sequence to the plaintext. Most encryption methods use a secret value called a key (usually a long string of bits), which works with the algorithm to encrypt and decrypt the text, as depicted below.

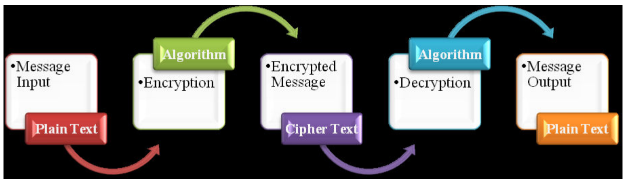


Figure2. Cryptosystem

The algorithm, the set of mathematical rules, dictates how enciphering and deciphering take place. Many algorithms are publicly known and are not the secret part of the encryption process. The way that encryption algorithms work can be kept secret from the public, but many of them are publicly known and well understood. If the internal mechanisms of the algorithm are not a secret, then something must be. The secret piece of using a well-known encryption algorithm is the key. The key can be any value that is made up of a large sequence of random bits. An algorithm contains a key-space, which is a range of values that can be used to construct a key. The key is made up of random values within the key-space range. The larger the key-space, the more available values can be used to represent different keys, and the more random the keys are, the harder it is for intruders to figure them out. A large key-space allows for more possible keys. The encryption algorithm should use the entire key-space and choose the values to make up the keys as random as possible. If a smaller key-space were used, there would be fewer values to choose from when forming a key. This would increase an attacker's chance of figuring out the key value and deciphering the protected information.

II. PRIVATE KEY ENCRYPTION ALGORITHMS

Private Key Encryption is underlined by the fact that same key is used for decryption that was used for encryption. These algorithms are characterized by their speed, reversibility and sub-key generation. These algorithms are very fast since they are much less computationally intensive. Cryptographic functions in these must, by definition, be reversible, since there is a need to be able to both encrypt and decrypt messages with the same key. The following section describes the three most important algorithms that are used in the encryption engine. They are 3DES, AES and Twofish. All of these are block ciphers with 128 bit implementations. The theoretical aspect has been described initially followed by the HDL implementations and then their applications.

A. DES

The Data Encryption Standard, or DES, is one of the most important examples of a Feistel cryptosystem. DES was the result of a contest set by the U.S. National Bureau of Standards (now called the NIST) in 1973, and adopted as a standard for unclassified applications in 1977. The DES is an example of a Feistel cipher, which operates on blocks of 64 bits at a time, with an input key of 64 bits. Every 8th bit in the input key is a parity check bit which means that in fact the key size is effectively reduced to 56 bits.

The DES is a 16-round Feistel cipher, which is preceeded and followed by an initial permutation IP and its inverse IP−1. That is, we start with a message M, and take L0R0 = IP(M) as input to the Feistel cipher, with output IP−1(R16L16). The 64-bits of the key are used to generate 16 internal keys, each of 48 bits. The steps of the round function fK is given by the following sequence, taking on 32-bit strings, expanding them to 48-bit strings, and applying a 48-bit block function.

1. Apply a fixed expansion permutation E — this function is a permutation the 32 bits with repetitions to generate a 48-bit block E(Ri).

2. Compute the bit-sum of E(Ri) with the 48-bit key Ki, and write this as 8 blocks B1, . . . ,B8 of 6 bits each.

3. Apply to each block Bj = B1B2B3B4B5B6 a substitution Sj . These substitutions are specified by S-boxes, which describe the substitution as a look-up table. The output of the substitution cipher is a 4-bit string Cj , which results in the 32-bit string C1C2C3C4C5C6C7C8.

4. Apply a fixed 32-bit permutation P to C1C2C3C4C5C6C7C8, and output the result as fKi(R). This completes the description of the round function fKi .

An iterated block cipher is a block cipher involving the repetition of an internal round function, which may involve a key as input. Each of the sequential steps is termed a round. We now describe in more detail an example of an iterated block cipher, called a Feistel cipher. In a Feistel cipher the input block is of even length 2t, of the form L0R0, and outputs ciphertext of the form RrLr. For each i such that 1 i r, the round map takes Li−1Ri−1 to LiRi, where Li = Ri−1 and Ri = Li−1fKi(Ri−1), where fKi is a cipher which depends only on an input subkey Ki, which is derived from the cipher key K. The flow of the Feistel cipher therefore looks something like:

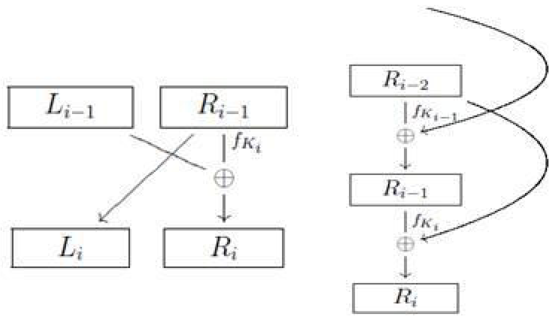


Figure3. Flow of the Feistel Chiper

The final output of the Feistel cipher is the inverted pair RrLr = RrRr−1, which allows the Feistel cipher to be inverted by running through the same algorithm with the key sequence reversed.

B. 3DES

In cryptography, Triple DES (also 3DES) is a block cipher formed from the Data Encryption Standard (DES) cipher. It was developed by Walter Tuchman (the leader of the DES development team at IBM) and is specified in FIPS Pub 46_3. There are several ways to use DES three times; not all are Triple_DES and not all are as secure. Triple DES is defined as performing a DES encryption, then a DES decryption, and then a DES encryption again. Triple DES uses a "key bundle" which comprises three DES keys, K1, K2 and K3, each of 56 bits (excluding parity bits). Each triple encryption encrypts one block of 64 bits of data. Triple-DES has a key length of 168-bits (three 56-bit DES keys), but because of an attack it has an effective key size of 112 bits. A variant uses k1 = k3, thus reducing the key size to 112 bits. This mode is susceptible to some attacks. When k1 = k2 or k2 = k3, Triple DES is reduced to single DES, and this is often used to provide backward compatibility. The use of three steps is essential to prevent meet-in-the-middle attacks; double DES would have serious vulnerabilities. The choice of decryption for the middle step (as opposed to encryption) does not affect the security of the algorithm but instead lets tools that implement triple DES interoperate with legacy single DES tools.
The standards define three keying options:

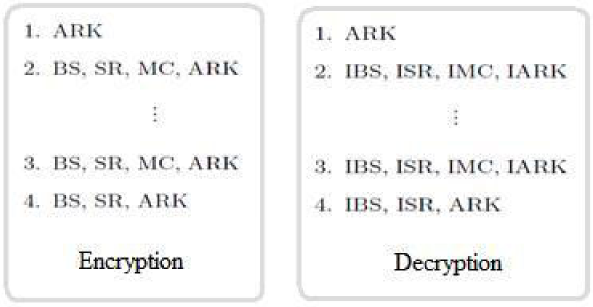
Keying option 1: All three keys are independent. Keying option 2: K1 and K2 are independent, and K3 = K1. Keying option 3: All three keys are identical, i.e. K1 = K2 = K3.
Keying option 1 is the strongest, with 3 x 56 = 168 independent key bits. Keying option 2 provides less security, with 2 x 56 = 112 key bits. This option is stronger than simply DES encrypting twice, e.g. with K1 and K2, because it protects against meet-in-the-middle attacks. Keying option 3 is no better than DES, with only 56 key bits. This option provides backward compatibility with DES, because the first and second DES operations simply cancel out. It is no longer recommended by the National Institute of Standards and Technology (NIST) and not supported by ISO/IEC 18033-3.

C. AES

In 1997, the NIST called for submissions for a new standard to replace the aging DES. The contest terminated in November 2000 with the selection of the Rijndael cryptosystem as the Advanced Encryption Standard (AES). It is described in FIPS publication 197. The Rijndael cryptosystem operates on 128-bit blocks, arranged as 4 × 4 matrices with 8-bit entries. The algorithm consists of multiple iterations of a round cipher, each of which is the composition of the following four basic steps:

• ByteSub transformation. This step is a nonlinear substition, given by a S-box (lookup table), designed to resist linear and differential cryptanalysis.
• ShiftRow transformation. Provides a linear mixing for diffusion of plaintext bits.
• MixColumn transformation. Provides a similar mixing as in the ShiftRow step.
• AddRoundKey transformation. Bitwise XOR with the round key.

We denote by BS, SR, MC, and ARK these four basic steps. There exist corresponding inverse operations IBS, ISR, IMC, IARK. The flow of the algorithms for enciphering and deciphering are as follows:



1. ARK
2. BS, SR, MC, ARK

⋮

3. BS, SR, MC, ARK
4. BS, SR, ARK

Encryption

1. ARK
2. IBS, ISR, IMC, IARK

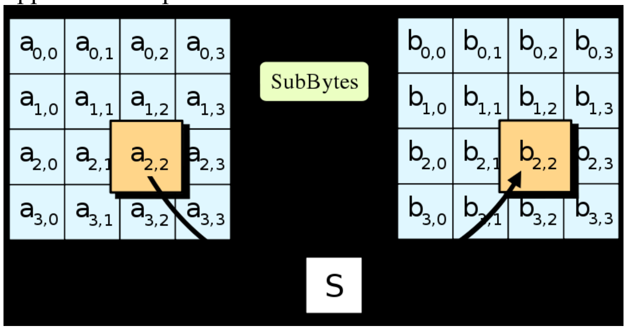
⋮

3. IBS, ISR, IMC, IARK
4. IBS, ISR, ARK

Decryption

Figure4. Flow of AES algorithm

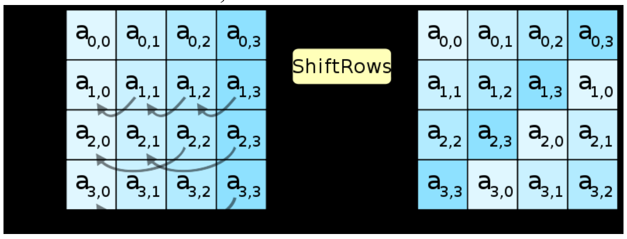
The Advanced Encryption Standard allows Rijndael with key lengths 128, 192, or 256 bits. The eight-bit byte blocks which form the matrix entries are interpreted as elements of the finite field of 28 = 256 elements. The finite field is represented by the quotient ring F28 = F2[X]/(X8 + X4 + X3 + X + 1), whose elements are polynomials c7X7 + c6X6 + c5X5 + c4X4 + c3X3 + c2X2 + c1X + c0. In the ByteSub step, each byte in the array is updated using an 8-bit substitution box, the Rijndael S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF(28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
S
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figute5. ByteSub step

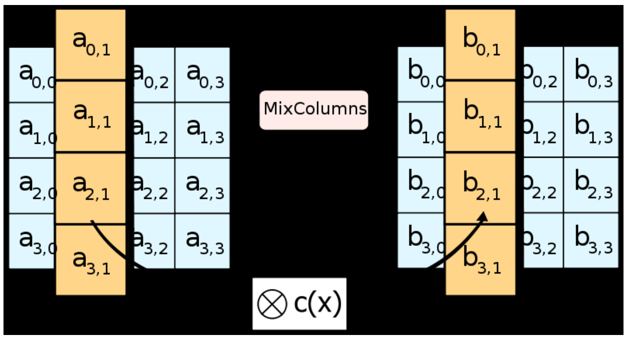
The ShiftRow step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For the block of size 128 bits and 192 bits the shifting pattern is the same. In this way, each column of the output state of the ShiftRows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). In the case of the 256-bit block, the first row is unchanged and the shifting for second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively - this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks.



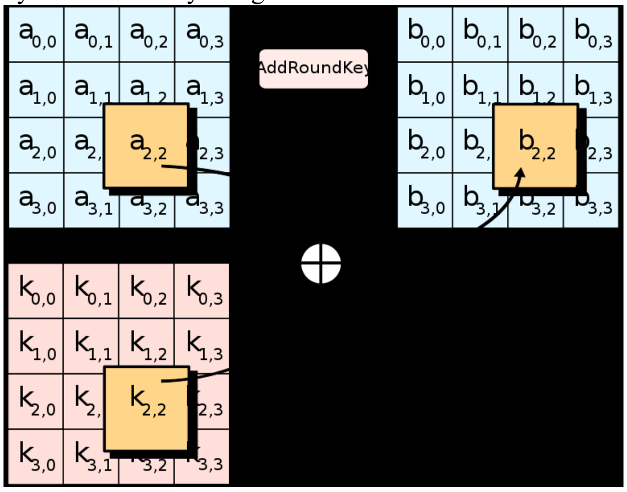
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure6. ShiftRow step

In the MixColumn step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher. Each column is treated as a polynomial over GF(28) and is then multiplied modulo x4 + 1 with a fixed polynomial c(x) = 3x3 + x2 + x + 2. The MixColumns step can also be viewed as a multiplication by a particular MDS matrix in Finite field. This process is described further in the article Rijndael mix columns.


MixColumns
⊗ c(x)
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
Figure7. MixColumn step

In the AddRoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main key using Rijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.


AddRoundKey
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
Figure8. AddRoundKey step

Implementation variations are possible that may, in many cases, offer performance or other advantages. Given the same input key and data (plaintext or ciphertext), any implementation that produces the same output (ciphertext or plaintext) as the algorithm specified in this standard, is an acceptable implementation of the AES.

D. *Twofish*

Twofish is a block cipher by Counterpane Labs, published in 1998. It was one of the five Advanced Encryption Standard (AES) finalists, and was not selected as AES. Twofish is a 128-bit block cipher that accepts a variable-length key up to 256 bits. The cipher is a 16-round Feistel network with a bijective F function made up of four key-dependent 8-by-8-bit S-boxes, a fixed 4-by-4 maximum distance separable matrix over GF(28), a pseudo-Hadamard transform, bitwise rotations, and a carefully designed key schedule.

Twofish uses a 16-round Feistel-like structure with additional whitening of the input and output. The only non-Feistel elements are the 1-bit rotates. The rotations can be moved into the F function to create a pure Feistel structure, but this requires an additional rotation of the words just before the output whitening step. The plaintext is split into four 32-bit words. In the input whitening step, these are xored with four key words. This is followed by sixteen rounds. In each round, the two words on the left are used as input to the g functions. (One of them is rotated by 8 bits first.) The g function consists of four byte-wide key-dependent S-boxes, followed by a linear mixing step based on an MDS matrix. The results of the two g functions are combined using a Pseudo- Hadamard Transform (PHT), and two keywords are added. These two results are then xored into the words on the right (one of which is rotated left by 1 bit first, the other is rotated right afterwards). The left and right halves are then swapped for the next round. After all the rounds, the swap of the last round is reversed, and the four words are xored with four more key words to produce the cipher text. In each of the 16 rounds, the first two words are used as input to the function F, which also takes the round number as input. The third word is xored with the first output of F and then rotated right by one bit. The fourth word is rotated left by one bit and then xored with the second output word of F. Finally, the two halves are exchanged.

III. FPGA SYNTHESIZABLE HDL IMPLEMENTATION

Cryptographic algorithms (and their associated keys) that are implemented in hardware are, by nature, more physically secure as they cannot easily be read or modified by an outside attacker. The downside of traditional (ASIC) hardware implementation is the lack of flexibility with respect to algorithm and parameter switch. A promising alternative for implementation block cipher are reconfigurable hardware devices such as Field Programmable Gate Arrays (FPGAs). FPGAs are hardware devices whose function is not fixed and which can be programmed in-system. The potential advantages of encryption algorithms implemented in FPGAs include:

• **Algorithm Agility:** This term refers to the switching of cryptographic algorithms during operation. The majority of modern security protocols, such as SSL or IPsec, allow for multiple encryption algorithms. The encryption algorithm is negotiated on a per-session basis. Whereas algorithm agility is costly with traditional hardware, FPGAs can be reprogrammed on-the-fly.

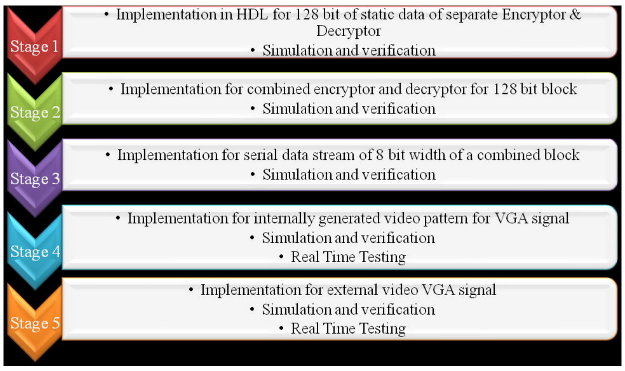
• **Algorithm Upload:** It is perceivable that yielded devices are upgraded with a new encryption algorithm which did not exist (or was not standardized!) at design time. In particular, it is very attractive for numerous security products to be upgraded for use of AES once the selection process is over. Assuming there is some kind of (temporary) connection to a network such as the Internet, FPGA- equipped encryption devices can upload the new configuration code.

• **Algorithm Modification**: There are applications which require modification of a standardized algorithm. Such modifications are easily made with reconfigurable hardware. Similarly, a standardized algorithm can be swapped with a proprietary one. Also, modes of operation can be easily changed.

• **Architecture Efficiency:** In certain cases, hardware architecture can be much more efficient if it is designed for a specific set of parameters. With FPGAs it is

possible to design and optimize architecture for a specific parameter set.
• **Throughput:** Although typically slower than ASIC implementations, FPGA implementations have the potential of running substantially faster than software implementations.
• **Cost Efficiency:** The time and costs for developing an FPGA implementation of a given algorithm are much lower than for an ASIC implementation. However, for high-volume applications, ASIC solutions usually become the more cost-efficient choice.

With these points in mind it can be concluded that the cryptographic algorithms implemented in FPGAs are much more proficient in their application. The block cipher implementations described in this report were tested for Virtex 4 series FPGA from Xilinx, Inc. namely, xc4vsx35-10ff668. The crypto cores are all implemented for 128 bit block cipher data. The section is divided as per the methodology of their development. The next sections cover the implementation details while the simulation and real time results are noted in a different section. The FPGA implementations were done using license free cores and were modified as per the requirement. The generalization of the design methodology can be described with the help of following flow chart:


Stage 1
• Implementation in HDL for 128 bit of static data of separate Encryptor & Decryptor
• Simulation and verification
Stage 2
• Implementation for combined encryptor and decryptor for 128 bit block
• Simulation and verification
Stage 3
• Implementation for serial data stream of 8 bit width of a combined block
• Simulation and verification
Stage 4
• Implementation for internally generated video pattern for VGA signal
• Simulation and verification
• Real Time Testing
Stage 5
• Implementation for external video VGA signal
• Simulation and verification
• Real Time Testing
Figure9. Design methodology
The flow was repeated for all the three algorithms and the results were comprehended. The procedure consists of simulation and testing whose results are mentioned further. The three block ciphers with symmetric encryption keys use separate cores but similar external modules for interface. They are described below.

A. 3DES
Triple DES algorithm implementation module has been developed for the 64 bit application. Since we need 128 bit encryption two parallel cores are implemented. Stage-wise descriptions for the development are given below. The important modules are listed along with their functionality.
Stage 1:
- tdes_top : Triple DES top level with three DES instances and control for encryption
- des_cipher_top : Single DES top level with encryption control
- key_schedule : Key management module for single DES
- des_top : Includes actual HDL code for DES implementation
- block_top : Block cipher implementation for 64 bit
- e_expansion_function : Expansion function for single DES
- add_key : Key adder module for single DES
- s_box : Substitution matrix box tables for DES
- p_box : Permutation matrix box tables for DES
- add_left : Adder for shift addition operations

Stage 2:
- top : Single Triple DES module with 128 bit functionality with two 64 bit tdes_top instances

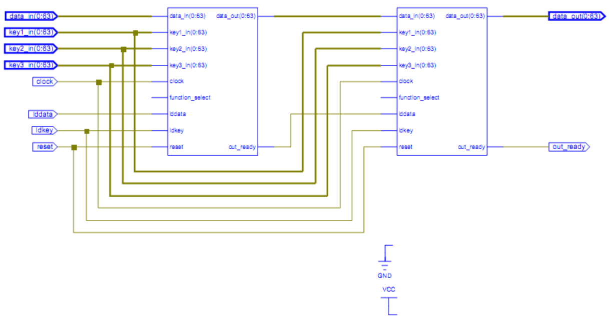


Figure10. Stage2 3DES

Stage 3:
- filter_replace : Single module with serial 8 bit data stream input and output capability with clock synchronization
- buffer_ser_par : Serial to parallel converter for 8 bit input 128 bit output data buffering
- buffer_par_ser : Parallel to serial converter for 128 bit input 8 bit output data buffering

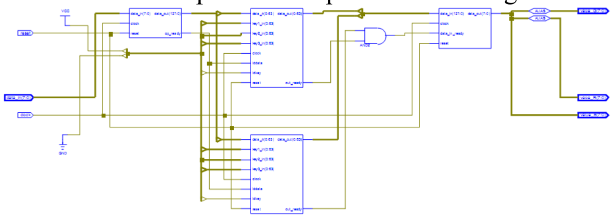


Figure11. Stage3 3DES

Stage 4:
- PR_MEAN_MEDIAN : top level module for internal pattern module

- I2C_AV_Config : Audio Video module configuration using I2C bus
- TV_to_VGA : Module to convert DAC output to VGA signals for VGA port
- CLK_PLL : Clock generator module with phase locked loop
- pattern_noise1 : Internal pattern generator module
- reconfig_filter : 8 bit stream serial encryptor decryptor module

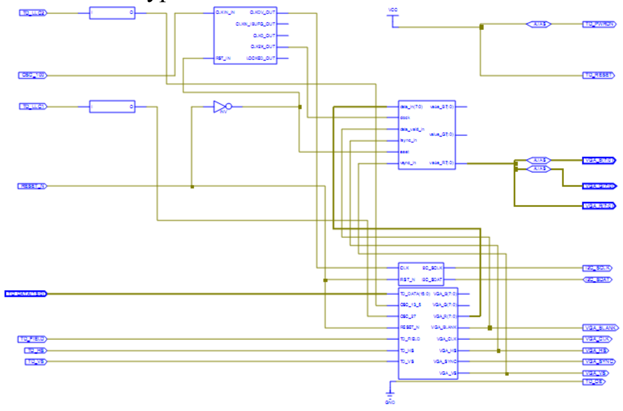


Figure12. Stage4 3DES

Stage 5:

- I2C_AV_Config : Audio Video module configuration using I2C bus
- TV_to_VGA : Module to convert DAC output to VGA signals for VGA port

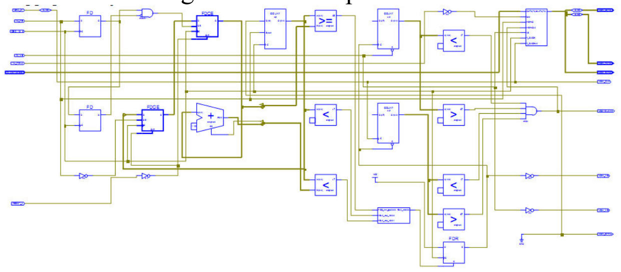


Figure13. Stage5 3DES

B. AES

Stage 1:

- aes : top level for single aes implementation
- round : top module for round specific generations
- s_box : substitution box module for aes
- shift_row : row shifter module for aes
- mix_column : column permutation module
- key_generator : round specific key generation using s_box
- control : synchronizing module for aes operation to communicate between other modules

Stage 2:

- top : single 128 bit encryptor decryptor combined module for direct implementation

Stage 3:

- filter_replace : Single module with serial 8 bit data stream input and output capability with clock synchronization
- buffer_ser_par : Serial to parallel converter for 8 bit input 128 bit output data buffering
- buffer_par_ser : Parallel to serial converter for 128 bit input 8 bit output data buffering

Stage 4:

- PR_MEAN_MEDIAN : top level module for internal pattern module
- CLK_PLL : Clock generator module with phase locked loop
- pattern_noise1 : Internal pattern generator module
- AES_replace : 8 bit stream serial encryptor decryptor module

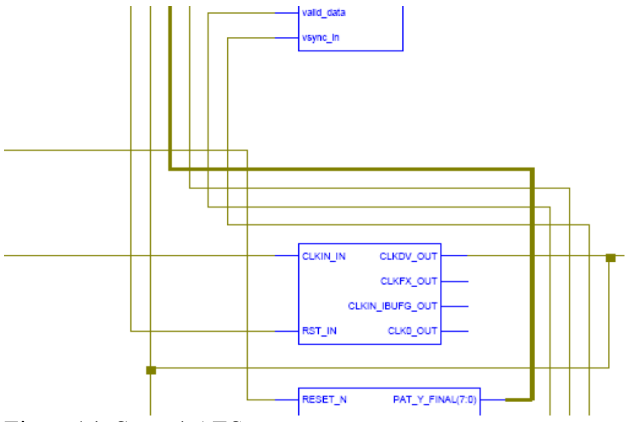


Figure14. Stage4 AES

Stage 5:

- I2C_AV_Config : Audio Video module configuration using I2C bus
- TV_to_VGA : Module to convert DAC output to VGA signals for VGA port

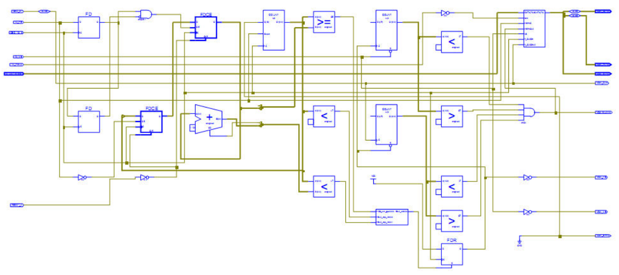


Figure15. Stage5 AES

F. *Twofish*

Twofish is a 128 bit implementation which is modified so as to enable video data encryption decryption before VGA signal generation. The Twofish algorithm is implemented in differentiated mode. Stage-wise descriptions for the development are given below. The important modules are listed along with their functionality.

Stage 1:

- core : top level module for 128 bit encryptor decryptor using Twofish algorithm
- little_endian_converter : converter module for big endian to little endian conversion of input value
- control : control module for synchronizing all elements of the core module
- keymodule : key generation module for Twofish block key generation for local usage per round
- modifiedF : modified F function module for Twofish implementation
- adder32 : 32 bit adder module for inter-round addition
- reg32 : 32 bit temporary storage register for inter-round storage
- opselect : operation selection module for functional conversion
- cleartext : module to clear text inputs from block cipher module
- ciphertext : module for cipher text generation after all rounds are completed

Stage 2:

- top : single 128 bit encryptor decryptor combined module for direct implementation
- delay : delay module for introducing synchronization delays at the control signals
- dffp : single Delay flip-flop modules for intermediate level delays

Stage 3:

- filter_replace : Single module with serial 8 bit data stream input and output capability with clock synchronization
- buffer_ser_par : Serial to parallel converter for 8 bit input 128 bit output data buffering
- buffer_par_ser : Parallel to serial converter for 128 bit input 8 bit output data buffering

Stage 4:

- PR_MEAN_MEDIAN : top level module for internal pattern module
- CLK_PLL : Clock generator module with phase locked loop
- pattern_noise1 : Internal pattern generator module
- AES_replace : 8 bit stream serial encryptor decryptor module

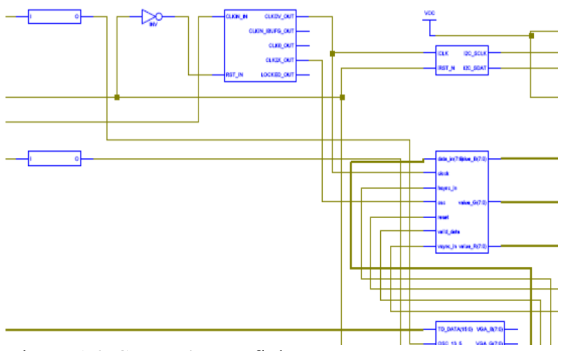


Figure16. Stage 4 Twofish

Stage 5:

- I2C_AV_Config : Audio Video module configuration using I2C bus
- TV_to_VGA : Module to convert DAC output to VGA signals for VGA port

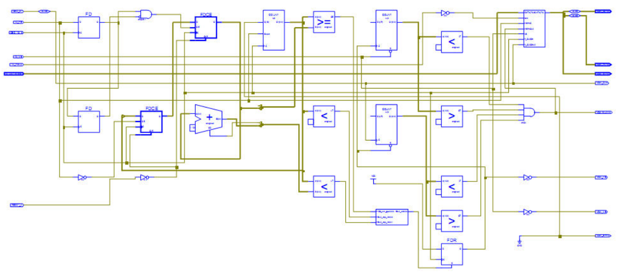


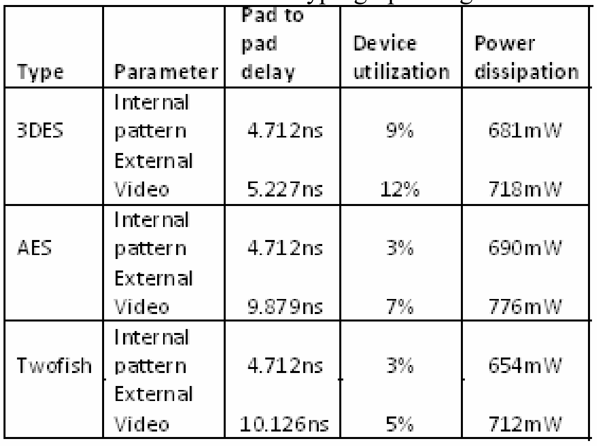
Figure17. Stage 5 Twofish

IV. Real Time Video Testing Results

Implementation of the three cores described so far has been done for Virtex 4 FPGA xc4vsx35-10ff668. These cores were tested initially for behavioral and then for post place & route simulation before alpha testing. The Simulation tools used were ModelSim 6.3(III)g Student Edition with Xilinx ISE Simulation tool. The Simulation results were verified in the same software. The actual implementation was done using Xilinx ML-402 Kit withVirtex- 4.

The real time video was inputted from the Digital Camera Samsung as single signal input. Only the Y signal which has information about the Luminance or Brightness was used as input to the video extension to the board. The output was retrieved from the VGA port and was assigned to Samsung SyncMaster color monitor with 60Hz display frequency and 640x480 pixels resolution. The results showed satisfactory levels of reproduction of the real time signal at a frequency of 200MHz for internal crypto core implementation. As is evident from the simulation, 3DES exhibits minimum delay and thus minimum distortion in terms of signal reproduction and noise. AES and Twofish reproductions are distorted at lower encryption frequencies. Rest of the implementation was run on nearest video frequency alternative of 25 MHz. The other output variables noted include maximum pad-to-pad delay for the FPGA, the power dissipation and the device utilization. These results are combined in the following table:

Table1: Comparative study of Real Time FPGA implementation with different cryptographic algorithms



Type	Parameter	Pad to pad delay	Device utilization	Power dissipation
3DES	Internal pattern	4.712ns	9%	681mW
	External Video	5.227ns	12%	718mW
AES	Internal pattern	4.712ns	3%	690mW
	External Video	9.879ns	7%	776mW
Twofish	Internal pattern	4.712ns	3%	654mW
	External Video	10.126ns	5%	712mW

From the above table it can be observed that pad to pad dealy is minimum for 3DES implementation, device utilization and power dissipation is minimum for Twofish implementation. But the recovery of external video after decryption is best with AES implementation. A 3DES and Twofish algorithm causes recovery of external video with rasters.

CONCLUSIONS

Various block ciphers can be implemented using modifications such as buffering and clock acceleration to get satisfactory response from the crypto core in terms of real time reproduction of the signals.The work is proposed for further extending the concept of real time data encryption decryption so that to add On-the-fly Partial Reconfigurability to the FPGA so as to change the device configuration at runtime and hence to foster agility and flexibility of the design. The design is also proposed to utilize key exchange and management at runtime. The algorithm for key exchange is proposed to be a private key encryption algorithm, namely SHA algorithm. The concept can be utilized in high speed real time data encryption opportunities with high level of flexibility.

REFERENCES

[1] Philippe Bulens, Fran¸cois-Xavier Standaert, Jean-Jacques Quisquater, Pascal Pellegrin, Ga¨el Rouvroy, “Implementation of the AES-128 on Virtex-5 FPGAs” , 2005.
[2] Hui QIN, Tsutomu SASAO, Yukihiro IGUCHI, “ An FPGA Design of AES Encryption Circuit with 128-bit Keys” , Dept. of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka, Japan, 2005.
[3] CoreTex Systems, “Triple-DES Encryption + Decryption Core OpenCores Specification” , LLC, 2006.
[4] Federal Information Processing Standards Publication 197, “ADVANCED ENCRYPTION STANDARD (AES)”, 2001.
[5] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, “ Twofish: A 128-Bit Block Cipher”, 1998.
[6] Federal Information Processing Standards PUB 46-3, “DATA ENCRYPTION STANDARD (DES)”, 1999.
[7] Bruce Schneier, “Applied Cryptography: Protocols, Algorithms, and Source Code in C”, Second Edition 2005.
[8] Behrouz A. Forouzan, “Cryptography and Network Security”, McGraw Hill Higher Education 2006.
[9] David R. Kohel , “Cryptography”, 2008.
[10] A. Menezes, P. van Oorschot, and S. Vanstone, “Handbook of Applied Cryptography”, CRC Press, 1996.
[11] Cryptography http://www.cccure.org/Documents/Cryptography/cisspallinone.pdf
[12] Video Signal Format http://www.kat5.tv/videoformats.html
[13] AES structural Description http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[14] 3DES Structural Description http://www.tropsoft.com/strongenc/des3.htm
[15] Twofish Structural Description http://www.schneier.com/twofish.html
[16] Clock synchronization http://fpgablog.com/posts/oregano-syn1588-ip-core/
[17] AES core http://opencores.org/project,aes_core
[18] 3DES core http://opencores.org/project,3des_vhdl
[19] Twofish core http://opencores.org/project,twofish_team