Implementation of AES 128-bit algorithm on FPGA boards for Altera devices uses VHDL code. At first it was recommended by Rijindael on Oct-2000 as per reference No.12 in reference section. Cryptography is the art and science of protecting information from undesirable individuals by converting it into a form non-recognizable by its attackers while stored and transmitted. The aim of this paper is to reduce the time, area and power consumption. In order to implement the pipe lined architecture is used to speed up the processing and run at very high speed. It is possible by introducing lookup tables instead of multipliers. And it is possible to construct 196 and 256 bit AES Algorithm. But it leads very complex circuitry and low end is very applicable in the present market. The AES algorithm is implemented in FPGA (Field Programmable Gate Array) for ease of operation in increased frequency. This paper utilizes the Quartus II software simulation tool and synthesizer. Quartus II Simulator tool is analogues to Electronic Design Automation (EDA) and is similar to ECAD Software. Category of Standard is Information Security Standard, Cryptography. Explanation is about The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called cipher text; . The AES algorithm is capable of using cryptographic keys of 128, 192 and 256 bits to encrypt and decrypt data in blocks of 128 bits.
1. Introduction

AES stands for Advanced Encryption Standard. It was developed after the DES Algorithm because of this fast parallel processing. The AES algorithm is implemented in FPGA (Field Programmable Gate Array) for ease of operation in increased frequency. Simulator tool is analogous to Electronic Design Automation (EDA) and is similar to ECAD Software. Category of Standard is Information Security Standard, Cryptography. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called cipher text. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

1.1 AES Overview

The below Fig 1 shows A. Lee, NIST Special Publication 800-21, Guideline for Implementing Cryptography in the Federal Government, National Institute of Standards and Technology, November 1999. Effective May 26, 2002 the National Institute of Science and Technology (NIST) has selected a block cipher called RIJNDAEL (named after its creators Vincent Rijmen and Joan Daemen) as the symmetric key encryption algorithm to be used to encrypt sensitive but unclassified American federal information. RIJNDAEL was originally a variable block (16, 24, 32 bytes) and variable key size (16, 24, 32 bytes) encryption algorithm. NIST has however decided to define AES with a block size of 16 bytes while keeping their options open to future changes. This AES algorithm was implemented so many languages like C, C++, JAVA and VHDL. It is in my project used VHDL code.

1.2 Block diagram for AES stream cipher

![AES Block Diagram](image-url)
2. Structure Of Advanced Encryption Standard

2.1 Structure of AES:

![AES Structure Diagram]

Figure 2: AES Structure

2.2. AES Algorithm

AES as well as most encryption algorithms is reversible. This means that almost the same steps are performed to complete both encryption and decryption in reverse order. The AES algorithm operates on bytes, which makes it simpler to implement and explain. This key is expanded into individual sub keys, a sub keys for each operation round. This process is called Encryption.

2.3. Encryption

As per J’arvinen K.U., Tommissa M.T., Skytta J.O.: A fully pipelined memoryless 17.8Gbps AES-128 encryptor, International Symposium on Field-Programmable Gate Arrays (FPGA 2003), Monterey, CA, 2003 Each round of processing includes one single-byte based substitution step, a row-wise permutation step, a column-wise mixing step, and the addition of the round key. The order in which these four steps are executed is different for encryption and decryption.
2.3.1. Add Round Key

i.) As Figure 3 takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words.

ii.) Start by copying key into first 4 words. Then loop creating words that depend on values in previous & 4 places back. In 3 of 4 cases just XOR these together.

iii.) 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back. The goal of the substitution step is to reduce the correlation between the input bits and the output bits at the byte level.
Each of the 16 bytes of the state is XORed against each of the 16 bytes of a portion of the expanded key for the current round as shown in Table 1. The Expanded Key bytes are never reused. So once the first 16 bytes are XORed against the first 16 bytes of the expanded key are shown in Table 1 then the expanded key bytes 1-16 are shown in Table 2 are never used again. The next time the Add Round Key function is called bytes 17-32 are XORed against the state.

The first time Add Round Key gets executed State

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td></td>
</tr>
</tbody>
</table>

The second time Add Round Key is executed

<table>
<thead>
<tr>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td></td>
</tr>
</tbody>
</table>

And so on for each round of execution. During decryption this procedure is reversed. Therefore the state is first XORed against the last 16 bytes of the expanded key, then the second last 16 bytes and so on. The method for deriving the expanded key is described.

2.3.2. Byte Sub

The SubBytes() transformation shown in Table 3 is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This Sbox which is invertible, is constructed by composing two transformations S-BOX during encryption each value of the state is replaced with the corresponding SBOX value and is quoted from Wolkerstorfer J., Oswald E., Lamberger M.: An ASIC Implementation of the AES S-Boxes, The Cryptographer’s Track at the RSA Conference, San Jose, CA, 2002.
2.3.3 AES S-Box Lookup Table

Table 3: Substitution BOX

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>63</td>
<td>7C</td>
<td>77</td>
<td>7B</td>
<td>F2</td>
<td>6B</td>
<td>6F</td>
<td>C5</td>
<td>30</td>
<td>01</td>
<td>67</td>
<td>2B</td>
<td>FE</td>
<td>D7</td>
<td>AB</td>
<td>76</td>
</tr>
<tr>
<td>1</td>
<td>CA</td>
<td>82</td>
<td>C9</td>
<td>7D</td>
<td>FA</td>
<td>59</td>
<td>47</td>
<td>F0</td>
<td>AD</td>
<td>D4</td>
<td>A2</td>
<td>AF</td>
<td>9C</td>
<td>A4</td>
<td>72</td>
<td>C0</td>
</tr>
<tr>
<td>2</td>
<td>B7</td>
<td>FD</td>
<td>93</td>
<td>26</td>
<td>36</td>
<td>3F</td>
<td>F7</td>
<td>CC</td>
<td>34</td>
<td>A5</td>
<td>E5</td>
<td>F1</td>
<td>71</td>
<td>D8</td>
<td>31</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>04</td>
<td>C7</td>
<td>23</td>
<td>C3</td>
<td>18</td>
<td>96</td>
<td>05</td>
<td>9A</td>
<td>07</td>
<td>12</td>
<td>80</td>
<td>E2</td>
<td>EB</td>
<td>27</td>
<td>B2</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>09</td>
<td>83</td>
<td>2C</td>
<td>1A</td>
<td>1B</td>
<td>6E</td>
<td>5A</td>
<td>A0</td>
<td>52</td>
<td>3B</td>
<td>D6</td>
<td>B3</td>
<td>29</td>
<td>E3</td>
<td>2F</td>
<td>84</td>
</tr>
<tr>
<td>5</td>
<td>53</td>
<td>D1</td>
<td>00</td>
<td>ED</td>
<td>20</td>
<td>FC</td>
<td>B1</td>
<td>5B</td>
<td>6A</td>
<td>CB</td>
<td>BE</td>
<td>39</td>
<td>4A</td>
<td>4C</td>
<td>58</td>
<td>CF</td>
</tr>
<tr>
<td>6</td>
<td>D0</td>
<td>EF</td>
<td>AA</td>
<td>FB</td>
<td>43</td>
<td>4D</td>
<td>33</td>
<td>85</td>
<td>45</td>
<td>F9</td>
<td>02</td>
<td>7F</td>
<td>50</td>
<td>3C</td>
<td>9F</td>
<td>A8</td>
</tr>
<tr>
<td>7</td>
<td>51</td>
<td>A3</td>
<td>40</td>
<td>8F</td>
<td>92</td>
<td>9D</td>
<td>38</td>
<td>F5</td>
<td>BC</td>
<td>B6</td>
<td>DA</td>
<td>21</td>
<td>10</td>
<td>FF</td>
<td>F3</td>
<td>D2</td>
</tr>
<tr>
<td>8</td>
<td>CD</td>
<td>0C</td>
<td>13</td>
<td>EC</td>
<td>5F</td>
<td>97</td>
<td>44</td>
<td>17</td>
<td>C4</td>
<td>A7</td>
<td>7E</td>
<td>3D</td>
<td>64</td>
<td>5D</td>
<td>19</td>
<td>73</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>81</td>
<td>4F</td>
<td>DC</td>
<td>22</td>
<td>2A</td>
<td>90</td>
<td>88</td>
<td>46</td>
<td>EE</td>
<td>B8</td>
<td>14</td>
<td>DE</td>
<td>5E</td>
<td>0B</td>
<td>DB</td>
</tr>
<tr>
<td>A</td>
<td>E0</td>
<td>32</td>
<td>3A</td>
<td>0A</td>
<td>49</td>
<td>06</td>
<td>24</td>
<td>5C</td>
<td>C2</td>
<td>D3</td>
<td>AC</td>
<td>62</td>
<td>91</td>
<td>95</td>
<td>E4</td>
<td>79</td>
</tr>
<tr>
<td>B</td>
<td>E7</td>
<td>C8</td>
<td>37</td>
<td>6D</td>
<td>8D</td>
<td>D5</td>
<td>4E</td>
<td>A9</td>
<td>6C</td>
<td>56</td>
<td>F4</td>
<td>EA</td>
<td>65</td>
<td>7A</td>
<td>AE</td>
<td>08</td>
</tr>
<tr>
<td>C</td>
<td>BA</td>
<td>7B</td>
<td>25</td>
<td>3F</td>
<td>26</td>
<td>1C</td>
<td>A6</td>
<td>B4</td>
<td>C6</td>
<td>E8</td>
<td>DD</td>
<td>74</td>
<td>1F</td>
<td>4B</td>
<td>BD</td>
<td>8B</td>
</tr>
<tr>
<td>D</td>
<td>70</td>
<td>3E</td>
<td>B5</td>
<td>66</td>
<td>19</td>
<td>03</td>
<td>F6</td>
<td>0E</td>
<td>61</td>
<td>35</td>
<td>57</td>
<td>B9</td>
<td>86</td>
<td>CI</td>
<td>1D</td>
<td>9E</td>
</tr>
<tr>
<td>E</td>
<td>E1</td>
<td>F8</td>
<td>98</td>
<td>11</td>
<td>69</td>
<td>D9</td>
<td>8E</td>
<td>94</td>
<td>9B</td>
<td>1E</td>
<td>87</td>
<td>E9</td>
<td>CE</td>
<td>55</td>
<td>28</td>
<td>DF</td>
</tr>
<tr>
<td>F</td>
<td>8C</td>
<td>A1</td>
<td>89</td>
<td>0D</td>
<td>BF</td>
<td>68</td>
<td>42</td>
<td>68</td>
<td>99</td>
<td>2D</td>
<td>0F</td>
<td>B0</td>
<td>54</td>
<td>BB</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

For example, HEX D4 would get replaced with HEX 19. Arranges the state in a matrix and then performs a circular shift for each row. This is not a bit wise shift. The circular shift just moves each byte one space over. A byte that was in the second position may end up in the third position after the shift. The circular part of it specifies that the byte in the last position shifted one space will end up in the first position in the same row. The state is arranged in a 4x4 matrix (square)

\[
\begin{array}{cccc}
byte0 & byte4 & byte8 & byte12 \\
byte1 & byte5 & byte9 & byte13 \\
byte2 & byte6 & byte10 & byte14 \\
byte3 & byte7 & byte11 & byte15 \\
\end{array}
\]

The confusing part is that the matrix is formed vertically but shifted horizontally. So the first 4 bytes of the state will form the first bytes in each row. So bytes 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 will form a matrix: This is perhaps the hardest step to both understand and explain. There are two parts to this step. The first will explain which parts of the state are multiplied against which parts of the matrix. The second will explain how this multiplication is implemented over what’s called a Galois Field.

3. AES Decryption

![Figure 5: AES Decryption](image-url)
\[ wi+5 = wi+4 \oplus wi+1 \] (1)
\[ wi+6 = wi+5 \oplus wi+2 \] (2)
\[ wi+7 = wi+6 \oplus wi+3 \] (3)

Note that except for the first word in a new 4-word grouping, each word is an XOR of the previous word and the corresponding word in the previous 4-word grouping as per Fig 5. So now we only need to figure out \( wi+4 \). This is the beginning word of each 4-word grouping in the key expansion. The beginning word of each round key is obtained by:

\[ wi+4 = wi \oplus g(wi+3) \] (4)

3.1 Architecture of FPGA

Xilinx introduced Field programmable gate arrays, or FPGAs, in 1985. Figure 6 and taken from Jarvinen K.U., Tommiska M.T., Skytt’a J.O.: A fully pipelined memoryless 17.8 Gbps AES-128 encryptor, International Symposium on Field-Programmable Gate Arrays (FPGA 2003), Monterey, CA, 2003, is conceptual models of an FPGA.1 FPGA are constructed of three basic elements: logic blocks, I/O cells, and interconnection resources. A useful analogy for an FPGA is the layout of a city. The logic blocks correspond to city blocks that are occupied by different businesses receiving products from various suppliers within the city, just as the logic blocks receive data from other logic blocks within the FPGA, and processing those products for consumption by other firms or end users, just as logic block outputs are sent to other blocks and ultimately to the device utilizing the FPGA.

3.2 Internal structure of FPGA

![Internal structure of FPGA](image-url)

Figure 7: Internal structure of FPGA.
3.3 Configurable Logic Blocks

The hearts of the FPGA lie in the CLBs. CLBs appear in rows and columns within all FPGAs and implement the logic functions desired by the programmer. Most CLBs accomplish this with a lookup table. Lookup tables (LUTs) are digital memory arrays that contain truth tables for any logic function that can be implemented by the given number of logic inputs for a CLB. The output of the CLB is then the logical result of the function recorded in the lookup table. In order to program the CLBs, truth tables be loaded into the LUTs of each CLB.

4. FPGA Implementation Of AES

A new architecture for a high speed AES encryption, decryption and combined encrypt and decryption using 128-bits key size is presented. This architecture is implemented using fully pipelining method and Järvinen K.U., Tommiska M.T., Skyttä J.O.: A fully pipelined memoryless 17.8 Gbps AES-128 encryptor, International Symposium on Field-Programmable Gate Arrays (FPGA 2003), Monterey, CA, 2003. This new architecture has shown greater performance in terms of throughput and Area comparing to previous pipeline AES Cryptography. The proposed top module of AES Encryption is similar to the existing pipelined designs, the proposed structure first uses loop unrolling to expand the 10-round operations and adds registers between rounds, forming a one pass data path for plaintexts.

4.1 Implementation Aspects

This report concerns FPGAs (Field Programmable Gate Arrays). The basic FPGA blocks, I/O, CLBs (Combinational Logic Blocks), and routing architecture, are discussed to impart a basic understanding of FPGA operation. The static RAM implementation method for the programming elements of FPGAs, An in-depth investigation of one common static RAM chip, I/O, CLB, and routing configurations used in industry

4.2 Implementation of Mixcolumn In mix Column

Four bytes in the corresponding position in the four “rows” are used for matrix multiplication in GF (2^8), which involves byte-wise multiplication and addition as per Fischer V. and Gramain F.: Resource sharing in a Rijndael implementation based on a new MixColumn and InvMixColymn relation, submitted to Electronic Letters, reference number: ELL 39 395, April 14, 2003. Byte wise additions are easily VLSI Implementation of Enhanced AES… www.ijceronline.com Open Access Journal Page 56 done by XOR, and several tricks are used for multiplications.

4.3 Altera Quartus II

The Quartus II development software provides a complete design environment for FPGA designs. Design entry using schematics, block diagrams of VHDL, and Verilog HDL. Design analysis and synthesis, fitting, assembling, and timing analysis, simulation are the important steps while performing the simulation. It provides modular Design for ease of debug and is suitable for High Speed altera devices.
5. Simulation Results:

![Simulation Results](image)

Figure 9: Simulation Results

5.1 Future Scope

One could work on selection of a larger key size which would make the algorithm more secure, and a larger input block to increase the throughput. The extra increase in area can however be tolerated. So such an algorithm with high level of security and high throughput can have ideal applications such as in multimedia communications. Furthermore study of optimization

approaches for the implementations supporting multiple key lengths and modes of operation have tremendous scope for future work.

i.) One could require to port on hardware and to be obtained at hardware outputs

ii.) In this paper I had the implementation of AES Algorithm. We can extend this implementation to image encryption and decryption using AES algorithm.

iii.) Video encryption and decryption has to be done

iv.) Encryption and decryption along with compression in which makes networks speeds

v.) Better

6. Conclusion

The above document provides with the basic information needed to implement the AES Encryption/Decryption algorithm for high speed Altera Devices. The mathematics and design reasons behind AES were purposely left out. A FPGA implementation of area-optimized AES algorithm which meets the actual application is proposed in this paper. After being coded with VHDL Hardware Description Language, the waveform simulation of the new algorithm was taken in the Quartus 9.2 platform uses EDA Simulation tool equal to ECAD Tool. Ultimately, a synthesis simulation of the new algorithm has been done. Need to dump the code into hardware for validation purpose. In this paper we had done the implementation of AES Algorithm. We can extend this implementation to image encryption and decryption using AES algorithm. It can be extended to Video encryption and decryption.

7. References