2.1 INTRODUCTION

Cryptography provides the necessary tools for accomplishing private and authenticated communication and for performing secure and authenticated transactions over the internet. In recent years, specially designed new flexible algorithms for the new protocols and applications have been introduced to face the increasing demand for cryptography. In the Third Advanced Encryption Standard candidate conference, papers from different research groups were presented [37-39]. Rijndael is announced to be Advanced Encryption Standard (AES), the new encryption standard as a replacement for DES and Triple-DES, by NIST in FIPS-197. Rijndael offers a faster hardware implementation compared to DES as well as a longer key size. Although many encryption algorithms can be efficiently implemented in software on general-purpose or embedded processors, there is still a need for special purpose cryptographic processors. The objective of this work is to enhance performance of the AES finalist Rijndael algorithm and the VLSI implementation of AES proposal is analyzed.

Different applications of the AES algorithm require different speed/area trade offs. Some applications such as WWW server and ATM are speed critical, smart cards and cellular phones require small area and other applications such as digital video recorders require an optimization of speed/area ratio. In mobile application like cellular phones, PDA’s, etc, Software implementation [1] on general-purpose processors consumes much more power than special purpose ASIC’s. The high performance is especially reached by keeping combinational paths balanced so that every clock cycle is fully utilized.
2.2 THE RIJNDAEL ALGORITHM

The Advanced Encryption Standard (AES) is a replacement to DES as the federal standard. AES has already received widespread use because of its standard definition, high security and freedom from patent entanglements. Rijndael is an iterated block cipher which supports variable block length and key length. Both lengths can be independently specified as 128, 192 or 256 bits. Rijndael has a variable number of iterations: 10, 12 and 14 for key lengths of 128, 192 and 256 respectively. In this work, a 128 bit block and key length are assumed, although the design could be adapted without difficulty to other block and key lengths.

STATE, CIPHER KEY AND NUMBER OF ITERATIONS

Transformations in Rijndael operate on an intermediate result, called the State as shown in Fig.2.1. The State can be pictured as a rectangular array of bytes. This array has 4 rows. The number of columns is denoted by Nb and is equal to the block length divided by 32. Transformations in Rijndael treat the AES standard 128-bit data block as a 4 column rectangular array of 4-byte vectors. A 128-bit plaintext has 16 bytes (B0, B1,..., B15) and it is interpreted as a State.

\[
\begin{array}{cccc}
  \text{In}_0 & \text{In}_4 & \text{In}_8 & \text{In}_{12} \\
  \text{In}_1 & \text{In}_5 & \text{In}_9 & \text{In}_{13} \\
  \text{In}_2 & \text{In}_6 & \text{In}_{10} & \text{In}_{14} \\
  \text{In}_3 & \text{In}_7 & \text{In}_{11} & \text{In}_{15} \\
\end{array}
\quad
\begin{array}{cccc}
  \text{S}_{0,0} & \text{S}_{0,1} & \text{S}_{0,2} & \text{S}_{0,3} \\
  \text{S}_{1,0} & \text{S}_{1,1} & \text{S}_{1,2} & \text{S}_{1,3} \\
  \text{S}_{2,0} & \text{S}_{2,1} & \text{S}_{2,2} & \text{S}_{2,3} \\
  \text{S}_{3,0} & \text{S}_{3,1} & \text{S}_{3,2} & \text{S}_{3,3} \\
\end{array}
\quad
\begin{array}{cccc}
  \text{O}_0 & \text{O}_4 & \text{O}_8 & \text{O}_{12} \\
  \text{O}_1 & \text{O}_5 & \text{O}_9 & \text{O}_{13} \\
  \text{O}_2 & \text{O}_6 & \text{O}_{10} & \text{O}_{14} \\
  \text{O}_3 & \text{O}_7 & \text{O}_{11} & \text{O}_{15} \\
\end{array}
\]

Fig. 2.1 Mapping of input bytes, State array and output Bytes

The cipher key is also considered to be a rectangular array with four rows, the number of column Nk being the key length divided by 32. The numbers of rounds are denoted by Nr and they depend on the values Nb and Nk. For 128,
Fig. 2.2 AES algorithm (a) Encryption Structure (b) Decryption Structure
bit blocks, Nb=4, Nk=4 and Nr =10. The encryption and decryption structure are shown in Fig 2.2. The 128-bit key Rijndael encryption algorithm consists of an initial data/key addition, then 9 round transformations followed by a final round. The Key Schedule expands the key entering the cipher so that a different round key is created for each iteration.

2.2.1 Algorithm parameters

The following algorithm parameters symbols, and functions are used throughout this standard:

**AddRoundKey ()** Transformation in the Cipher and Inverse Cipher in which a Round Key is added to the State using an XOR operation. The length of a Round Key equals the size of the State.

**RotWord ()** Function used in the Key Expansion routine that takes a four-byte word and performs a cyclic permutation.

**Shift Rows ()** Transformation in the Cipher that processes the State by cyclically shifting the last three rows of the State by different offsets.

**SubBytes ()** Transformation in the Cipher that processes the State using a nonlinear byte substitution table (S-box) that operates on each of the State bytes independently.

**SubWord ( )** Function used in the Key Expansion routine that takes a four-byte input Word and applies an S-box to produce an output word.

**InvMixColumns()** Transformation in the Inverse Cipher that is the inverse of MixColumns().

**InvShiftRows()** Transformation in the Inverse Cipher that is the inverse of ShiftRows ()

**InvSubBytes()** Transformation in the Inverse Cipher that is the inverse of SubBytes().

**Mix Columns()** Transformation in the Cipher that takes all of the columns of the State and mixes their data (independently of one another) to produce new columns.

**Rcon[ ]** The round constant word array.

**K Cipher Key**

**Nb** Number of columns (32-bit words) comprising the State. Nb = 4

**Nk** Number of 32-bit words comprising the Cipher Key. Nk = 4, 6, or 8

**Nr** Number of rounds, which is a function of Nk and Nb (which is fixed). Nr = 10

θ Multiplication of two polynomials modulo \(x^4+1\).

* Finite field multiplication.
2.3 ROUND TRANSFORMATION

The Rijndael round transformation consists of four different operations[69]. They are a ByteSub Transformation, a ShiftRow Transformation, a MixColumn Transformation and a Round Key Addition. In pseudocode, a round transformation is:

Round (State, RoundKey) {
    ByteSub (State);
    ShiftRow (State);
    MixColumn(State);
    AddRoundKey (State, RoundKey);
}

The final round does not have the Mix Column (State) transformation:

FinalRound (State, RoundKey) {
    ByteSub (State);
    ShiftRow (State);
    AddRoundKey (State, RoundKey);
}

The ByteSub Transformation is a byte substitution operated on each of the State bytes independently. The lookup table for substitution, i.e. the S-Box, is constructed by finding the multiplicative inverse of each byte in GF ($2^8$). An affine Transformation is then applied, which inverses multiplying the result by a matrix and adding to the hexadecimal number ‘63’. The inverse of ByteSub is a byte substitution using the inverse table.

The ShiftRow Transformation shifts the rows of the State cyclically. Row 0 is not shifted. Row 1 is shifted over C1 bytes, row 2 over C2 bytes and row 3 over C3 bytes where for Nb=4, C1, C2 and C3 are 1, 2 and 3 respectively. The inverse of ShiftRow is a cyclic shift of the 3 bottom rows over Nb – C1, Nb – C2, and Nb – C3 bytes respectively so that the byte at position j in row i moves to position (j + Nb – Ci) mod Nb.

The MixColumn Transformation operates on the columns of the State. Each column is considered as a polynomial over GF ($2^8$) and multiplied
modulo $x^4 + 1$ with a fixed polynomial $c(x)$, where, $c(x) = '03'x^3 + '01' x^2 + '01'x + '02'$. The inverse of MixColumn is similar to MixColumn. Every column is transformed by multiplying it with, a specific polynomial $d(x)$, where $d(x) = '0B'x^3 + '0D' x^2 + '09'x + '0E'$

In **Round Key addition**, a Round Key is applied to the State by a simple bitwise EXOR operation. The Round Key is derived from the Cipher Key by means of the key schedule operation, which would be described in detail later. The length of Round Key is equal to the block length $Nb$. AddRoundKey is its own inverse. This means when a State is EXORed with a round key to give a new State, the original State can be recovered by EXORing the new State with the same round key.

### 2.4 KEY SCHEDULE

The Round Keys, $K_i$, are derived from the Cipher Key by means of the Key Schedule and are different for each round number. The Key Schedule consists of two parts: Key Expansion and Round Key Selection. Key Expansion is a process of expanding the Cipher Key into a linear array of 4-byte words as shown in Fig.2.3. The length of this array is determined by the length of data block $Nb$, multiplied by the number of rounds $Nr$ plus 1, i.e., $Nb \times (Nr + 1)$. Thus for $Nb=4$ and $Nr=10$, the length is 44.

![Fig 2.3 Round Key Generation](image-url)

**Encryption Rounds $r_1 \ldots r_n$**
Key Expansion starts with the original key being the first $N_k$ words, say, $W_0$ to $W_3$ for $N_k = 4$. Then $W_0$ to $W_3$ are expanded to generate the next 4 words, $W_4$ to $W_7$. The iterations continue until the final 4 words $W_{40}$ to $W_{43}$ are generated. Each word $W_i$ is the EXOR of the previous word $W_{i-1}$ with the word 4 positions earlier, i.e. $W_{i-4}$. Additional operations are performed prior to expansion in word $W_i$. The words need to undergo the ByteSub, ByteRot and RCons transformations. The Rcons transformation produces output which is the EXOR result of the input with a predetermined constant. The predetermined constant is dependent on the round number of the current key.

The Round Key $K_i$ is selected from the expanded key for $W[Nb * i]$ to $W[Nb * (i + 1)]$. A 10-round design requires 11 round keys (44 words). Round key 0 is $W_0$ to $W_3$ and is used in the initial "Data/Key Addition". Round key 1 is $W_4$ to $W_7$ and used in round 1, round key 2 is $W_8$ to $W_{11}$ and used in round 2 and so on. Finally, round key 10, $W_{40}$ to $W_{43}$, is used in the final round. Round key $RK \ (0)$ is the 128-bit input key itself. $RK \ (1) = W \ (7) \ || \ W \ (6) \ ||W \ (5) \ ||W \ (4)$ is derived from round key $RK \ (0)$ as shown in Fig.2.4. Any round key $RK \ (i)$ is derived from the previous round key $RK \ (i-1)$ in a similar fashion. Rijndael round keys can be generated in two ways. In the first approach, all round keys are computed and stored and then used during the round transformation.

In the on-the-fly approach, the round keys are generated when needed during round transformation because each Rijndael round key depends only on the previous round key. The on-the-fly architecture for round key generation does not require memory for storing the round keys and does not entail pre-processing overhead. It has three multiplexers, one 16x8 SRAM, one 256x8 S-box, two 32 bit xor operators and four temporary registers. The decryption process has the same structure as the encryption architecture. The only main difference is that for every function that is used in the basic round, the mathematical inverse of it is taken. The key expansion unit performs almost the same operation with the encryption process. The only difference is that the decryption of the round keys is obtained by applying the inverse Mixcolumn to the corresponding round keys.
Fig. 2.4 Key expansion (a) Key-scheduling architecture (b) algorithm flow
2.5 HARDWARE IMPLEMENTATION

Although the strength of the algorithm, and its resistance to attack, is the primary selection criteria, speed and portability are critical. Hardware implementations of the Rijndael algorithm have recently been the object of an intensive evaluation. The major blocks of the algorithm are Key expand, control and encryption/Decryption. The speed and area trade offs of the AES algorithm can be made by changing the overall architecture of the encryption/Decryption block and by suitable implementation of individual round unit [40-43]. The basic organization of the hardware implementation of a symmetric block cipher is shown in Fig. 2.5. The design uses two major pieces, a 128b subkey generation core and an encryption pipeline. When combined with a small piece of control logic, this produces a complete encryption pipeline. The organization includes the following units:

Encryption/Decryption unit, which is used to encipher and decipher input blocks of data.

Key scheduling unit, which is used to compute a set of internal cipher keys based on a single external key.

Memory of internal keys, which is used to store internal keys computed by the key scheduling unit, or loaded to the integrated circuit through the input interface.

Input interface, used to load blocks of input data and internal keys to the circuit, and to store input blocks awaiting encryption/decryption.

Output interface, used to temporarily store output from the encryption/decryption unit and send it to the external memory.

Control unit, used to generate control signals for all other units.

The key length is limited to 128 bits. It helps to reduce the critical path of the key scheduling data path. Since the hardware cryptosystem is securely encapsulated it provides physical protection and cannot easily be read or modified. In addition the software cryptosystem may have problems in vulnerability such as trap-door and hacking. The main challenge in the hardware implementation is to maximize the encryption throughput while minimizing the area consumption at the same time.
2.6 PROPOSED ARCHITECTURE OPTIMIZATION

The Rijndael algorithm has a simple structure and can be implemented efficiently on a wide range of microprocessors, a very important feature when a fast software module is needed and when the algorithm is coded for embedded platforms. There are a lot of regularities in the design of Rijndael algorithm.

![Block diagram of hardware implementation of block cipher](image)

Therefore the critical path as well as the overall area can be minimized, with perfect VLSI design. In this section, the four Rijndael transformations focusing on their possible hardware realization is presented. Since the AddRoundKey transformation is a simple bit wise XOR between the round key and the cipher state, the hardware implementation consists of 128 two-inputs XOR gates. The ShiftRows transformation consists of a permutation of the bytes inside the cipher State. For this reason, it does not consume any silicon area and is sufficient to switch the wires corresponding to the correct bytes. In order to design the hardware cryptosystem [44], there were two possible architectures: iterative architecture or pipelined architecture. The iterative architecture is more concerned with the size of the...
physical hardware thus sacrificing the speed. Iterative architecture requires an output from previous step as input for the next step.

The pipelined architecture can increase the speed of encryption/decryption by processing multiple blocks of data simultaneously. It is realized by inserting rows of registers among combinational logic. Parts of logic between two consecutive registers form pipeline stages. Each pipeline stage is one round unit in this case. During each clock cycle, the partially processed data moves to the next stage and its place is taken by the subsequent data block. In many cases it is important to develop versions that maximize performance for minimum cost. In this work, it is intended to achieve a high speed in throughput, so the pipelined architecture was selected. The suitability of various design points depends heavily on how the cipher will be used. The full mixed inner- and outer-round pipelining should be the architecture of choice for comparing hardware performance of the AES candidates in non-feedback cipher modes.

![Diagram of One Round](image)
Fig. 2.6 Architectures for implementation of an encryption/decryption unit of a block cipher:
(a) k-stage inner-round pipelining (b) k-stage outer-round pipelining (c) Full mixed inner- and outer round pipelining
The throughput in the full mixed inner and outer-round pipelining is given by
\[ \text{Throughput}_{\text{full mixed}} = \frac{\text{block}_\text{size}}{T_{\text{CLK inner round}}(k_{\text{opt}})} \]
where \( T_{\text{CLK inner round}}(k_{\text{opt}}) \) is the delay of a single pipeline stage for the optimum number of registers introduced inside of a single round. The hardware architecture used to implement an encryption unit of a secret key cipher is shown in Fig. 2.6 a, b & c.

The round keys are generated on the fly in the encryption process. The hardware required to generate one set of round key is implemented and reused for calculating the rest of the round keys. This results in reduction of space used for storing the sub keys values and also improves the speed of operation since round key is generated simultaneously while the sub bytes, shift rows and mix columns taking place. Hence, by the time these operations get over, the round key is ready for performing the XOR operation with the output of mix column operation. Since the same hardware is used for
generating all the round keys, the amount of hardware involved in the round key generation process is reduced greatly. The key agile encryption cores offer substantial benefits in applications that change keys often while eliminating the need to store the expanded subkeys.

Fig. 2.8 Structure of AES Core
Because the subkey generation depends on the key size, with the larger key sizes requiring greater area, the 128-bit key size is selected. The cost of generating round keys through this method is far less than the cost involved in the storing the round keys in the buffer. In the S-Box implementation, the sub bytes values are already calculated and no further processing is required for sub byte calculation. Even though the S-Box implementation involves the use of a significant amount for memory space, the time required of retrieving the sub byte values from the S-Box is significantly lesser than the time involved in calculating the sub bytes through Affine transformation. The ByteSub transformation employed in both the Round Transformation and Key Schedule was implemented as a look-up table (LUT).

Since State bytes were operated individually, each Rijndael round for a 128-bit block required sixteen 256 x 8 bit LUTs. In the Key Schedule, LUTs were also used when words were passed through the S-Box. For the Key Schedule, since Round Keys were computed on-the-fly, two BRAMs were dedicated to Round Key generation. The Control Unit is a Finite State Machine (FSM) that controls other components of the core.

The Key_In and Key_Expansion processes are independent too. When a new cipher key is fed through the Key_In Register, the Key_Expansion automatically generates all the round keys and stores in an internal RAM, based on the clock signal CLK. These values will be indexed and read by the Rijndael algorithm. As these processes are independent of the Rijndael, when a new key is put, it is recommended to activate the SETUP signal, so all the states are reset and everything could start again. Fig 2.7 shows the block diagram of the AES crypto block. The data path for encryption/decryption of AES core is shown in Fig.2.8. The inherent parallelism of cryptographic core and the low level hardware features of FPGAs are exploited to enhance the performance.

2.7 VHDL SIMULATION RESULTS

The crypto processor is modeled using VHDL language and then verified with FPGA implementation. The program is simulated in ModelSim 5.7c [44-46] and the results are captured. VHDL simulation of the Rijndael
was compared to the test vectors provided in the AES submission package. The simulation results are verified. Modeling the processor using VHDL facilitates quick prototyping and modification of the target design while considering various possible trade-offs in different implementations of the crypto algorithms with differing speed and area characteristics. The primary criteria used by NIST to evaluate candidates for the new Advanced Encryption Standard (AES) include: security, efficiency in hardware and software, and flexibility. In terms of performance, the efficiency of a block cipher is Throughput (Mbits/s)/Area (slices). The test vectors provided by [48] for the algorithm was applied to assure that the design was working as intended and tested at clock frequency of 100 MHz.

The Performance comparison of proposed and other AES128 core designs is tabulated in Table 2.1. The encryption throughput of the proposed Implementation is many times higher when compared to the software implementations. Despite the strong key size, the algorithm should be faster than both DES and Triple-DES algorithms in terms of throughput. The number of cycles per encryption block is 7. The throughput is given as 2.18 Gb/sec. Hardware resources within CLB slices may not be fully utilized by the place-and-route software so as to relieve routing congestion. This results in an increase in the number of CLB slices without a corresponding increase in logic gates. To achieve a more accurate measure of chip utilization, CLB slice count was chosen as the most reliable area measurement. Therefore, to measure the hardware resource cost associated with an implementation’s resultant throughput, the Throughput per Slice (TPS) metric is used. Here the encryption rate is 2.18 Gb/sec. The number of slices used is 548.

The throughput per slice (TPS) performance is good compared to other implementations. Software performance and security of the algorithms are only two measures that will be judged by the NIST in the selection of the winner algorithm(s). Some other important performance metrics to be considered are Hardware complexity, flexibility and simplicity. Fig.2.9 shows the comparison of proposed method with commercial implementations targeting Virtex-E FPGAs.
AES -128 Core specifications:

Plaintext and Key Vector : 128 bit
State transformation : 10 Rounds
Description : FIPS 197
Mode : CBC
Device family : Virtex II
Operational parameters : Key size, Block size, Round size

![Bar chart showing comparison of Rijndael Implementations](image)

Fig. 2.9 Comparison of Rijndael Implementations (a) Throughput

**Implementation Metrics:**

- \( \text{Area} = \text{No. of Virtex slices (CLBs) used} \) \hspace{1cm} (2.1)
- \( \text{Throughput} = \frac{(128 \text{ bits} \times \text{Clock Frequency})}{(\text{Cycles Per Encrypted Block})} \) \hspace{1cm} (2.2)
- \( \text{TPS} = \frac{(\text{Encryption Rate})}{(\# \text{ CLB Slices Used})} \) \hspace{1cm} (2.3)
Table 2.1 Performance Comparison of 128-bit Rijndael Encryption Implementations

<table>
<thead>
<tr>
<th>Design</th>
<th>T(Mbs)</th>
<th>T/A</th>
<th>Area(Slices)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex[44]</td>
<td>1750</td>
<td>2.27</td>
<td>770</td>
</tr>
<tr>
<td>Amp[37]T1</td>
<td>290</td>
<td>0.68</td>
<td>421</td>
</tr>
<tr>
<td>Amp[37]T2</td>
<td>1060</td>
<td>1.85</td>
<td>570</td>
</tr>
<tr>
<td>Helion[38]</td>
<td>1190</td>
<td>2.64</td>
<td>450</td>
</tr>
<tr>
<td>A.Elbirt[42]</td>
<td>300</td>
<td>0.56</td>
<td>530</td>
</tr>
<tr>
<td>H.Kuo[43]</td>
<td>1820</td>
<td>2.53</td>
<td>730</td>
</tr>
<tr>
<td>Proposed</td>
<td>2176</td>
<td>3.98</td>
<td>548</td>
</tr>
</tbody>
</table>

Fig. 2.10 shows the comparison of proposed method with commercial implementations targeting FPGAs. It can be seen that proposed implementation provides better performance and there is a reasonable balance of both area and throughput. FPGA based solutions have shown significant speedups compared with software based approaches [41]. The timing diagram for the behavioral modeling of the AES encryption core has been drawn using the simulation tool as shown in Fig. 2.11.
2.10 Performance analysis of AES-128 core

![Performance Analysis Graph]

Fig. 2.11 AES encryption core Simulation result

33
2.8 SUMMARY

Dedicated hardware processing is a must for security implementations to be practical in terms of speed. Among the AES finalists, Rijndael is the best suited for processors with a parallel architecture, which is the prospective architecture of the future. Moreover, a dedicated hardware implementation in which the Rijndael round is fully hardwired can give very high speed. Reprogrammable devices such as Field Programmable Gate Arrays (FPGAs) are highly attractive options for hardware implementations of encryption algorithms as they provide cryptographic algorithm agility, physical security, and potentially much greater performance than software solutions [47]. A strong focus is placed on high throughput implementations, which are required to support security for current and future high bandwidth applications. A VLSI implementation for the Rijndael encryption algorithm is presented. There is a trade of between speed and use of resources. The proposed VLSI implementation of the algorithm reduces the covered area and achieves a data throughput up to 2.18Gbit/sec and 548 Slices of VirtexII-Pro FPGA.

Architectural innovations like on the fly round key generation, which facilitates simultaneous execution of sub bytes, shift rows and mix columns and round key generation has been incorporated. The combination of security, and high speed implementation, makes it a very good choice for wireless systems. The general cryptographic features of s-boxes for block cipher are considered to have high nonlinearity, strict avalanche criterion (SAC), bijection, etc. Rijndael offers a faster hardware implementation compared to DES as well as a longer keysize. Therefore, it is feasible to implement a Rijndael accelerator for virtual private network. The high performance and high flexibility of the hardware design makes it applicable to various security applications such as storage devices, embedded systems, network routers, security gateways for IPSec and SSL protocol processing. A stream cipher is another type of symmetric key algorithm and it can be designed to be much faster than block cipher. Encryption scheme based on stream cipher is discussed in the next chapter.