Satellite On-Board Encryption

Pokhali Sayeda Roohi Banu

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

UNIVERSITY OF SURREY

Surrey Space Centre
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

October 2007

© Pokhali Sayeda Roohi Banu 2007
To our little daughter

Sana
Summary

In the light of latest intrusions into satellite data the demand to protect the sensitive and valuable data transmitted from satellites to ground has increased and hence the need to use encryption on-board. The Advanced Encryption Standard (AES), which is a very popular choice in terrestrial communications, is slowly emerging as the preferred option in the aerospace industry including satellites.

Computing systems on-board satellites have limited power and computational resources as in terrestrial embedded systems. With these constraints in mind various implementations of the AES algorithm using different optimization techniques have been carried out on FPGAs and the implementations have been evaluated in terms of power, throughput and device area.

Satellites operate in a harsh radiation environment and consequently any electronic system used on board, including the encryption processor, is susceptible to radiation-induced faults. Hence, in addition to consuming limited resources, the encryption processor should be immune to radiation induced faults to avoid faulty data transmission to ground station. Most of the faults that occur in satellite on-board electronic devices are radiation induced bit flips called single event upsets (SEUs). A detailed novel analysis of the effect of faults on imaging and telemetry data during on-board encryption is carried out. Also the impact of faults in the data which occur during transmission to the ground station due to noisy channels is discussed and compared. In order to avoid data corruption due to SEUs a novel fault-tolerant model of the AES is presented, which is based on the Hamming error correction code. Implementation of the proposed model is carried out on FPGAs and measurements of the power and throughput overhead are presented.

Key words: Satellites, Advanced Encryption Standard (AES), AES modes, FPGA, radiation faults, fault propagation, Single Event Upset (SEU), fault tolerance, fault detection and correction and error correcting codes.

Email: R.Banu@surrey.ac.uk
Acknowledgments

First and foremost I am very grateful to the Almighty for giving me the opportunity to carry out this study.

I am deeply indebted to my supervisor, Dr. Tanya Vladimirova for her invaluable suggestions and superb guidance throughout this PhD study. Without her constant support and encouragement, at both personal and professional levels, this journey of PhD would have been just impossible to carry out.

I am very grateful to Prof. Sir Martin Sweeting for his valuable feedback on my work during this research program. I would like to say a big thank you to Karen Collar for the support and help throughout my stay at Surrey Space Centre (SSC).

I would like to thank SSC besides providing me financial support to carryout my PhD it also provided a good environment and infrastructure to work. I am also thankful to UniversitiesUK for awarding me Overseas Research Studentship (ORS) to partly fund my PhD study.

I am grateful to my family members including my dear parents, who worked extremely hard to provide us the best education, my late grand parents, who always wished and prayed for my success, my loving sisters and brother and many other family members and friends. Most importantly I would like to say thank you to my husband Fakruddin, whose unwavering support and constant encouragement helped me to stay on track with my studies. Without his rock solid support this PhD study would have been just stayed in my dreams. And finally I would like to say a huge thank you to my wonderful little daughter Sana Mohammed, who is very kind enough to allow me to carry out my PhD studies. She deserves all the credit and I don’t have any words to describe her patience at such a young age. I would like to dedicate this thesis to my lovely little angle Sana, my most precious gift.
Index

Summary ....................................................................................................................... i
Acknowledgments ....................................................................................................... ii
Index ............................................................................................................................. iii
List of Figures .............................................................................................................. vi
List of Tables ............................................................................................................... ix
Glossary of Terms ......................................................................................................... x
List of Symbols .......................................................................................................... xiv

1 Introduction .............................................................................................................. 1
  1.0 Motivation .......................................................................................................... 1
  1.1 Objectives of the Thesis .................................................................................... 4
  1.2 Novelty of the Research Work ........................................................................ 4
  1.3 Structure of the Thesis ..................................................................................... 5
  1.4 Publications ...................................................................................................... 7

2 Encryption: Introduction & Algorithms .................................................................... 9
  2.0 Introduction ...................................................................................................... 9
  2.1 Introduction to Security Services ...................................................................... 10
    2.1.1 Confidentiality .......................................................................................... 10
    2.1.2 Authentication .......................................................................................... 12
    2.1.3 Integrity ..................................................................................................... 13
  2.2 Introduction to Symmetric Key Encryption ..................................................... 14
    2.2.1 Principles of Symmetric Key Algorithms ................................................. 15
    2.2.2 Brief History of Symmetric Key Algorithms ........................................... 16
    2.2.3 Introduction to the AES Algorithm ......................................................... 16
  2.3 Encryption Using the AES Algorithm ............................................................... 19
    2.3.1 Encryption of Round Transformations ................................................... 20
    2.3.2 Key Expansion .......................................................................................... 25
  2.4 Decryption Using the AES Algorithm .............................................................. 26
    2.4.1 Straight Forward Decryption .................................................................. 27
# Index

2.4.2 Equivalent Decryption ................................................................. 28
2.4.3 Decryption of Round Transformations .......................................... 29
2.5 The AES Modes .............................................................................. 30
  2.5.1 ECB Mode .................................................................................. 30
  2.5.2 CBC Mode .................................................................................. 31
  2.5.3 OFB Mode .................................................................................. 32
  2.5.4 CFB Mode .................................................................................. 33
  2.5.5 CTR Mode .................................................................................. 33
  2.5.6 Discussion .................................................................................. 34
2.6 Implementation Approaches to AES .................................................. 35
  2.6.1 Architectural Optimization Techniques ......................................... 35
  2.6.2 Algorithmic Optimization Techniques .......................................... 38
  2.6.3 Implementation of Key Expansion ............................................... 40
2.7 Literature Survey of AES Implementations ........................................ 40
  2.7.1 Review of AES Hardware Implementations .................................... 41
2.8 Conclusions .................................................................................... 45

3 Satellite On-Board Encryption ............................................................ 47
  3.0 Introduction ................................................................................... 47
  3.1 Overview to Small Satellites ......................................................... 47
    3.1.1 Small Satellites Missions ......................................................... 49
    3.1.2 Earth Observation Small Satellite On-Board Block Diagram ........ 52
  3.2 Encryption Used in Present Earth Observation Satellites .................... 56
    3.2.1 STRV - 1d .............................................................................. 57
    3.2.2 KOMPSAT - 2 ........................................................................ 57
    3.2.3 MetOp-A ............................................................................... 58
    3.2.4 RASAT ................................................................................. 60
    3.2.5 RADARSAT-2 ....................................................................... 61
  3.3 On-Board Security Architecture for Earth Observation Small Satellites .. 61
    3.3.1 Security Services for Uplink Commands ...................................... 62
    3.3.2 Security Services for Downlink Data ......................................... 63
  3.4 Encryption of Satellite Images .......................................................... 64
    3.4.1 Encryption of Satellite Images Using AES Modes ....................... 64
  3.5 Conclusions .................................................................................. 66

4 Design Space Exploration of the AES Algorithm ................................... 67
  4.0 Introduction ................................................................................... 67
  4.1 AES Implementation: Platform and Technology ............................... 67
    4.1.1 Hardware or Software platform? ............................................... 68
    4.1.2 FPGA or ASIC? ...................................................................... 69
    4.1.3 Antifuse or SRAM FPGAs? ...................................................... 70
    4.1.4 Structure of SRAM Based FPGAs ............................................ 71
    4.1.5 Suitable Platform for the Implementation of AES for On-Board Use ... 73
    4.1.6 Overview of the Radiation Environment and Effects on Integrated Circuits ... 74
    4.1.7 Radiation Effects in SRAM Based FPGA .................................... 77
    4.1.8 Fault-Tolerant Approaches for SRAM Based FPGA Design .......... 77
    4.1.9 SEU Mitigation Techniques in SRAM Based FPGAs .................... 78
    4.1.10 Redundancy Methods for SEU Mitigation .................................. 80
Index

4.2 FPGA Development Tools & Flow ................................................. 82
4.2.2 Design Parameters ...................................................... 85
4.3.1 AES Implementations Using Algorithmic Optimizations ........ 87
4.3.2 Effect of Algorithmic Optimization .................................... 89
4.3.3 AES Implementations Using Architectural Optimizations .... 92
4.3.4 Effect of Architectural Optimization .................................... 94
4.3.5 Effect of FPGA Technology ........................................... 96
4.3.5 Discussion ................................................................. 99
4.4 Conclusions ................................................................. 99

5 Fault Tolerant Model of the AES Algorithm ......................... 101
5.0 Introduction ........................................................................... 101
5.1 Faults in Satellite Data .................................................... 102
5.2 Fault Propagation in AES Modes ....................................... 103
5.2.1 AES Modes for On-Board Use ....................................... 107
5.3 Fault Detection ................................................................. 108
5.3.1 Parity Based Fault Detection AES Model ...................... 109
5.4 Fault-Tolerant AES Model .............................................. 112
5.4.1 Model Description ....................................................... 113
5.4.2 Software Simulation .................................................... 117
5.4.3 Hardware Implementation of the Fault-Tolerant AES Model 120
5.5 System-on-a-Chip Approach to On-Board Encryption .......... 123
5.6 Conclusion ........................................................................... 127

6 Conclusions & Future Work ................................................. 129

References ..................................................................................... 134

A. Error Correction Using Hamming Codes ............................. 145

B. Fault Propagation in Satellite Data (Non-imaging Data) ........ 148

C. MixColumns & SubBytes Block Diagrams ....................... 150
List of Figures

Figure 2-1 Block Diagram of Symmetric Key Encryption .................................................. 11
Figure 2-2 Block Diagram of Asymmetric Key Encryption ........................................... 12
Figure 2-3 Adding Digital Signature to Data ................................................................. 12
Figure 2-4 AES Algorithm Flow Chart For Encryption ............................................... 20
Figure 2-5 S-Box Look Up Table (LUT) ........................................................................ 21
Figure 2-6 MixColumns Transformation ........................................................................ 23
Figure 2-7 AES Key Expansion ...................................................................................... 25
Figure 2-8 Pseudo Code for Key Expansion Algorithm ............................................... 25
Figure 2-9 Key Expansion Algorithm ............................................................................ 26
Figure 2-10 Straightforward Decryption ....................................................................... 27
Figure 2-11 Equivalent Decryption ................................................................................ 28
Figure 2-12 Block Diagram of the ECB Mode ................................................................. 31
Figure 2-13 Block Diagram of the CBC Mode ................................................................. 31
Figure 2-14 Block Diagram of the OFB Mode ................................................................. 32
Figure 2-15 Block Diagram of the CFB Mode ................................................................. 33
Figure 2-16 Block Diagram of the CTR Mode ................................................................. 33
Figure 2-17 Iterative AES Architecture ......................................................................... 36
Figure 2-18 Pipelining of the AES Algorithm ............................................................... 36
Figure 2-19 Pipelining & Sub-pipelining of AES ............................................................. 36
Figure 2-20 A Loop Unrolling AES Architecture ........................................................... 37
Figure 2-21 Block Diagram of the SubBytes and InvSubBytes Transformations .......... 45
Figure 3-1 Small Satellites Developed by SSTL ............................................................. 50
Figure 3-2 Constellation of Small Satellites (DMC) ....................................................... 51
Figure 3-3 Block Diagram of On-Board Architecture ................................................... 53
Figure 3-4 DMC Imaging Sensors .................................................................................. 54
Figure 3-5 Solid State Data Recorders (a) Power PC Based (b) Strong Arm Based .. 55
Figure 3-6 Block Diagram of Payload Data Handling in DMC Satellite ..................... 55
Figure 3-7 Multi Spectral Unit (MSC) of KOMPSAT-2 .................................................. 58
List of Figures

Figure 3-8 On-Board Encryption Block Diagram Used in METOP-A............................. 60
Figure 3-9 Block Diagram of the Proposed On-Board Security Architecture................. 63
Figure 3-10 (a) Plain Image (b) Encrypted Image using ECB (c) Encrypted Image

Using CBC, OFB, CFB & CTR ................................................................. 65
Figure 4-1 Structure of Xilinx Virtex 2 FPGA....................................................... 72
Figure 4-2 Slice of Virtex 2 FPGA........................................................................... 73
Figure 4-3 Fault Tolerant Approaches Used in SRAM Based FPGA Designs............. 78
Figure 4-4 Triple Redundancy Mitigation (a) Triple Modular Redundancy (TMR) (b)

Triple Device Redundancy (TDR) ....................................................................... 81
Figure 4-5 FPGA Design Implementation Flow ..................................................... 83
Figure 4-6 Block Diagram of the AES Encryption Core........................................... 87
Figure 4-7 Data Path of Option1 AES Encryption Core ......................................... 88
Figure 4-8 AES Data Path of the Option3 AES Core............................................... 89
Figure 4-9 Throughput Vs frequency for AES implementation Option1, 2&3............ 91
Figure 4-10 Dynamic Power Consumption Vs Frequency for the AES Implementation

Option1, 2 & 3 ......................................................................................... 91
Figure 4-11 Slice Utilization of the Virtex 2 XC2V1000 FPGA for AES

Implementation Options1,2& 3 ......................................................................... 91
Figure 4-12 Block Diagram of the Pipelined Implementation of the AES.................. 93
Figure 4-13 Block Diagram of the Sub-Pipelined implementation of the AES........... 93
Figure 4-14 Dynamic Power Consumption Vs Frequency for Option1 on Different

Family FPGAs ....................................................................................... 97
Figure 4-15 Total Power Consumption Vs Frequency for Option1 on Different Family

FPGAs ................................................................................................. 97
Figure 4-16 Share of Static Power Consumption in the Total Power of Option1 AES

Implementation on a Spatran 3 FPGA (a) 25 MHz (b) 75 MHz............................... 98
Figure 5-1 Fault Propagation during Encryption in CBC Mode............................... 104
Figure 5-2 Transmission Fault Propagation in CBC Mode....................................... 104
Figure 5-3 (a) Plain Image (b) Decrypted image with SEU at 20,000th Block (c)

Decrypted Image with SEU at 40,000th Block .................................................... 105
Figure 5-4 Propagation of Transmission Fault in CFB Mode.................................... 106
Figure 5-5 Flow Chart of Fault Detection in AES................................................... 112
Figure 5-6 Fault Detection and Correction Flow Chart......................................... 116
Figure 5-7 JAVA GUI To Simulate AES Encryption.............................................. 118
Figure 5-8 JAVA GUI To Simulate AES Decryption........................................... 118
Figure 5-9 JAVA GUI to simulate of fault injection and detection at 'bit' level ...... 119
Figure 5-10 JAVA GUI to simulate of fault injection and detection at 'bit' level ..... 120
Figure 5-11 Block Diagram of the Fault Tolerant AES Datapath........................ 122
Figure 5-12 Generic Architecture of Embedded IP Core-Based SOC ................. 124
Figure 5-13 Block Diagram of SOC Based Encryption Approach to DMC Payload 127
List of Tables

Table 2-1 Security Services, Mechanisms and Algorithms ............................................... 14
Table 2-2 Characteristics of Popular Symmetric Key Algorithms ....................................... 15
Table 2-3 Round Constant Values ......................................................................................... 26
Table 2-4 Characteristics of the AES Modes ........................................................................ 34
Table 2-5 Implementation Options of the AES Algorithm ..................................................... 39
Table 2-6 Software Implementations of the AES ................................................................. 41
Table 2-7 ASIC Implementations of the AES ...................................................................... 42
Table 2-8 FPGA Implementations of the AES ...................................................................... 43
Table 3-1 Classification of Satellites .................................................................................... 49
Table 3-2 Summary of the Use of Encryption in Current Satellite Missions ......................... 56
Table 4-1 Types of SEU Upsets in SRAM Based FPGAs ....................................................... 79
Table 4-2 AES Implementations using Xilinx Virtex 2 Device XC2V1000 (On-line Key Expansion) ................................................................................................................. 92
Table 4-3 The AES Implementations with Pipelining (128-bit Data Path & Off-Line Key Expansion) ................................................................................................................. 95
Table 4-4 Logic Utilization of the Pipelined Implementations ............................................... 96
Table 4-5 Different Xilinx Family FPGA Devices Used for the AES Implementation .......... 96
Table 5-1 Fault Propagation Due to Single Bit Errors During Encryption and Transmission ................................................................................................................................. 108
Table 5-2 Hamming Code Bit Match Table To Locate A Faulty Bit ..................................... 117
Table 5-3 FPGA Implementation of the Fault Tolerant Model with Option1 AES .......... 122
Table 5-4 FPGA Implementation of the Fault Tolerant Model with Option3 AES .......... 123
## Glossary of Terms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>AHB</td>
<td>Advanced High performance Bus</td>
</tr>
<tr>
<td>APB</td>
<td>Advanced Peripheral Bus</td>
</tr>
<tr>
<td>API</td>
<td>Application Programmer Interface</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuits</td>
</tr>
<tr>
<td>AU</td>
<td>Antenna Unit</td>
</tr>
<tr>
<td>BUF</td>
<td>Buffer</td>
</tr>
<tr>
<td>CAN</td>
<td>Control Area Network</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher Block Chaining mode</td>
</tr>
<tr>
<td>CCSDS</td>
<td>Consultative Committee for Space Data Systems</td>
</tr>
<tr>
<td>CCU</td>
<td>Channel Coding Unit</td>
</tr>
<tr>
<td>CFB</td>
<td>Cipher FeedBack mode</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
</tr>
<tr>
<td>CMAC</td>
<td>Cipher based Message Authentication Code</td>
</tr>
<tr>
<td>CORDIC</td>
<td>CO-ordinate Rotational Digital Computer</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off The Shelf</td>
</tr>
<tr>
<td>CP</td>
<td>Co-Processor</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CTR</td>
<td>CountTeR mode</td>
</tr>
<tr>
<td>DERA</td>
<td>Defense Evaluation and Research Agency</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DMC</td>
<td>Disaster Monitoring Constellation</td>
</tr>
<tr>
<td>DSCU</td>
<td>Data Storage and Compression Unit</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>EO</td>
<td>Earth Observation</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>ECB</td>
<td>Electronic Code Book mode</td>
</tr>
<tr>
<td>ECC</td>
<td>Elliptic Curve Cryptography</td>
</tr>
<tr>
<td>EDAC</td>
<td>Error Detection And Correction</td>
</tr>
<tr>
<td>EO</td>
<td>Earth Observation</td>
</tr>
<tr>
<td>EPS</td>
<td>EUMETSAT’s Polar System</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>EUMETSAT</td>
<td>European Organisation for the Exploitation of</td>
</tr>
<tr>
<td></td>
<td>Meteorological Satellites</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standard</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FPU</td>
<td>Floating Point Unit</td>
</tr>
<tr>
<td>F/F</td>
<td>Flip-flop</td>
</tr>
<tr>
<td>GAO</td>
<td>General Accounting Office</td>
</tr>
<tr>
<td>GEO</td>
<td>Geo-stationary Earth orbit</td>
</tr>
<tr>
<td>GRM</td>
<td>General Routing Matrix</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>HDLC</td>
<td>high-level Data link Control</td>
</tr>
<tr>
<td>HMAC</td>
<td>Hash Message Authentication Code</td>
</tr>
<tr>
<td>HRPT</td>
<td>High Rate Picture Transmission</td>
</tr>
<tr>
<td>ICV</td>
<td>Integrity Check Value</td>
</tr>
<tr>
<td>IDEA</td>
<td>International Data Encryption Algorithm</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>ISE</td>
<td>Integrated Software Environment</td>
</tr>
<tr>
<td>ITAR</td>
<td>International Traffic in Arms Regulation</td>
</tr>
<tr>
<td>IV</td>
<td>Initial Vector</td>
</tr>
<tr>
<td>KARI</td>
<td>Korean Aerospace Research Institute</td>
</tr>
<tr>
<td>KHTT</td>
<td>Know-How Transfer and Training</td>
</tr>
<tr>
<td>KOMSAT</td>
<td>KOREan Multi-purpose SATellite</td>
</tr>
<tr>
<td>KMC</td>
<td>Key Management Centre</td>
</tr>
<tr>
<td>LEO</td>
<td>Lower Earth Orbit</td>
</tr>
<tr>
<td>LET</td>
<td>Linear Energy Transfer</td>
</tr>
<tr>
<td>LRPT</td>
<td>High Rate Picture Transmission</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>LU</td>
<td>Loop Unrolling</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Table</td>
</tr>
<tr>
<td>MD</td>
<td>Message Digest</td>
</tr>
<tr>
<td>MetOP</td>
<td>Meteorological Operational</td>
</tr>
<tr>
<td>M2M</td>
<td>Metal to Metal</td>
</tr>
<tr>
<td>MSC</td>
<td>Multi Spectral Camera</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NOAA</td>
<td>National Oceanic and Atmospheric Administration</td>
</tr>
<tr>
<td>NSA</td>
<td>National Security Agency</td>
</tr>
<tr>
<td>OBC</td>
<td>On-Board Computer</td>
</tr>
<tr>
<td>OBDH</td>
<td>On-Board Data Handling</td>
</tr>
<tr>
<td>OFB</td>
<td>Output Feedback mode</td>
</tr>
<tr>
<td>ONO</td>
<td>Oxide Nitride Oxide</td>
</tr>
<tr>
<td>OTP</td>
<td>One Time Programmable</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>PGP</td>
<td>Pretty Good Privacy</td>
</tr>
<tr>
<td>PNK</td>
<td>Pseudo Random Keys</td>
</tr>
<tr>
<td>PP</td>
<td>Pipelining</td>
</tr>
<tr>
<td>PSK</td>
<td>Public Satellite Keys</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest Shamir Adleman</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SAR</td>
<td>Synthetic Aperture Radar</td>
</tr>
<tr>
<td>SEE</td>
<td>Single Event Effect</td>
</tr>
<tr>
<td>SEL</td>
<td>Single Event Latchup</td>
</tr>
<tr>
<td>SEU</td>
<td>Single Event Upset</td>
</tr>
<tr>
<td>SEFI</td>
<td>Single Even Functional Interrupt</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hash Algorithms</td>
</tr>
<tr>
<td>SP</td>
<td>Sub Pipelining</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>SSC</td>
<td>Surrey Space Centre</td>
</tr>
<tr>
<td>SSDR</td>
<td>Solid State Data Recorders</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>SSR</td>
<td>Solid State Recorders</td>
</tr>
<tr>
<td>SSTL</td>
<td>Surrey Satellite Technologies Ltd</td>
</tr>
<tr>
<td>STRV</td>
<td>Space Technology Research Vehicles</td>
</tr>
<tr>
<td>TDES (3DES)</td>
<td>Triple Data Encryption Standard</td>
</tr>
<tr>
<td>TDR</td>
<td>Triple Device Redundancy</td>
</tr>
<tr>
<td>TID</td>
<td>Total Ionizing Dose</td>
</tr>
<tr>
<td>TMR</td>
<td>Triple Modular Redundancy</td>
</tr>
<tr>
<td>VCDU</td>
<td>Virtual Channel Data Unit</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
<tr>
<td>XTMR</td>
<td>Xilinx Triple Modular Redundancy</td>
</tr>
</tbody>
</table>
List of Symbols

⊕ Bitwise Exclusive OR
⊗ Multiplication in Galois Filed GF(2^8)
\text{Pri}_A Private key of 'A'
\text{Pub}_A Public key of 'A'
S-Box \textit{SubBytes} LUT
T-Box2 Galois filed multiplication of S-Box LUT elements by 2
T-Box3 Galois filed multiplication of S-Box LUT elements by 3
S_{RD} S-Box LUT
p_{RD} Parity bits of S-Box
p_{2RD} Parity bits of T-Box2
p_{3RD} Parity bits of T-Box3
h_{RD} Hamming code bits of S-Box
h_{2RD} Hamming code bits of T-Box2
h_{3RD} Hamming code bits of T-Box3
m(x) Irreducible polynomial used in AES algorithm
\Sigma Summation
\{a\} Representation of byte 'a' in hex
* Representation of a fault
\mu m Micro meter
Hz Hertz
MHz Mega Hertz
kbps Kilo bits per second
Mbps Mega bits per second
Gbps Giga bits per second
kg kilo gram
W Watts
mW Milli Watts
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>Joules</td>
</tr>
<tr>
<td>mJ</td>
<td>Milli Joules</td>
</tr>
<tr>
<td>f</td>
<td>Frequency of operation</td>
</tr>
<tr>
<td>$f_{\text{max}}$</td>
<td>Maximum frequency of operation</td>
</tr>
<tr>
<td>ms</td>
<td>Milli second</td>
</tr>
<tr>
<td>rad</td>
<td>Radiation absorbed dose</td>
</tr>
</tbody>
</table>
1 Introduction

1.0 Motivation

The history of cryptography stretches from the times of ancient Egypt to today and its importance is increasing day by day [1]. In recent years, with the explosive advancement of computers and the Internet, the dependence of both organizations and individuals on the security of the information stored and communicated using these systems has increased [2,3]. Security in military satellites is mandatory, and classified security products are used to protect the transmitted information. Security in commercial satellites has, however, been overlooked for various reasons such as limited computational resources, and partly due to the impression that satellites are very far and out of reach to hackers [4]. Presently, satellite manufacturers are realizing the importance of security in satellites and the demand for security services in satellites is increasing steadily [5, 6, 7].

Recent intentional hack attacks have proved that intrusion into satellite data is not an impossible task. A team at the Embry Riddle Aeronautical University managed to obtain National Oceanic and Atmospheric Administration (NOAA) satellite imagery with basic apparatus built as part of an experimental project and by using open sources available from the Internet [8]. Similarly, researchers from a Japanese University were able to access data from National Aeronautics and Space Administration’s (NASA) Earth observation satellite LandSat as it flew over Japan [9]. Furthermore, the idea behind NASA’s concept of a Space Internet is that satellite users and scientists will directly access the satellite just like any other computer over the Internet to get the required information [10, 11, 23]. Allowing direct access to
spacecraft certainly gives flexibility, but at the cost of threats such as unauthorized access and illegal use of valuable data. In order to prevent such problems, adequate security services are absolutely necessary.

Satellites are broadly classified into large and small satellites according to their mass. Satellites weighing more than 500 kg are classified as large satellites and less than 500 kg as small satellites [12]. Small satellites are cheaper, less complex, require less maintenance and also take less amount of time to build compared to traditional large satellites. The demand for them is increasing more and more in recent years as they are affordable by large number of nations across the world. Small satellites can provide platforms for carrying out successful civilian and military missions. The targeted missions of small satellites are science, Earth Observation (EO), commercial telecommunications, military, technical demonstration and education. Remote sensing or EO satellites observe the Earth by taking images with smart imaging sensors (cameras) on-board to be used in monitoring the environment, disasters, vegetation, map marking, urban planning etc. The demand for small EO satellites is growing [13]. A network of small satellites in Low Earth Orbit (LEO) can provide an effective low-cost platform for remote sensing of various phenomena on Earth. EO satellites used for disaster monitoring and mitigation applications, usually require effective, real-time monitoring in order to be able to react quickly to mitigate the effects of such disasters. Better performance and wide range of services can be achieved by using a network or constellation of low cost EO small satellites [12,13,14].

Recent unauthorized intrusions into EO satellite data have raised the importance of using security services on board. Encryption, by far the most widely adopted security service in terrestrial networks, is used to protect data from unauthorized users [2, 3]. More and more EO satellites are equipped with on-board encryption to protect the data transmitted to the ground station [15,16,17,18]. However, the encryption algorithms used in present satellite missions are typically proprietary or outdated algorithms like the Data Encryption Standard (DES) rather than algorithms based on the latest encryption standards [11].
The Rijndael algorithm approved as the Advanced Encryption Standard (AES) by the US National Institute of Standards and Technology (NIST) is being adopted by many organizations across the world [20,21]. AES is used across a wide range of platforms ranging from smart cards to big servers because of its simplicity, flexibility, easiness of implementation and high throughput. Therefore, the AES is well suited to resource-constrained platforms like satellites [11,19].

In order to meet the requirement for high data rate processing demanded by present EO satellites, hardware implementation is considered to be the preferred choice in satellites imaging payloads [12,13]. Advantages of Field Programmable Gate Arrays (FPGAs) such as flexibility of design, shorter time-to-market, lower cost, remote configurability etc., make them very suitable for use in small satellite on-board systems [22]. The effect of various optimization techniques on throughput and device area has been thoroughly investigated in present FPGA implementations. But for satellite on-board use, in addition to throughput and area, power analysis is vital as small satellites are energy constrained [13].

In addition to the above design characteristics, fault tolerance is very important in satellite applications [26,27]. Satellites operate in a harsh radiation environment and therefore any electronic systems used on-board, such as processors, memories etc., are very susceptible to faults induced by radiation [24,25]. There is no exception for an encryption processor used on-board, which should be robust enough to faults in order to avoid transmission of corrupted data to ground. Faults must be detected and corrected on-board before sending the data to ground to avoid redundant transmission and use of erroneous data. Also, if faulty data is transmitted to the ground station, the user’s request for data re-transmission has to wait until the next satellite revisit period, with revisit times varying from a couple of hours to weeks. Most of the faults that occur in satellite on-board electronic devices are radiation induced single bit flips called single event upsets (SEU) [28]. SEUs can corrupt the data during on-board encryption and hence should be mitigated.
1.1 Objectives of the Thesis

Only very few EO satellites are equipped with on-board security services, in particular only encryption services are used to protect the data transmitted to the ground station. Security services such as authentication and data integrity, which are required for the overall protection of satellite data, are not addressed at present [11]. In order to secure the communication between the satellite and the ground station, both the uplink and the downlink need to be protected. In addition, similar to secure terrestrial architectures, all security services like authentication, integrity and encryption should be used for complete protection of the satellite communication links [4,5]. Hence the initial objective of this research is to identify the security services required to protect the EO satellite links and present with a security block diagram.

Encryption algorithms used in present satellite missions are typically proprietary algorithms or outdated algorithms like DES, rather than using the latest encryption standards [11]. The Rijndael algorithm approved as the AES by the NIST in October 2000 is being adopted by many organizations across the world. One of the main objectives of this research is to investigate the suitability of the AES, in terms of speed, area and power consumption, for satellite on-board use and present with an optimal implementation.

The space environment is significantly different from the terrestrial environment. The lack of atmospheric protection increases the incident radiation, which can produce soft and hard circuit faults [28]. When an FPGA is used in space, the effects of radiation must be considered and accounted for [25]. Hence the investigation of the robustness of AES algorithm against radiation induced faults is the primary objective of this research.

1.2 Novelty of the Research Work

The following novel contributions have been made to meet the above objectives

- This research have identified the necessary security services required to protect the satellite links and presented the security architecture for small EO satellites.
• Suitability of the AES algorithm for satellite on-board use has been investigated. Novel design space exploration of the AES has been carried out in order to identify the optimal implementation for on-board use based on FPGAs. Design parameters such as throughput, area, power and energy have been considered in the exploration process.

• Satellites operate in a harsh radiation environment and therefore the AES encryption processor used on-board should be robust enough to faults in order to avoid transmission of corrupted data to ground. Hence the other dimension of exploration is to check the robustness of the AES algorithm against radiation induced faults. In order to analyse this, a novel study of fault propagation in encrypted satellite images using the AES has been carried out.

• In order to minimise the damage caused to the data due to radiation induced SEUs and hence to avoid redundant transmission and use of erroneous data, a novel fault tolerant model for the AES algorithm based on the Hamming error detection and correction codes is presented.

• Both software (using Java) and hardware (FPGA based) implementations of the proposed model are also developed to validate and to measure the power, area & throughput overhead of the proposed model.

1.3 Structure of the Thesis

Chapter 2 of this thesis introduces various cryptographic security services such as confidentiality, authentication and integrity and security mechanisms to provide these services such as encryption, digital signatures, and integrity check values. This chapter concentrates on encryption, in particular symmetric key encryption, the most widely adopted security service to provide confidentiality. The basic principles and brief history of symmetric key algorithms are discussed and the AES, the latest symmetric key encryption algorithm is introduced. The transformations used for encryption and decryption are also discussed in greater detail. AES is a block cipher, which encrypts one block of data at a time. To encrypt more than one block modes of operation have been defined by NIST. The most popular modes ECB, CBC, OFB, CFB and CTR are discussed and compared. An elaborate survey of the AES implementations on various platforms is carried out and documented at the end of the chapter.
Chapter 3 addresses the security needs of small EO satellites. This chapter starts with a brief introduction to the small satellite platform and on-board architecture block diagram. A detailed survey of on-board security measurements used in existing and planned satellites is presented. Then this chapter discusses generic requirements of on-board security measurements for EO satellites and presents on-board security architecture for small satellites. At the end of this chapter, satellite image encryption using the AES algorithm is discussed.

Chapter 4 discusses the suitability of the AES algorithm for on-board use in terms of high processing speed, low area, power and energy consumption. It discusses a suitable platform and technology for the implementation of the AES for on-board use. This chapter addresses the effect of various AES algorithmic optimization techniques on the throughput, device area and power of SRAM-based FPGA implementations targeting the space application domain. Architectural optimizations such as pipelining and sub-pipelining are also introduced in the AES implementations and design parameters are presented and compared.

Chapter 5 describes the impact of radiation faults during on-board encryption caused by SEUs and faults during transmission caused by noise. Also the impact of transmission faults and SEUs are discussed for each of the five modes, ECB, CBC, CFB, OFB and CTR. And finally a novel fault tolerant model for the AES algorithm based on the Hamming error detection and correction codes is presented. FPGA implementation of the proposed model is presented in this chapter.

General conclusions drawn from the previous chapters, together with a summary of the original contributions are presented in Chapter 6. Future work, including possible extensions on the work presented in this thesis is described as well in the same chapter. In Appendices, some related information about the previous chapters is provided.
1.4 Publications

The results of the thesis are published in a journal paper and 7 conference papers as listed below:

**Journal Papers**


**Conference Papers**


Papers under review

Another paper titled "FPGA Implementation of the AES Algorithm for Satellite On-Board Use" is submitted to Journal of Microelectronics, Elsevier on the 11th May 2007. This paper is under review.
Chapter 2

2 Encryption: Introduction & Algorithms

2.0 Introduction

Cryptography, the science of keeping information secure, is used for many centuries to protect information from being accessed or used by unauthorized people. The history of cryptography stretches from the times of ancient Egypt to today and its importance is increasing day by day [1]. People are interested in protecting their information for different reasons. The ancient Chinese used the ideographic nature of their character-based language to hide the trade secrets of silk manufacturing. While Germans used Enigma machine during Second World War to protect their military secrets from the enemies [3]. In recent years, with the explosive advancement of computers, Internet and interconnectivity the dependence of both organizations and individuals on the information stored and communicated using these systems has increased. This, in turn, has led to a heightened awareness of the need to secure data and resources from hacking and intrusion. Many lessons were learnt with the ignorance of security measures over Internet [2, 29]. Now cryptography has become mandatory and it is considered as a basic building block for the security of any computer system or network.

Before the revolution of computers and Internet, cryptography is mostly concerned about keeping the information confidential by using secret codes [1]. But in the present information technology era, in addition to confidentiality, other cryptographic security services such as authentication, authorization, access control and integrity are quite common to provide security at different abstraction levels [2,3].

In this chapter, first the brief introduction of various cryptographic security services such as confidentiality, authentication and integrity and security mechanisms to
provide these services such as encryption, digital signatures, and integrity check values have been presented (section 2.1). The rest of the chapter concentrates on encryption, in particular symmetric key encryption, the most widely adopted security service to provide confidentiality. Section 2.2 discusses the basic principles and brief history of symmetric key algorithms and introduces the AES, the latest encryption algorithm endorsed by NIST. Section 2.3 and 2.4 discusses, in greater detail, the transformations used for encryption and decryption respectively. AES is a block cipher, which encrypts one block of data at a time. To encrypt more than one block modes of operation have been defined by NIST. Section 2.5 describes and compares the most popular modes ECB, CBC, OFB, CFB and CTR. An elaborate survey of the AES implementations on various platforms has been carried out and documented in Section 2.7. Section 2.8 concludes the chapter.

2.1 Introduction to Security Services

A number of security services, such as confidentiality, integrity, authentication, non-repudiation, access control etc. are included in today's terrestrial security architectures to provide various security measures at different abstraction levels. These security services are implemented using different security mechanisms such as encryption, hash functions, digital signatures etc [2,29,33]. A brief description of few important security services and security mechanisms is given below.

2.1.1 Confidentiality

Confidentiality, the most popular security service, is used to keep the contents of information accessible to only those authorized to have it. The security mechanism that provides confidentiality service is known as encryption. Encryption is performed on plain data to produce cipher data. The reverse process is known as decryption. An encryption algorithm or cipher is used to achieve confidentiality. A key is used during the encryption and decryption process. Encryption algorithms may be symmetric or asymmetric [30,31,32].
Chapter 2. Encryption

Symmetric key cryptography uses the same key (K) for both encryption and decryption as shown in Figure 2-1[2]. The sender encrypts the plain data with key ‘K’ and sends to the receiver through unsecured channel. The receiver decrypts the data, again with key ‘K’, into its original form. The key should be kept secret and is shared by secure channel by both the sender and the receiver.

![Block Diagram of Symmetric Key Encryption](image)

Another class of algorithms uses a key pair, one key for encryption and the other one for decryption. Either of the keys in a key pair can be used for encryption and the other for decryption. These algorithms are called asymmetric key or public key algorithms. The encryption key, also known as public key, can be made public for anyone to do the encryption but only the owner of the decryption key, also known as private key, can decrypt and read the message or vice versa. The asymmetric key encryption shown in Figure 2-2 [2], uses key pair K1 (public key) and K2 (private key). Public key K1 can be made public and can be shared by many users through unsecured channel. Given a key pair, it is computationally, or otherwise, difficult to derive one key from the other, the difficulty depending on the size of the key. Hence the key size of the public key cryptography is higher than the symmetric key cryptography. This ensures that, in practice, the private key cannot be deduced from the public key.
Chapter 2. Encryption

The asymmetric key algorithms are slower than symmetric key algorithms as they use large key sizes and complex mathematical functions for encryption and decryption. For instance RSA (Rivest, Shamir and Adleman) public key algorithm uses 1024-bit key and uses modular exponentiation and multiplication of larger prime numbers [2]. Public key algorithms are mainly used for authentication, key exchange, digital certificates and digital signatures. The symmetric key algorithms are fast as they use small key size. These are used for high-speed bulk data encryption. There are wide varieties of algorithms available for symmetric key encryption. Section 2.2 will discuss principles of symmetric key cryptography and describes few popular symmetric key algorithms.

2.1.2 Authentication

Authentication provides the ability to verify the identity of a user or entity in a system. Authentication provides the assurance that information transmitted from a claimed source actually came from that source [32,33]. This service is also known as data origin authentication and data integrity. Data authentication is usually achieved by appending an extra unit of information to the original message. This extra unit of information is called the digital signature and is shown in Figure 2-3 [2]. The digital signature identifies the origin of the data, and the receiver of the data is thus assured that the data is from the claimed source. The essential characteristic of the digital signature mechanism is that the signed data unit cannot be created by an unauthorized entity.

Figure 2-3 Adding Digital Signature to Data
Many digital signature generation mechanisms require the use of an asymmetric cryptographic algorithm where sender and receiver do not hold the same cryptographic keys (as described in section 2.1.1). Rather, a pair of public and private keys that are mathematically related to one another are used. At the origin of the data, the cryptographic algorithm generates a digital signature using the sender's private key. The signature may be generated from the data itself and is of a specific length depending on the algorithm used. Data origin authentication is achieved when the digital signature is successfully verified by the receiver using the sender's public key.

Encryption of the data itself can also provide implicit authentication when using a symmetric cryptographic algorithm. Authentication is achieved because the recipient must have and use the correct key to decipher the digital signature appended to the data. This assumes there is an assured key distribution mechanism. Also, encryption provides implicit authentication when using a public key system.

2.1.3 Integrity

An integrity service is used to ensure that unauthorized users have not manipulated the data in any way. Data integrity provides assurance that data transmitted from a source is unchanged by detecting if it has not been accidentally or maliciously modified, altered, or destroyed [32,33].

Integrity of data is achieved by appending an Integrity Check Value (ICV) to the data structure in a manner similar to the way a digital signature is appended. However, the ICV is always a function of the data itself. A Cyclic Redundancy Check (CRC) is a simple example of such a function. Stronger functions include Message Digest 5 (MD5) and the Secure Hash Algorithms (SHA-1, SHA-2). The receiver generates a corresponding check value by performing an operation (which may be cryptographic) on the data and compares the result to a received value to determine if the data has been modified in transit. Cryptographic functions such as keyed-Hash Message Authentication Code (HMAC) or Cipher-based MAC (CMAC) are used to achieve both authentication and integrity of the data simultaneously. Using HMAC, authentication code is generated using a cryptographic hash value in combination with a secret key. Hash function is used for integrity check whereas the use of secret key
Chapter 2. Encryption

provides authentication. And hence both authentication and integrity can be simultaneously verified. More details on authentication and integrity can be found at [2, 3].

Other popular security services include non-repudiation, which prevents both the sender and the receiver of a transmission from denying previous commitments or actions, and access control, which limits and controls the access to information only to authorized people. More detailed description of popular security services, mechanisms, algorithms and their features can be found in [2,11]. Security mechanisms make use of different cryptography algorithms to provide different security services, some of which are listed in Table 2-1.

Table 2-1 Security Services, Mechanisms and Algorithms

<table>
<thead>
<tr>
<th>Security Service</th>
<th>Security Mechanism</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
<td>Symmetric and Asymmetric Key Encryption Algorithms</td>
<td>DES, 3-DES, IDEA, AES, RSA, ECC</td>
</tr>
<tr>
<td>Authentication</td>
<td>MAC and Digital Signatures</td>
<td>MD5, SHA-1, KEBROS, DSA</td>
</tr>
<tr>
<td>Integrity</td>
<td>Hash Functions, Digital Signatures, Checksum Function</td>
<td>RSA, DSA, ECC</td>
</tr>
</tbody>
</table>

For secure terrestrial communication, well-defined security policies and infrastructures like X.800, Public Key Infrastructure (PKI), and Pretty Good Privacy (PGP) etc are available for the implementation of security services using the security mechanisms and algorithms listed in Table 2-1.

The rest of the chapter concentrates on symmetric key encryption, which is widely adopted to provide confidentiality.

2.2 Introduction to Symmetric Key Encryption

As discussed above, symmetric key encryption uses the same key for both encryption and decryption as shown in Figure 2-1. The symmetric key algorithms use small key size compared to public or asymmetric key encryption. Symmetric key encryption algorithms are used for bulk data encryption, as public key encryption algorithms
require a lot of computational resources. The following section gives a quick insight into the principles used in the design of these symmetric ciphers.

2.2.1 Principles of Symmetric Key Algorithms

Most of the encryption algorithms are based on the combination of following general principles [2]. They are -

- **Substitution**, in which each element in the plaintext is mapped into another element.
- **Transposition (diffusion)**, in which elements in the plaintext are rearranged by means of shifts and rotate.
- **XOR (exclusive OR)**, in which elements in the plaintext are manipulated according to the truth table of XOR.

Table 2-2 Characteristics of Popular Symmetric Key Algorithms

<table>
<thead>
<tr>
<th>Encryption Algorithms</th>
<th>Transformations/ Mathematics Involved</th>
<th>Key Length (bits)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Encryption Standard (DES)</td>
<td>1. Initial Permutation</td>
<td>56</td>
<td>DES is easily breakable because of short key length.</td>
</tr>
<tr>
<td></td>
<td>2. Expansion Permutation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. S-Box Substitution</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Final Permutation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Key Generation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Encryption Standard (AES)</td>
<td>1. AddRoundKey</td>
<td>128 (Supports 192, 256 bits)</td>
<td>AES is the latest encryption algorithm suitable for variety of platforms ranging from smart cards to big servers.</td>
</tr>
<tr>
<td></td>
<td>2. SubBytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. ShiftRows</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. MixColumns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most systems, referred to as product systems, involve multiple stages of substitutions, transpositions and XOR transforms. Table 2-2 summarises the various substitution, transposition, and XOR transformations involved in popular symmetric ciphers. Section 2.2.2 will discuss popular symmetric key algorithms in brief.
2.2.2 Brief History of Symmetric Key Algorithms

Symmetric key encryption, also referred as conventional encryption or single key encryption, was the only type of encryption in use prior to the development of asymmetric key encryption. It remains by far the most widely used of the two types of encryption schemes. Symmetric key encryption is in use from Romans period to till today [1].

DES is the most widely used symmetric key encryption algorithm since 1977. However, with the advancement in modern technology, it has now become increasingly feasible to break a DES-encrypted cipher data [2].

As a result, the Triple DES [2,3] algorithm emerged, as the name implies, encrypts a given plaintext by applying DES algorithm three times. If $E_K(I)$ and $D_K(I)$ represent the encryption and decryption of I using DES-key K respectively, then Triple-DES encryption is given by $E_K^3(D_K^2(E_K^1(I))$ where $K_1$, $K_2$, and $K_3$ are three keys. The decryption of I using Triple-DES is given by $D_K^1(E_K^2(D_K^3(I)))$.

But in 1997, NIST officials re-energized cryptography by holding a global competition and inviting cryptographers to submit their best encryption algorithms [2,19,20,21]. From those submissions, NIST selected a fast and tough-to-break Rijndael algorithm, submitted by two Belgians researchers: Dr. Vincent Rijmen Dr. Joan Daemen, as the winner and referred it as AES. The AES is expected to replace Triple-DES eventually because of its strong cryptographic features and larger key sizes. Section 2.3 will discuss the AES in greater detail.

2.2.3 Introduction to the AES Algorithm

In January 1997, the US NIST announced the start of an initiative to develop a new encryption standard: the AES. The new encryption standard was to become a Federal Information Processing Standard (FIPS), replacing the old DES and triple-DES. NIST invited the proposals for the new encryption standard from researchers, organisations across the world. In September 1997, the final request for candidate nominations for the AES was published. NIST declared that it was looking for a block cipher as secure as triple-DES, but much more efficient [19,20,21,35].
There were 15 AES candidate algorithms that were accepted for consideration for the first evaluation round. The names of the candidate algorithms are CAST-256, Crypton, DEAL, DFC, E2, Frog, HPC, LOK197, Magenta, MARS, RC6, Rijndael, SAFER+, Serpent and Twofish. All these candidate algorithms were presented at The First Advanced Encryption Standard Candidate conference, held in Ventura, California, on 20-22 August 1998. This was the official start of the first evaluation round, during which the international cryptographic community was asked to mount attacks and try different cryptanalysis on different candidates and also to evaluate the implementation cost. In March 1999, the second AES conference was held in Rome, Italy. The papers presented at the conference ranged from crypto-attacks, cipher cross-analysis, smart card related papers and candidate algorithms cost evaluation. After the second conference, NIST announced the five finalists in August 1999. The finalists were MARS, RC6, Rijndael, Serpent and Twofish.

After the announcement of five candidates NIST made another open call for contributions focused on the finalists. A third conference was held in New York City in April 2000 to discuss intellectual property issues and performance and chip area in dedicated hardware implementations. On 2nd October, 2000, NIST officially announced that Rijndael would become the AES.

The report by NIST [19] that justifies the choice of Rijndael as the Advanced Encryption Standard states the following advantages of AES:

"Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time is excellent, and key agility is good. Rijndael's very low memory requirements make it very well suited for restricted space environments, in which it also demonstrates excellent performance. Rijndael is very robust to power and timing attacks."
2.2.3.1 Differences Between Rijndael and AES

Rijndael algorithm is a block cipher that became the AES in year 2000. A block cipher is a function which maps n-bit plain data blocks to n-bit cipher data blocks; n is called the block length. The function is parameterized by a key.

The difference between Rijndael and the AES is the range of supported values for the block length and cipher key length. Rijndael is a block cipher with both a variable block length and a variable key length. The block and key length can be independently specified to any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. It would be possible to define version of Rijndael with a higher block length or key length. The AES fixes the block length to 128 bits, and supports key lengths of 128, 192 or 256 bits only. The extra block and key lengths in Rijndael were not evaluated in the AES selection process, and consequently they are not adopted in the current FIPS standard [19,20,21].

2.2.3.2 Mathematics Involved in the Design of the Rijndael Algorithm

The design of the Rijndael algorithm relies on properties of finite fields, one of the elements of modern or abstract algebra. A finite field is a field with a finite number of elements. The number of elements in the field is called the order of the field. A finite field of order $p^n$ is generally written as GF ($p^n$); GF stands for Galois Field. P is called the characteristic of the finite field [2,33]. All fields used in the description of Rijndael have a characteristic of 2 with n = 8. Thus basic operations of AES are defined over elements of the field GF ($2^8$).

Finite fields GF ($2^8$) can be represented in several ways. The specification of Rijndael has adopted the polynomial representation of bytes with coefficients over the field GF (2). A polynomial representation of byte $b(x)$ with coefficient over the field GF (2) is represented as follows:

$$b(x) = \sum_{i=0}^{7} b_i x^i = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

(2-1)
And the binary representation of this polynomial is as follows:

\[ b(x) \mapsto b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \]  \hspace{1cm} (2-2)

For example a byte C1 in hexadecimal can be represented in binary form as 1100 0001 and in polynomial form as \( x^7 + x^4 + 1 \).

The addition of two polynomial bytes in GF is defined as addition of the corresponding polynomials. In case of polynomials over GF (2) addition is just a XOR operation. The bitwise XOR addition is represented as \( \oplus \).

For byte multiplication the following irreducible polynomial is used in the description of AES.

\[ m(x) = x^8 + x^4 + x^3 + x + 1 \]  \hspace{1cm} (2-3)

Multiplication in Galois field is represented as \( \otimes \). The multiplication of two polynomials \( a(x) \) and \( b(x) \) is defined as the algebraic product of the polynomials modulo the irreducible polynomial \( m(x) \).

\[ c(x) = a(x) \otimes b(x) = a(x)b(x) \mod m(x) \]  \hspace{1cm} (2-4)

2.3 Encryption Using the AES Algorithm

AES is a symmetric key algorithm, in which both the sender and the receiver use a single key for encryption and decryption. For encryption the input is a plain data block and a key, and the output is a cipher data block. For decryption, the input is a cipher data block and a key, and the output is a plain data block. It is an iterative algorithm and each iteration is called a round. The number of rounds is denoted by \( N_r \) and depends on block length and key length. For 128-bit block length of data, the total number of rounds \( (N_r) \) is 10, 12, or 14 when the key length is 128, 192 or 256 bits, respectively. Each round in AES except the final round consists of four transformations: \textit{SubBytes}, \textit{ShiftRows}, \textit{MixColumns} and \textit{AddRoundKey}. The final round does not have the \textit{MixColumns} transformation as shown in Figure 2-4 [2].
decryption flow is simply the reverse of the encryption, and each operation is the inverse of the corresponding one in encryption [2,19,20,33].

The round transformation of AES and its steps operate on some intermediate results, called state. The state can be visualized as a rectangular matrix with four rows. The number of columns in the state is denoted by $N_b$ and is equal to the block length in bits divided by 32. For a 128-bit data block (16 bytes) the value of $N_b$ is 4, hence the state is treated as a 4 x 4 matrix and each element in the matrix represents a byte. For the sake of simplicity, in the rest of the chapter both, the data block and the key lengths are considered as 128-bit long. However all the discussions and the results hold true for 192-bit and 256-bit keys as well.

![Figure 2-4 AES Algorithm Flow Chart For Encryption](image)

**2.3.1 Encryption of Round Transformations**

For a block of 128 bits, the input is grouped into 16 bytes and arranged into a 4x4 matrix called a state matrix [2,19]. In 2-5, input state matrix $[a_{ij}]$ represents four
columns and four rows of the state matrix, where $0 \leq i < 4$, and $0 \leq j < 4$ for 128-bit data.

$$
\begin{bmatrix}
a_{00} & a_{01} & a_{02} & a_{03} \\
a_{10} & a_{11} & a_{12} & a_{13} \\
a_{20} & a_{21} & a_{22} & a_{23} \\
a_{30} & a_{31} & a_{32} & a_{33}
\end{bmatrix}
$$

(2-5)

**2.3.1.1 SubBytes Transformation**

This is the non-linear transformation of the algorithm. This transformation is carried out using the S-Box look-up table (LUT) [2,19]. AES defines a 16 X 16 matrix of byte values, called an S-Box look-up table that contains a permutation of all possible 256 8-bit values [2,19]. Each individual byte of state matrix is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the right most 4 bits are used as a column value. These row and column values serve as indexes into the S-Box to select a unique 8-bit output value.

![Figure 2-5 S-Box Look Up Table (LUT)](image-url)

In Figure 2-5, leftmost four bits of a byte are denoted by $y$ and the right most four bits by $x$. Each byte in the state matrix is substituted with another byte by looking for the entry in the $x$-row and the $y$-column of the LUT as shown in Figure 2-5 [2,19].
The input state matrix \([a_{ij}]\), is transformed into a new matrix \([b_{ij}]\) using S-Box as shown in (2-2). Each individual byte of input state matrix \(a_{ij}\), is mapped into a new byte, \(b_{ij}\), using the S-Box i.e., \(b_{ij} = S-Box(a_{ij})\).

\[
\begin{bmatrix}
    b_{00} & b_{01} & b_{02} & b_{03} \\
    b_{10} & b_{11} & b_{12} & b_{13} \\
    b_{20} & b_{21} & b_{22} & b_{23} \\
    b_{30} & b_{31} & b_{32} & b_{33}
\end{bmatrix} = \begin{bmatrix}
    S - Box(a_{00}) & S - Box(a_{01}) & S - Box(a_{02}) & S - Box(a_{03}) \\
    S - Box(a_{10}) & S - Box(a_{11}) & S - Box(a_{12}) & S - Box(a_{13}) \\
    S - Box(a_{20}) & S - Box(a_{21}) & S - Box(a_{22}) & S - Box(a_{23}) \\
    S - Box(a_{30}) & S - Box(a_{31}) & S - Box(a_{32}) & S - Box(a_{33})
\end{bmatrix}
\]  

This S-Box LUT used in \textit{SubBytes} transformation is constructed using the following steps-

\begin{enumerate}
  \item Initialize the S-Box with the byte values in ascending sequence row by row. The first row contains 00, 01, 02, 03,...,0F; the second row contains 10, 11, 13,...,1F and so on. Thus, the value of the byte at row \(x\), column \(y\) is \(xy\).
  \item Calculate the multiplicative inverse of each byte in the S-Box under Galois Field \((2^8)\) with irreducible polynomial \((x^8 + x^4 + x^3 + x + 1)\)
  \item Then carry out the affine transformation on each bit \((b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)\) of the byte \((b)\) as follows
\end{enumerate}

\[
b_i' = b_i \oplus b_{(i+4) \text{mod} 8} \oplus b_{(i+5) \text{mod} 8} \oplus b_{(i+6) \text{mod} 8} \oplus b_{(i+7) \text{mod} 8} \times c_i
\]  

where \(c_i\) is the \(i^{th}\) bit of byte \(c\) with the value \((c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0) = (01100011)\)

\(b_i'\) represents the updated bit value by the expression on the right.

\subsection{2.3.1.2 ShiftRows Transformation}

This step causes diffusion of the bits over multiple rounds [2,19]. \textit{ShiftRows} transformation cyclically shifts the rows of the state over different offsets. The row 0 in the matrix is not shifted, row 1 is shifted left by one byte, row 2 is shifted left by two bytes, and row 3 is shifted left by three bytes, giving the new state matrix.
The state matrix from the SubBytes transformation \([b_{ij}]\) is transformed into a new matrix \([c_{ij}]\) using the ShiftRows transformation as in (2-4).

### 2.3.1.3 MixColumns Transformation

This linear transformation operates on the state matrix column by column [2,19]. The matrix obtained from the last step i.e. ShiftRows, \([c_{ij}]\), is multiplied with a standard matrix to produce a new output matrix \([d_{ij}]\). The transformation can be defined by a matrix multiplication as follows.

\[
\begin{bmatrix}
  d_{00} & d_{01} & d_{02} & d_{03} \\
  d_{11} & d_{12} & d_{13} & d_{10} \\
  d_{22} & d_{23} & d_{20} & d_{21} \\
  d_{33} & d_{30} & d_{31} & d_{32}
\end{bmatrix}
= \begin{bmatrix}
  02 & 03 & 01 & 01 \\
  01 & 02 & 03 & 01 \\
  01 & 01 & 02 & 03 \\
  03 & 01 & 01 & 02
\end{bmatrix}
\begin{bmatrix}
  c_{00} & c_{01} & c_{02} & c_{03} \\
  c_{10} & c_{11} & c_{12} & c_{13} \\
  c_{20} & c_{21} & c_{22} & c_{23} \\
  c_{30} & c_{31} & c_{32} & c_{33}
\end{bmatrix}
\]

(2-9)
Chapter 2. Encryption

Each byte of a product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF \((2^8)\). The \textit{MixColumns} transformation on a single column \(j\) of state matrix can be expressed as follows.

\[
\begin{align*}
    d_{0j} &= (2 \odot c_{0j}) \oplus (3 \odot c_{1j}) \oplus c_{2j} \oplus c_{3j} \\
    d_{1j} &= c_{0j} \oplus (2 \odot c_{1j}) \oplus (3 \odot c_{2j}) \oplus c_{3j} \\
    d_{2j} &= c_{0j} \oplus c_{1j} \oplus (2 \odot c_{2j}) \oplus (3 \odot c_{3j}) \\
    d_{3j} &= (3 \odot c_{0j}) \oplus c_{1j} \oplus c_{2j} \oplus (2 \odot c_{3j})
\end{align*}
\] (2-10)

The graphical representation of the \textit{MixColumns} transformation is as shown in Figure 2-6. Here each column of the state matrix is multiplied by a predefined matrix to produce a new column in the output matrix, \([d_{ij}]\).

2.3.1.4 AddRoundKey Transformation

In this step, the matrix is XORed with the expanded round key [2,19]. The original key consists of 128 bits arranged as a 4x4 matrix of 16 bytes. Each column in the matrix is treated as a 32-bit word thus the initial key matrix with four columns or four words. This initial key is expanded to 40 more columns or words, four columns for each round, by key expansion algorithm. The key matrix is XORed with the state matrix as in (2-11). The key expansion is described in detail in the following section.

\[
\begin{bmatrix}
e_{00} & e_{01} & e_{02} & e_{03} \\
e_{10} & e_{11} & e_{12} & e_{13} \\
e_{20} & e_{21} & e_{22} & e_{23} \\
e_{30} & e_{31} & e_{32} & e_{33}
\end{bmatrix} = \begin{bmatrix}
d_{00} & d_{01} & d_{02} & d_{03} \\
d_{10} & d_{11} & d_{12} & d_{13} \\
d_{20} & d_{21} & d_{22} & d_{23} \\
d_{30} & d_{31} & d_{32} & d_{33}
\end{bmatrix} \oplus \begin{bmatrix}
k_{00} & k_{01} & k_{02} & k_{03} \\
k_{10} & k_{11} & k_{12} & k_{13} \\
k_{20} & k_{21} & k_{22} & k_{23} \\
k_{30} & k_{31} & k_{32} & k_{33}
\end{bmatrix}
\] (2-11)

These aforementioned four transformations are carried out for \(N_r-1\) rounds [32]. During the final round, only the following transformations are performed: \textit{SubBytes}, \textit{ShiftRows} and \textit{AddRoundKey}. \textit{MixColumns} is not performed in the final round. The entire encryption process is shown by a block diagram given in Figure 2-4.
2.3.2 Key Expansion

The second input to the AES algorithm is 128-bit key which is expanded internally by KeyExpansion algorithm [2,19,20,33]. In this algorithm 128-bit input key is treated as four words or four columns key array. Each word is 32-bits length. KeyExpansion algorithm expands the four word input key to 44 word key. This is sufficient to provide a key for the initial round and each of the 10 rounds of the AES cipher.

The 128-bit input key is represented as a key matrix \([k_i]_0^3, 0 \leq i < 4\), and each element in the matrix is a byte. For 128-bit key, the key matrix has 16 elements or bytes. Each column of the key matrix is treated as a 32-bit word, and hence the input key has four words namely \(w_0, w_1, w_2\) and \(w_3\) as shown in Figure 2-7 [2]. This input key is expanded into 44 word key by KeyExpansion algorithm as shown in Figure 2-8. The first group of four words, \(w_0, w_1, w_2\) and \(w_3\), provide the key for the initial round. The subsequent group of four words, \(w_4, w_5, w_6\) and \(w_7\), provide the round key for the first round, \(w_8, w_9, w_{10}\) and \(w_{11}\) for the second round and so on.

![Figure 2-7 AES Key Expansion](image)

The key expansion algorithm can be expressed by the pseudo code as follows:

```
KeyExpansion (byte key [16], word w [44])
{
    word temp
    for (i=0; i<4; i++)
        w[i] = (key[4*i], key[4*i + 1], key[4*i + 2], key[4*i + 3]);
    for (i=4; i< 44 ; i++)
    {
        temp = w[i-1];
        if ( i mod 4 = 0)
            temp = SubWord(RotWord(temp)) \oplus Rcon[i/4];
        w[i] = w[i-4] \oplus temp
    }
}
```

![Figure 2-8 Pseudo Code for Key Expansion Algorithm](image)
SubWord in the above pseudo code applies SubBytes to each of the four bytes in a word. The function RotWord rotates each byte in a word one position to the left. For example, the input word \([k_{00}, k_{10}, k_{20}, k_{30}]\) transformed to \([k_{10}, k_{20}, k_{30}, k_{00}]\).

RCon(i) is the round constant word array, whose value is \([R(i), 0, 0, 0]\). The values of RC (i) are listed as in Table 2-3. All these functions of key expansion are used shown in Figure 2-9.

<table>
<thead>
<tr>
<th>RC(1)</th>
<th>RC(2)</th>
<th>RC(3)</th>
<th>RC(4)</th>
<th>RC(5)</th>
<th>RC(6)</th>
<th>RC(7)</th>
<th>RC(8)</th>
<th>RC(9)</th>
<th>RC(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>02</td>
<td>04</td>
<td>08</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>1b</td>
<td>36</td>
</tr>
</tbody>
</table>

![Table 2-3 Round Constant Values](image)

**Figure 2-9 Key Expansion Algorithm**

### 2.4 Decryption Using the AES Algorithm

Decryption is the opposite process of encryption. The inputs to decryption block are cipher data block and a key, and the output is a plain data block \([2, 19, 20, 33]\). To decrypt cipher data, the procedure followed is exact opposite of the encryption process. The decryption flow is simply the reverse of the encryption, and each operation is the inverse of the corresponding one in encryption. An encryption round has the transformations in the sequence SubBytes, ShiftRows, MixColumns and AddRoundKey. The decryption has InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns.
2.4.1 Straight Forward Decryption

The block diagram for straight forward decryption algorithm can be obtained by inverting the encryption block diagram in Figure 2-4 [2]. Similar to the transformations used in encryption, the equivalent and inverse transformations used for decryption are $\text{InvSubBytes}$, $\text{InvShiftRows}$, $\text{InvMixColumns}$ and $\text{AddRoundKey}$ [2,19,36]. The round keys are the same, and hence the same name $\text{AddRoundKey}$, as those in encryption generated by Key Expansion, but is used in reverse order. A standard round consists of $\text{InvSubBytes}$, $\text{InvShiftRows}$, $\text{InvMixColumns}$ and $\text{AddRoundKey}$ operations while the final round consists of the same operations excluding the $\text{InvMixColumns}$ operation. The straight forward decryption block diagram is showed in Figure 2-10 [36].

![Figure 2-10 Straightforward Decryption](image)

But the sequence of transformations for encryption shown in Figure 2-4 is completely different from the sequence of transformations for decryption shown in Figure 2-10. However, it is possible to get an equivalent version of the decryption algorithm that has the same sequence of transformations as encryption with minor changes to the key schedule.
2.4.2 Equivalent Decryption

An encryption round has the transformations in the sequence $\text{SubBytes}$, $\text{ShiftRows}$, $\text{MixColumns}$ and $\text{AddRoundKey}$. The straightforward decryption has $\text{InvShiftRows}$, $\text{InvSubBytes}$, $\text{AddRoundKey}$ and $\text{InvMixColumns}$. In order to make the sequence of decryption identical to encryption two changes have to be done. The first change is to interchange the $\text{InvShiftRows}$ and $\text{InvSubBytes}$ transformations and the second is to interchange $\text{InvMixColumns}$ and $\text{AddRoundKey}$ [2,19,36].

The first change can be made without change to the structure of the decryption algorithm. Because $\text{InvShiftRows}$ simply shift bytes and has no effect on the byte values. $\text{InvSubBytes}$ operates on individual bytes, independent of their position. Therefore these two transformations can be interchanged. $\text{InvMixColumns}$ transformation is linear and hence the following holds true.

$$\text{InvMixColumns} (\text{InvSubBytes} \oplus \text{RoundKey}) = \text{InvMixColumns} (\text{InvSubBytes}) \oplus \text{InvMixColumns} (\text{RoundKey})$$

(2-12)

![Figure 2-11 Equivalent Decryption](image-url)
Chapter 2. Encryption

This property allows the exchange of $InvMixColumns$ and $AddRoundKey$ if $RoundKeys$ are modified by $InvmixColumns$ transformation before they are added up in the $AddRoundKey$ transformation. By making the above two changes the decryption structure can be made equivalent to encryption structure as shown in Figure 2-11 [36].

2.4.3 Decryption of Round Transformations

The transformations used during encryption are $SubBytes$, $ShiftRows$, $MixColumns$ and $AddRoundKey$. The structure of the AES round transformation requires that all steps be invertible so that decryption is possible. The invertible round transformations for the decryption process are $InvShiftRows$, $InvSubBytes$, $AddRoundKey$ and $InvMixColumns$, which are discussed below [2,19,36].

2.4.3.1 $InvSubBytes$ Transformation

This transformation makes use of the inverse S-Box. The inverse S-Box is constructed by applying the inverse affine transformation followed by taking the multiplicative inverse in $GF(2^8)$.

The inverse affine transformation is

$$b'_i = b_{(i+2) \mod 8} \oplus b_{(i+5) \mod 8} \oplus b_{(i+7) \mod 8} \oplus d_i$$  \hspace{1cm} (2-13)

Where $d_i$ is the $i^{\text{th}}$ bit of byte $d$ with the value $(d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0) = (00000101)$

2.4.3.2 $InvShiftRows$ Transformation

$InvShiftRows$ is the inverse transformation of $ShiftRows$. In this transformation, the bytes in the first row of the state do not change; the second, third and fourth row shift cyclically one byte, two bytes, and three bytes to the right, respectively [2,19,36].

2.4.3.3 $InvMixColumns$ Transformation

This linear transformation operates on the state column by column similar to $MixColumns$ transformations. The state matrix is multiplied with a standard matrix to
produce an output matrix. The transformation can be defined by matrix multiplication as shown below

\[
\begin{bmatrix}
    d_{00} & d_{01} & d_{02} & d_{03} \\
    d_{11} & d_{12} & d_{13} & d_{10} \\
    d_{22} & d_{23} & d_{20} & d_{21} \\
    d_{33} & d_{30} & d_{31} & d_{32}
\end{bmatrix} =
\begin{bmatrix}
    c_{00} & c_{01} & c_{02} & c_{03} \\
    c_{10} & c_{11} & c_{12} & c_{13} \\
    c_{20} & c_{21} & c_{22} & c_{23} \\
    c_{30} & c_{31} & c_{32} & c_{33}
\end{bmatrix} \otimes \begin{bmatrix}
    0E & 0B & 0D & 09 \\
    09 & 0E & 0B & 0D \\
    0D & 09 & 0E & 0B \\
    0B & 0D & 09 & 0E
\end{bmatrix}
\]

(2-14)

2.4.3.4 AddRoundKey Transformation

The round keys for decryption are the same, and hence the same name AddRoundKey, as that in encryption generated by key expansion algorithm, but is used in reverse order [2,19,36].

2.5 The AES Modes

The AES encryption algorithm accepts one data block and the key and produces the encrypted data block. The input and output data blocks are of identical size. The decryption algorithm accepts one encrypted data block and the key to produce the encrypted data block. Modes of operation have been defined to apply the AES block cipher to encrypt more than one 128-bit block of data. So, before embarking on implementation of AES, the mode of operation has to be selected [2,19,26,36,37].

The most commonly used modes with AES are: Electronic Code Book (ECB) mode, Cipher Block Chaining (CBC) mode, Output Feedback (OFB) mode, Cipher FeedBack (CFB) mode and Counter (CTR) mode. ECB and CTR are known as non-feedback modes whereas CBC, CFB and OFB are known as feedback modes. In addition, ECB and CBC are referred to as block cipher modes as they require the entire data block before the start of the encryption and OFB, CFB and CTR are referred to as stream cipher modes as they operate in a stream-like fashion.

2.5.1 ECB Mode

The ECB mode is the basic mode from which all other modes are built. As illustrated in Figure 2-12 [26], in this mode blocks of plain data are encrypted independent to
each other to form the cipher data. In Figure 2-12 P1, P2 ... Pn represent the plain data and C1, C2 ... Cn represents the cipher data. K is the key used in both encryption and decryption. The plain data blocks, the cipher data blocks and the key are of 128-bit each. The ‘E’ and ‘D’ blocks perform encryption and decryption respectively using the AES algorithm.

When the ECB mode encrypts data, the same plain data input results in the same cipher data output, so patterns in the input can be revealed to an eavesdropper. Therefore, the ECB mode is insecure for many applications [2,19,26,37].

2.5.2 CBC Mode

The CBC mode, illustrated in Figure 2-13 [26], is the mode in which the plain data block is XOR-ed with the cipher data of the previous block before it is encrypted.
The encryption of each block depends on all the previous blocks. The first block is XOR-ed with an initial vector (IV), which is a random number. As data is processed sequentially in the CBC mode, parallel processing is not possible. So it is not suitable for high-speed applications. CBC requires a complete block of data to start encryption so it is referred to as block cipher. For this reason it may not be suitable for real-time applications [2,26,37].

2.5.3 OFB Mode

In the OFB mode, as illustrated in Figure 2-14 [26], the output of the encryption is fed back into the input to generate a keystream, which is then XOR-ed with the plain data to generate the cipher data. Using this mode, pre-processing of the keystream is possible as it is not dependent on the incoming plain data [2,26,37]. The data block need not be 128-bit long but it can be of any length less than 128-bit. In such a case, OFB is implemented using a shift register as shown in Figure 2-13. An initial vector and the encrypted output data are divided into equal length sub-blocks of n bits (n is less than 128-bits). For example if n = 32 then IV and the subsequent 128-bit blocks are treated as consisting of four sub-blocks each.

First the initial vector is encrypted and the output is XOR-ed with n bits of plain data to form cipher data. Then IV is shifted n bits to the left and the n least significant bits of it are replaced by the n least significant bits of the encrypted IV to form the next data block. This encryption scheme is repeated until the end of the message is
reached. Hence OFB is referred to as n-bit stream cipher since it does not require the whole 128-bit block for encryption as in the ECB, CBC block ciphers.

### 2.5.4 CFB Mode

The CFB mode is another stream cipher mode, in which the block encryption output is XOR-ed as a keystream with the plain data, but the feedback term to the encryption algorithm is the cipher data itself (Figure 2-15). The n-bit stream cipher based on CFB can be implemented using a shift register as shown in Figure 2-14 [26], where n is less than 128 bits [2,37].

![Figure 2-15 Block Diagram of the CFB Mode](image)

### 2.5.5 CTR Mode

Figure 2-16 [26] shows a new mode, the CTR, which came into effect after the AES has been made as a standard. In CTR mode, a counter is encrypted to generate a keystream, which is then XOR-ed with the plain data to generate the cipher data [2,26,37]. A property of CTR mode, which is different from the CBC, CFB and OFB modes, is that there is no feedback or chaining; therefore one can perform several encryptions in parallel. This is a significant advantage in high-performance applications.

![Figure 2-16 Block Diagram of the CTR Mode](image)
However the disadvantage with the CTR mode is that the successive blocks usually have small difference in their values as only few numbers of bits are different between two consecutive counter values. This would lead to differential cryptanalysis because an attacker can obtain many plain data pairs with a known small plain data difference. In addition, in CTR mode it is crucial that a counter value should not be reused as it will weaken the security of the message [3].

2.5.6 Discussion

Table 2-4 [26] compares and summarizes the characteristics of all the AES modes. As discussed above, ECB and CTR modes do not use feedback from one block to another and hence they are known as non-feedback modes. CBC, OFB and CFB are known as feedback modes as they use feedback. ECB mode is not suitable to many applications as it reveals the patterns in the encrypted output. Using CBC, OFB and CFB modes patterns are not observable in the output because of the feedback [26,37]. In non-feedback mode like CTR also patterns are not observable because of the randomness of the keystream produced by the counter. Feedback modes, such as CBC, OFB and CFB, or a non-feedback mode, such as CTR, should be used in order to prevent data patterns in the cipher data. ECB and CBC are the block ciphers as they need complete block before starting of encryption whereas OFB, CFB and CTR are known as stream ciphers as they don’t need the complete block to start encryption. In OFB and CTR modes, keystream can be generated before hand i.e. even before the input data is available. Hence these modes are suitable in applications where high-speed real-time encryption is needed.

<table>
<thead>
<tr>
<th>Feature</th>
<th>ECB</th>
<th>CBC</th>
<th>OFB</th>
<th>CFB</th>
<th>CTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern Observable?</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Block/Stream Cipher?</td>
<td>Block</td>
<td>Block</td>
<td>Stream</td>
<td>Stream</td>
<td>Stream</td>
</tr>
<tr>
<td>Feedback Mode?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Pre-processing possible?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
2.6 Implementation Approaches to AES

Several software and hardware implementations of AES have recently been proposed. The software implementations have targeted various platforms with the goal of reducing the number of CPU clock cycles required to encrypt a data block. Several hardware implementations are also available targeting Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) with the aim of reducing gate count, power and to achieve high throughput [19,36]. These design goals can be achieved by applying various optimization design techniques as detailed below.

Optimization techniques for hardware implementations are broadly divided into two categories - architectural and algorithmic optimization techniques [36].

2.6.1 Architectural Optimization Techniques

A block diagram of the AES algorithm is shown in Figure 2-4. The corresponding processing architecture is known as basic or iterative architecture, which could be simplified as illustrated in Figure 2-17 [36]. Architectural optimization techniques include design practices such as pipelining (PP), sub-pipelining (SP) and loop unrolling (LU). These optimizations are described below.

2.6.1.1 Pipelining

Pipelining can increase the speed of encryption/decryption by processing multiple blocks of data simultaneously. It is realised by inserting rows of registers among combinational logic blocks. Parts of logic between two consecutive registers form a pipeline stage as shown in Figure 2-18 [36]. Each pipeline stage is one round unit of AES and is referred as outer round pipelining. Figure 2-17[36] shows the original iterative architecture of AES to which pipelining is applied as shown in Figure 2-18.
2.6.1.2 Sub-pipelining

Similar to pipelining, sub-pipelining also inserts registers among combinational logic blocks, but in this case, registers are inserted both between and inside each round as shown in Figure 2-19 [36]. Sub-pipelining is used to further increase the speed of encryption. It is also referred to as inner round pipelining.
2.6.1.3 Loop Unrolling

Loop unrolling or unfolded architectures can process only one block of data at a time, but multiple rounds are performed in each clock cycle. Figure 2-20[36] shows the fully unrolled block diagram of the AES. Detailed studies of these architectures can be found in [19, 48].

![Figure 2-20 A Loop Unrolling AES Architecture](image)

In addition to the above optimization techniques the size of the architecture (8-bit, 32-bit, 128-bit) also affects the performance in particular chip area of the implementation. For instance, a 32-bit data path architecture needs four S-Boxes to compute the SubBytes function of a 32-bit word, while a 128-bit architecture requires 16 S-Boxes. S-Boxes are the most area consuming parts of an AES hardware implementation. The number of S-Boxes determines the overall size of an AES hardware module. Thus, using an efficient approach for implementing the S-Boxes is crucial for an AES hardware design.

Pipelining and sub-pipelining can increase the throughput for non-feed modes like ECB and CTR. In feedback modes like CBC, CFB, OFB, the encryption/decryption of the next block cannot start until the current block is finished. In this case, pipelining does not lead to any speedup, because only one stage is processing one block of data in each cycle, while the other stages are idle. Therefore pipelined architectures are not suitable for feedback applications. So the optimisation option for feedback modes is loop unrolling but it comes with a very high hardware overhead [36].

The speed and area trade-offs of the AES algorithm are dependent not only on the overall architecture of the encryption/decryption block, but also on the implementation of each round unit. A variety of methods have been brought up to implement an individual round unit. They are discussed in the following section.
2.6.2 Algorithmic Optimization Techniques

Algorithm level optimization techniques target the method of implementation of each individual AES transformation. As discussed in Section 2.3, AES has four transformations in each round called SubBytes, MixColumns, ShiftRows and AddRoundKey. No optimization is to be performed on ShiftRows and AddRoundKey transformations, since no logic gates are needed for the former transformation and only one step of XOR operation is needed for the latter. However different methods can be used to implement the SubBytes and MixColumns transformations. Both transformations can be implemented using either look-up table (LUT) or combinational logic (Non-LUT) approaches. Algorithmic optimization techniques can be applied to both feedback and non-feedback modes, irrespective of presence of feedback.

As discussed in Section 2.4, the SubBytes transformation can be implemented using the pre-calculated LUT called S-Box. Multiplicative inverse of each possible combination of a byte element followed by affine transform is pre-calculated and stored in the form of a LUT of 256 elements. Alternatively, the SubBytes transformation can be calculated on the fly by calculating the multiplicative inverse of each byte in the state matrix. Various approaches such as extended Euclid, powers of primitive elements, Itoh and Tsuji's algorithm, composite field mathematics etc. are available for the calculation of multiplicative inverse in GF (2^8). In particular, composite field inversions were found to be efficient over GF (2^8), and were used to create compact AES implementations. Using composite field arithmetic operations, elements in GF (2^8) are mapped to isomorphic-field GF ((2^4)^2). GF (2^4) operations are used to calculate the multiplicative inverse in GF (2^8) [19,38].

Similarly, there are two ways for implementing the MixColumns transformation. It can be implemented either using the LUT or non-LUT approaches. As discussed in Section 2.4, MixColumns transforms every column in the state matrix by multiplying it with a predefined polynomial [2 3 1 1]. Following the LUT approach the MixColumns transformation can be implemented using the pre-calculated LUTs [19,36]. S-Box is multiplied by 2 and 3 in GF (2^8) and is stored in tables referred to as
T-Box2 and T-Box3 respectively. Alternatively, Galois field multiplication by 2 and 3 is carried out on the fly.

The implementation of the SubBytes and MixColumns transformations using LUT and non-LUT approaches gives rise to several implementation options. Four possible implementations using different approaches are defined in Table 2-5.

<table>
<thead>
<tr>
<th>Transformation Approach</th>
<th>SubBytes</th>
<th>MixColumns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option 1</td>
<td>LUT (SBox)</td>
<td>Non-LUT (Multiplication in Galois field)</td>
</tr>
<tr>
<td>Option 2</td>
<td>Non-LUT (Composite field arithmetic)</td>
<td>Non-LUT (Multiplication in Galois field)</td>
</tr>
<tr>
<td>Option 3</td>
<td>LUT (S-Box, T-Box 2, T-Box 3)</td>
<td>LUT (S-Box, T-Box2, T-Box3)</td>
</tr>
<tr>
<td>Option 4</td>
<td>Non-LUT (Composite field arithmetic)</td>
<td>LUT (S-Box, T-Box2, T-Box3)</td>
</tr>
</tbody>
</table>

Option 1 in Table 2-5, which is a combination of LUT and combinational logic, is the most widely used hardwire implementation approach to the AES algorithm. Option 2 is implemented using combinational logic entirely. Option 3 is purely a look-up table approach, where both the SubBytes and MixColumns LUTs are merged together and implemented using the S-Box and T-Boxes.

Option 4 in Table 2-5 is not a viable implementation option. To implement MixColumns using LUT approach one needs to have all the three look up tables - SBox, TBox 2 and TBox 3 as in Option 3. Also additional combinational logic is needed for composite field implementation of SubBytes. Hence this implementation option does not offer any novel aspects but presents an additional overhead in terms of power and area as SubBytes is implemented using both SBox LUT and combinational logic.
2.6.3 Implementation of Key Expansion

In a hardware implementation of AES, the key expansion process can be accomplished in one of two ways to generate the round keys. *Roundkeys* can either be generated beforehand and stored in memory or be generated on the fly. The AES key expansion algorithm was designed to be usable on the fly, such that the *Roundkeys* can be expanded iteratively in real-time as and when they are required by the encryption algorithm. This is especially useful if the AES keys need to change on a regular basis. The only penalty here is that additional *Roundkey* expansion hardware is required. This approach is the most commonly used, as the solution is completely implemented in hardware, with no external support required [19].

If AES keys do not get changed too often, then *Roundkeys* may be expanded off-line and stored in memory for subsequent use. This can save a significant number of gates and reduce the total power consumption, and is especially appropriate in low resource implementations in FPGA, where suitable *Roundkeys* buffer RAM is readily available at low cost.

2.7 Literature Survey of AES Implementations

Table 2-6 summarises recent software implementations of the AES algorithm on various general-purpose processors. As it can be seen from Table 2-6, AES has been coded using different programming languages and executed on various platforms including personal computers (Pentium CPU) and embedded computing systems, represented by the soft microprocessor cores LEON2 and Xilinx’s Microblaze and DSP processors (Texas Instruments). JAVA on SPARC is delivering the throughput of 450 bps [39], lower when compared to the C implementations. Implementations using the C programming language have significantly higher throughputs ranging from 141 Kbps to 112.3 Mbps [39,40].
In addition to software implementations, various hardware implementations targeting FPGAs and ASICs are described in the literature. It has been found that hardware implementations of AES are targeted at different design goals such as high throughput, low power and compact design etc., which is achieved by applying various algorithmic and architectural optimization design techniques as detailed below.

2.7.1 Review of AES Hardware Implementations

Better and more efficient hardware implementation of AES has been the focus of numerous research projects aimed at achieving high throughput, low power and minimal device utilization. Many hardware implementations of the AES targeting ASICs and FPGAs are described in the literature. Table 2-7 shows a summary of the ASIC implementations of the AES whereas Table 2-8 shows a summary of the FPGA implementations. Each table is classified into two categories. The first classification, shown on the left hand side of the tables, divides the tables in two parts. The first part details encryption module implementations only whereas the second part relates to combined encryption and decryption module implementations. The second classification, shown on the right hand side of the tables, is done according to the algorithmic optimization options namely Options 1, 2 & 3. Almost all the FPGA implementations have targeted Xilinx FPGAs and most of the ASIC implementations have employed CMOS technology.
Table 2-7 ASIC Implementations of the AES

<table>
<thead>
<tr>
<th>Author &amp; Publication Year</th>
<th>Technology</th>
<th>Gate Count Or Area</th>
<th>Throughput Mbps</th>
<th>Throughput/gatecount (Mbps/Kgate)</th>
<th>Power mW</th>
<th>Archi. Opt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Henry 2002 [43]</td>
<td>CMOS 0.18 µm</td>
<td>613 K</td>
<td>2,290</td>
<td>3.7</td>
<td>54</td>
<td>LU</td>
</tr>
<tr>
<td>Papaefstathiou 2004 [44]</td>
<td>CMOS 0.18 µm</td>
<td>349 Kµm²</td>
<td>2,310</td>
<td>-</td>
<td>79.24</td>
<td>Iterative</td>
</tr>
<tr>
<td>Hodjat 2005 [45]</td>
<td>CMOS 0.18 µm</td>
<td>484 Kµm²</td>
<td>2,960</td>
<td>-</td>
<td>112.48</td>
<td>Iterative</td>
</tr>
<tr>
<td>Hodjat 2006 [46]</td>
<td>CMOS 0.18 µm</td>
<td>0.79mm²</td>
<td>3,840</td>
<td>-</td>
<td>54</td>
<td>Iterative</td>
</tr>
<tr>
<td>N. Kim 2003 [47]</td>
<td>TSMC 0.18 µm</td>
<td>28.62 K</td>
<td>2,300</td>
<td>80.36</td>
<td>314</td>
<td>Iterative</td>
</tr>
<tr>
<td>Su 2003 [48]</td>
<td>CMOS 0.35 µm</td>
<td>58.43 K</td>
<td>2,000</td>
<td>34.22</td>
<td>N.A.</td>
<td>PP-4</td>
</tr>
<tr>
<td>Mangard 2003 [49]</td>
<td>CMOS 0.6 µm</td>
<td>15 K</td>
<td>241</td>
<td>16.06</td>
<td>N.A.</td>
<td>SP</td>
</tr>
<tr>
<td>Lai 2004 [50]</td>
<td>CMOS 0.25 µm</td>
<td>80 K</td>
<td>1,454</td>
<td>18.17</td>
<td>N.A.</td>
<td>SP-5-10</td>
</tr>
<tr>
<td>Mukhopadhyay 2005 [51]</td>
<td>CMOS 0.18 µm</td>
<td>252 K</td>
<td>8,000</td>
<td>31.7</td>
<td>300</td>
<td>SP-2-10</td>
</tr>
<tr>
<td>Feldhofer 2005 [52]</td>
<td>CMOS 0.35 µm</td>
<td>3.4 K</td>
<td>9.9</td>
<td>2.91</td>
<td>0.0045</td>
<td>Iterative</td>
</tr>
</tbody>
</table>
Table 2-8 FPGA Implementations of the AES

<table>
<thead>
<tr>
<th>Author &amp; Publication Year</th>
<th>FPGA Used</th>
<th>FPGA Utilization</th>
<th>Throughput (Mbps)</th>
<th>TPS (Mbps/slice)</th>
<th>Algor. Opt.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Slices (% of total slices) BRAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elbirt 2001 [53]</td>
<td>Xilinx Virtex (XCV1000-4)</td>
<td>3,528 (28.7%) 0 294.2 0.083</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5,302 (43.1%) 0 300.1 0.057</td>
<td>LU-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,286 (83.7%) 0 237.4 0.023</td>
<td>PP-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5,281 (43%) 0 545.9 0.103</td>
<td>PP-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,533 (85.7%) 0 1,156.8 0.111</td>
<td>SP-1-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3,020 (24.9%) 0 491.9 0.161</td>
<td>SP-2-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4,871 (39.6%) 0 949.1 0.195</td>
<td>SP-5-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10,992 (89.5%) 0 1,937.9 0.176</td>
<td>SP-5-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saqib 2003 [54]</td>
<td>Xilinx Virtex (XCV812)</td>
<td>2,744 (58.2%) 0 258.5 0.09</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chodowiec 2003 [55]</td>
<td>Xilinx Spartan (XC2S30-6)</td>
<td>2,163 (45.4%) 100 2,868 1.29</td>
<td>PP-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standaert 2003 [56]</td>
<td>Xilinx VirtexE (XCV3200E)</td>
<td>542 (1.67%) 10 1,450 2.7</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,784 (6.6%) 100 11,776 4.23</td>
<td>PP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zambreno 2004 [57]</td>
<td>Xilinx Virtex 2 (XC2V4000)</td>
<td>387 (1.7%) 10 1,410 3.64</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,532 (6.6%) 50 4,640 3.03</td>
<td>LU-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kotturi 2005 [58]</td>
<td>Xilinx Virtex 2 Pro (XC2VP70-7)</td>
<td>5,408 (16.3%) 200 29,770 5.5</td>
<td>SP-3-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chitu 2002 [59]</td>
<td>Xilinx Virtex 2 (XC2V1000-4)</td>
<td>4,325 (84.4%) 38 739 0.17</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>McLoone 2003 [60]</td>
<td>Xilinx VirtexE (XCV800E)</td>
<td>4,681 (67.7%) 20 310 0.067</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rodriguez 2003 [61]</td>
<td>Xilinx VirtexE (XCV2000E)</td>
<td>5,677 (29.8%) 80 4,121 0.72</td>
<td>PP-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J. Wang 2003 [62]</td>
<td>Xilinx VirtexE (XCV812E)</td>
<td>3,046 (32.4%) 280 1,952 0.64</td>
<td>P-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rouvroy 2004 [63]</td>
<td>Xilinx Spartan3 (XCS550-4)</td>
<td>163 (37.7%) 3 208 1.26</td>
<td>Iterative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HodjaT 2004 [64]</td>
<td>Xilinx Virtex 2 Pro (XC2VP20-7)</td>
<td>9,446 (101.7%) 0 21,640 2.3</td>
<td>SP-7-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhang 2004 [38]</td>
<td>Xilinx Virtex (XCV1000-6)</td>
<td>11,014 (89.6%) 0 16,032 1.456</td>
<td>SP-7-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good 2006 [65]</td>
<td>Xilinx Spartan 2 (XC2S15)</td>
<td>9,496 (89.9%) 0 9,248 0.976</td>
<td>SP-3-10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LUT → Look up table
Non-LUT → non look up table (combinational)
PP-X → Pipelining with X registers in between the rounds
SP-Y-Z → Sub-pipelining with Y registers within the round and Z registers in between the rounds
LU-A → Loop unrolled for A rounds
TPS → Throughput per slice
SB → SubBytes
MC → MixColumns
In order to observe the trends in both the encryption and the encryption & decryption implementations, the entries in Tables 2-7 & 2-8 are arranged in chronological order according to the year of the implementation, starting from the first published work to the latest. Clearly the trend is to achieve high throughput with minimum power and device utilization. The ASIC implementation in [52] uses just 3.4 K gates (CMOS 0.35 μm process) with ultra low power of 4.5 μW. Similarly, the FPGA implementation in [65] uses just 122 slices of Spartan 2 FPGA, one of the smallest of Xilinx family FPGAs. Both these implementations use an iterative AES architecture and employ a complete combinational approach, as in Option2 (Table 2-5). However the throughputs achieved using these implementations are very low. Hence these AES implementations are suitable for applications where power and device area are very constrained but high throughput is not a concern.

It can also be observed from Tables 2-7 & 2-8 that the majority of the ASIC & FPGA implementations have adopted the Option1 approach where SubBytes is based on S-Box LUT whereas MixColumns uses combinational logic.

Another observation from Tables 2-7 & 2-8 is that most of the combined implementations of the encryption & decryption modules are carried out using Option2. The reason for this is that SubBytes is implemented by calculating the multiplicative inversion GF (2^8) of each byte in the state matrix followed by an affine transformation. The InvSubBytes transformation required for the decryption is calculated by applying the inverse affine transformation followed by a multiplicative inverse of the state byte. Both SubBytes and InvSubBytes use the multiplicative inverse and hence there is much reduction in the hardware resources required [19,38]. By just using simple multiplexers to select between encryption or decryption both SubBytes and InvSubBytes can be implemented as shown in Figure 2-21 [36]. Few FPGA implementations [62,63] employ the Option3 approach where both the SubBytes and MixColumns are implemented using LUTs.
Architectural optimization techniques resulting in basic iterative, loop unrolling, pipelining and sub-pipelining architectures have been adopted across the whole spectrum of AES implementations. It can be observed that an iterative architecture is adopted where moderate throughput of the order of few hundred Mbps is required, whereas LU and PP is used where high throughput in the order of few hundred Mbps to few Gbps is required. SP is used in applications where very high throughput is required in the order of tens of Gbps. It can also be observed that all the SP implementations have followed the Option2 algorithmic optimization, where pure combinational logic is used throughout the implementation. Using the Option2 approach, it is possible to insert registers inside the round as shown in Figure 2-19 to form sub-pipelining achieving ultra high throughputs. Among the FPGA implementations, a throughput as high as 30 Gbps on a Virtex 2 Pro FPGA with 5.5 Mbits/s/slice is achieved [58]. ASIC implementations [46] achieved the highest throughput of 70 Gbps using Option2 and the SP technique.

2.8 Conclusions

In this chapter security services used in modern cryptography such as confidentiality, authentication, integrity, access control, and authorization have been discussed. Various security techniques, mechanisms and algorithms to serve the security services have been also discussed.

The main focus of this chapter is on encryption, by far the most widely adopted security service for confidentiality. All the encryption algorithms can be classified into two categories namely symmetric and asymmetric key encryption algorithms. Symmetric key algorithms are used in high-speed bulk data encryption as they use...
smaller key sizes compared to public key encryption algorithms. A brief history of symmetric key algorithms has been summarized.

The latest encryption standard called the Advanced Encryption Standard is introduced. The transformations used in the AES algorithm are discussed in greater detail. The AES key expansion and decryption algorithms are also discussed. Modes of operation of the AES such as ECB, CBC, CFB, OFB and CTR are discussed in detail and their characteristics are summarized too.

AES is widely adopted to protect terrestrial systems and communications and is implemented on a wide variety of platforms. Various implementations of AES using different optimizations are investigated in detail. Results of a comprehensive survey of AES software and hardware (ASIC and FPGA) implementations on various platforms are presented and categorized according to the optimization techniques, algorithmic or architectural, employed in their implementation. The next chapter discusses the use of the AES on board satellites to encrypt the valuable data transmitted to ground.
Chapter 3

3 Satellite On-Board Encryption

3.0 Introduction

Even though there are many encryption algorithms and products available, the use of encryption technology in spacecrafts lags well behind the terrestrial systems. This is partly due to limited computational resources on-board, and partly due to the impression that satellites are very far and out of reach to hackers. But this is no longer true. Satellite manufacturers and users are realizing the importance of satellite communication security, especially after the cases where it has been proved that intrusion into satellite data is not impossible task [8, 9, 10, 11, 23,66].

In this chapter we address security needs of small EO satellites. Before going into the details of security services required in satellites, Section 3.1 briefly introduces the small satellite platform and on-board architecture block diagram. Section 3.2 gives the detailed description of on-board security measurements used in the existing and planned satellites. Section 3.3 discusses generic requirements of on-board security measurements for EO satellites and presents on-board security architecture for small satellites. Satellite image encryption using the AES algorithm is discussed in Section 3.4 and Section 3.5 concludes the chapter.

3.1 Overview to Small Satellites

The trend in satellite industry has always been towards the design of large-sized, more capable, more sophisticated and more expensive missions. But, during the last two decades, with the advancements in microelectronics and the use of Commercial-Off-The-Shelf (COTS) technology in the design of spacecrafts has led to the emergence of the so-called small satellite missions [12,14]. Small satellites are cheaper, less complex, require less maintenance and also take less amount of time to build compared to traditional large satellites. The demand for them is increasing more and more in recent years as they are affordable by large number of nations across the
world. The spirit of the current small satellite world is encompassed by the slogan "Faster, Better, Smaller and Cheaper".

Traditionally large satellites have been built by governments or large organizations, which had sufficient funding to build and maintain the satellite [66]. These satellites were designed and built without severe restrictions on mass and power of the satellite. For example, the communication satellite Intelsat 6 was built for 10 to 14 year operation with 6 m x 4 m x 12 m meters dimension and a mass of 4600 kg producing 2600 W power by solar panels [68]. On the other hand a small satellite of today is built with very limited funding and with very strict constraints on power and mass. A typical small satellite has a mass of 50 kg, accommodating a space of 0.6 m x 0.6 m x 0.6 m producing only 30 W of solar panel power. However, small satellite industry claims that 95 % of performance of large satellites can be achieved with small satellites at 5 % of the cost or 70 % performance at 1 % of the cost [14].

In general, satellites are broadly classified into large and small satellites according to their weight. Satellites weighing more than 500 kg are classified as large satellites and less than 500 kg as small satellites. Classification of satellites according to their weight in listed in Table 3-1 [12]. Approximate cost for each satellite family is also included in the table. Small satellites further fall into several categories as detailed in Table 3-1. Small satellites coming in between 10 kg and 500 kg are referred as mini satellites, and those that fall between 10 and 100 kg are considered to be microsatellites. The smaller satellites are so-called nanosatellites, ranging from 10 kg down to 1 kg, and picosatellites that weigh in at less than 1 kg. The smallest category is the femtosatellites at less than one-tenth of a kilogram [12].

It has already been demonstrated that small satellites can provide platforms for carrying out successful civilian and military missions. The targeted missions of small satellites are as follows: science, EO, commercial telecommunications, military, technical demonstration and education. The scientific aims of small satellite missions are accomplished using a range of on-board sensors such as imaging sensors, radars etc. The next section will illustrate the operation of a small satellite, in particular EO small satellite, and will present an on-board block diagram. Also the operation of
various on-board sub-systems like communications, data handling and imaging sensors will be discussed. Satellites built by Surrey Space Technologies Limited (SSTL), are used for the discussion of the small satellite platform in this thesis [12, 69].

### Table 3-1 Classification of Satellites

<table>
<thead>
<tr>
<th>Class</th>
<th>Mass (Kg)</th>
<th>Cost (£ M)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Large Satellites</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large Satellite</td>
<td>&gt; 1000</td>
<td>&gt; 100</td>
</tr>
<tr>
<td>Medium Satellite</td>
<td>500 – 1000</td>
<td>25 – 100</td>
</tr>
<tr>
<td><strong>Small Satellites</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini Satellite</td>
<td>100 – 500</td>
<td>7 – 25</td>
</tr>
<tr>
<td>Micro Satellite</td>
<td>10 – 100</td>
<td>1 – 7</td>
</tr>
<tr>
<td>Nano Satellite</td>
<td>1- 10</td>
<td>0.1 – 7</td>
</tr>
<tr>
<td>Pico Satellite</td>
<td>0.1 – 1</td>
<td>&lt; 0.1</td>
</tr>
<tr>
<td>Femto Satellite</td>
<td>0.001- 0.1</td>
<td></td>
</tr>
</tbody>
</table>

#### 3.1.1 Small Satellites Missions

SSTL, a highly innovative spinout company of the University of Surrey, was the first professional organisation to offer low-cost small satellites by employing advanced terrestrial technologies. SSTL manufactures satellite platforms spanning the spectrum from the 6.5-kg Surrey Nanosatellite Applications Platform (SNAP) to the 400-kg Galileo In-Orbit Validation Element (GIOVE –A) as shown in Figure 3-1. SSTL has developed twenty seven small satellite missions to-date and few other satellites are in development. The latest satellite built by SSTL is CFESat and was successfully launched on 9th March 2007 from Cape Canaveral Air Force Station in Florida, USA. Some of the satellites built by SSTL are shown in Figure 3-1. SSTL's development of small satellites using COTS has dramatically reduced costs to the point where they are now affordable for many more nations [12,13].
SSTL’s satellites are aimed at widely ranging applications, including remote sensing, communications and technology demonstration. Remote sensing or EO satellites observe the Earth by taking images with smart imaging sensors (cameras) on-board to be used in monitoring the environment, disasters, vegetation, map marking, urban planning etc. EO satellites used for disaster monitoring and mitigation application, are usually require real-time monitoring in order to be able to react quickly to mitigate the effects of such disasters. Better performance and wide range of services can be achieved by using a network or constellation of low cost EO small satellites. The next section will discuss about SSTL’s satellite constellation project called Disaster Monitoring Constellation (DMC).

Earth observation with a constellation, i.e. a fleet of satellites in the same or similar orbit, is proposed for improvement of the monitoring coverage and re-visits. Modern small satellite technology now makes the rapid implementation of a network/constellation of disaster monitoring and mitigation satellites both feasible and affordable [70].

The Disaster Monitoring Constellation (DMC), shown in Figure 3-2 [72], is the first Earth observation constellation of 5 low cost small satellites providing daily images for applications including global disaster monitoring. Using DMC it is now possible
to monitor any point on the globe with a minimum of a 24 hour repeat time. This capability is very useful to aid agencies when dealing with fires, floods, volcanic eruptions and earthquakes [12,13].

The DMC is an international project proposed and led by SSTL to construct a network of five affordable microsatellites. It comprises a partnership between organizations in Algeria, China, Nigeria, Turkey and the United Kingdom. Each organization has built an advanced yet low-cost Earth observation microsatellite to form the first ever constellation specifically designed and dedicated to monitoring natural and man-made disasters [72]. The first DMC microsatellite, AlSAT-1 of Algeria, was launched Nov. 28, 2002. Satellites for Algeria, Turkey and Nigeria, built under a Know-How Transfer and Training (KHTT) program at Surrey (SSTL services from concept to orbit), were launched Sept. 27, 2003. The China DMC satellite was launched in October 2005. Each satellite in the DMC constellation has approximately a mass of 50 kg, accommodating a space of 0.6 m x 0.6 m x 0.6 m producing only 30 W of solar panel power [73]. The next section will discuss the block diagram of on-board EO satellite.

![Figure 3-2 Constellation of Small Satellites (DMC)](image-url)
3.1.2 Earth Observation Small Satellite On-Board Block Diagram

In general, an EO small satellite consists of a number of subsystems like communications, command and data handling, attitude determination and orbit control, power, propulsion subsystems and imaging payloads as shown in Figure 3-3. All these subsystems are interconnected to each other through an on-board bus or network. For the sake of simplicity Figure 3-3 shows only the main connections between the subsystems in order to give an overview of the satellite on-board structure [72,73]. Generally in satellites most of the blocks are duplicated to increase reliability and therefore, the reason for having duplicate receivers, transmitters, on-board computer (OBC) etc. in Figure 3-3.

The receiver block in the communications subsystem receives the commands from ground station, known as uplink, and demodulates and decodes the uplink signals and generates telecommands. These telecommands are sent to the OBC in the on-board command and data handling (OBCDH) subsystem and from there to subsequent subsystems like attitude control and/or payloads etc depending on the nature of the command through the on-board bus. The OBCDH subsystem is responsible for the house keeping and distribution of commands to other subsystems through the bus.

The transmitter in the communications subsystem encodes and modulates the data collected from the on-board subsystems in the satellite and transmits to ground station, known as downlink. Usually, downlink consists of two parts; telemetry and payload data received from the satellite on-board control and imaging payload subsystem respectively. In Figure 3-3, low rate transmitters are used to downlink telemetry and high rate transmitters are used to transmit the bulk data stored in the mass memory of imaging payload subsystem. The following section will discuss the imaging payload unit in detail.

The attitude control and propulsion subsystems are responsible for controlling the attitude and propulsion of spacecraft respectively. Global Positioning System (GPS) and power system are the other subsystems in the on-board architecture.
3.1.2.1 Imaging Payload

A typical imaging payload system consists of an imager, storage devices and a transmission system as shown in Figure 3-3. A brief description of the stages which image has to pass from image capture to transmission has been outlined below. A satellite in the DMC constellation, AlSat-1, has been taken as an example to outline the process [13, 14].

In the first instance, payload takes the snapshot of area of interest as commanded by the ground station through uplink commands. In DMC satellites a three pair camera imager is employed for imaging as shown in Figure 3-4 [12]. Each of the three camera pairs operate at different bandwidths. The red and the green band scanning cameras provide visually representative data, whereas the near infra-red band camera can provide information about wild life in natural regions and heat exhausts of cities.
After the on-board camera has captured the image, the acquired data is transferred to the Solid State Data Recorders (SSDR). The main SSDR and redundant SSDR prototyping boards are shown in Figure 3-5 (a) and 3-5 (b) [12] respectively. An overview of the image data flow in the AISAT-1 is shown in Figure 3-6 [12]. AISAT-1 uses two main recorders with 4 Gbit (512 Mbytes) memory each and are complemented by a functionally redundant but technologically different SSDR with less storage capacity of 1 Gbit (128 Mbytes). Although the naming suggests that the SSDR is purely a storage device, it actually possesses powerful processing capabilities. For instance the main SSDRs of AISat-1 is based on MPC8260 Power PC processor and the redundant SSDR is based on SA1100 Strong Arm processor. Once the satellite is in contact with a ground receiving station, the recorders can dump the image data to one of the high-speed S-band transmitters for downlink. All nodes are controlled via Control Area Network (CAN).
The down-link speed of AlSat-1 through S-band is 40 Mbps. Figure 3-6 shows two antennas, but only one is in use. The second antenna is only used as backup if the first antenna fails. Both antennas are rarely in use at the same time, because that would use too much system power.

In recent years, to predict and mitigate disasters quickly and accurately, there is a constant demand to monitor Earth’s resources and environment changes very closely
and accurately [12]. The imagers such as hyper spectral imagers, synthetic aperture radar (SAR) imagers etc are being adopted to meet this demand. These imagers will generate terabits of data to meet the scientists demand for higher spectral and spatial resolutions [70,73]. The data rates needs to be very high to transmit the high volumes of data. It is projected that the data rates might be as high as 1Gbps by 2010 [87]. Therefore the demand for high speed on-board processing such as compression, encryption etc is also increasing.

3.2 Encryption Used in Present Earth Observation Satellites

At present, only few EO satellites are equipped with on-board encryption to protect the data transmitted to ground station. But more and more organizations are planning to have on-board encryption in their future EO missions. To name a few EO satellites that are using on-board encryption are Space Technology Research Vehicle (STRV -1d), Meteorological Operational (MetOp-A) satellite, KOrea MultiPurpose Satellite (KOMPSAT-2) etc. Many more future missions are planning to have on-board encryption including the Canadian satellite RADSAT-2, the Turkish satellite RASAT etc. Table 3-2 summarizes the use of encryption in current satellites. The following section gives a brief description of each of the existing and planned satellites with encryption on-board.

<table>
<thead>
<tr>
<th>Spacecraft Name</th>
<th>Algorithms Used</th>
<th>Implementation Platform</th>
<th>Encrypted Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Technology Research Vehicle</td>
<td>Data Encryption</td>
<td>Software (on SPARC processor)</td>
<td>S-band downlink 10Kbps</td>
</tr>
<tr>
<td>(STRV-1d) [15]</td>
<td>Standard (DES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Korea Multipurpose Satellite (KOMPSAT-2)</td>
<td>International Data Encryption Algorithm (IDEA)</td>
<td>FPGA Hardware</td>
<td>X-band downlink 160 Mbps</td>
</tr>
<tr>
<td>Meteorological Operational Satellite(MetOp-A) [16]</td>
<td>Triple Data Encryption Standard (3-DES)</td>
<td>ASIC Hardware</td>
<td>VHF 72 kbps &amp; L-band 3.5 Mbps</td>
</tr>
<tr>
<td>Turkish Satellite RASAT [75,76]</td>
<td>AES</td>
<td>ASIC</td>
<td>X-band 160 Mbps</td>
</tr>
<tr>
<td>Canadian Satellite RADSAT-2 [77]</td>
<td>DES</td>
<td>N.A.</td>
<td>Both S and X band</td>
</tr>
</tbody>
</table>

N.A. – Not Available
3.2.1 STRV-1d

STRV-1d satellite was built by the Defence Evaluation and Research Agency (DERA). It is launched from Kourou, French Guiana, on 14 November 2000 [15]. DERA built a series of satellites, namely STRV-1a, 1b, 1c & 1d, to demonstrate new technologies in orbit. STRV-1d satellite carries a suite of environmental monitors, which will give comprehensive radiation measurement, electrostatic charging effects and detect cosmic dust. The other objective of this mission was to demonstrate secure communications through on-board encryption. It has 1 kbps S-band uplink and 10 kbps S-band downlink. The 10kbps S-band downlink was encrypted using the DES. The software implementation of DES algorithm running on a SPARC processor encrypts the data.

3.2.2 KOMPSAT-2

KOPMSAT-2 is an Earth observation satellite built by the Korean Aerospace Research Institute (KARI). The 800 kg craft carries imaging systems to yield high-resolution, multispectral images of Earth's surface. The main objective of this mission was to provide high-resolution images of the Korean peninsula for the production of maps and digital elevation models, applications for use with planning, disaster and risk management. It was launched in July 2006. KARI is planning to develop KOMPSAT-3 & 5 and few more in the coming years.

The imaging unit, called Multi Spectral Camera (MSC) unit, of KOMPSAT-2 is shown in Figure 3-7 [17]. The MSC itself comprises all the elements required to form a stand-alone payload. The multispectral images captured by Optical Unit (OU) are compressed and stored in the mass memory unit called Data Storage and Compression Unit (DSCU). On demand from the ground station, the images are encrypted and formatted and encoded by Channel Coding Unit (CCU). International Data Encryption Algorithm (IDEA) is used for encryption and CCSDS & Reed-Solomon coding for formatting and encoding. The encrypted data is transmitted to ground through the Antenna Unit (AU) through X-band [17].
The Channel Coding Unit (CCU) chip is implemented on a Xilinx FPGA (XQVR 600). The chip performs the encryption with IDEA including the CCSDS-compatible link processing at 160 Mbps. The power consumption of the chip is 2.5W. Low internal core voltages of the FPGA are used for a low power implementation of the complex processing tasks.

3.2.3 MetOp-A

MetOp satellite programme is jointly established by ESA and the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT), forming the space segment of EUMETSAT's Polar System (EPS). MetOp is a new European undertaking to provide weather data services that will be used to monitor climate and improve weather forecasts. MetOp is a series of three satellites to be launched sequentially over the next 14 years and the first in the series, MetOp- A, was launched in October 2006. With an array of sophisticated imaging payloads, MetOp provides images of high resolution to monitor global weather forecasting and improve weather forecasting [16].

The science data generated by imaging payloads is multiplexed and provided on three channels going to an on-board solid state recorder (SSR), the High Rate Picture Transmission (HRPT) and Low Rate Picture Transmission (LRPT) direct broadcast systems. Science data transmission to ground is ensured by three links. One link is through X-band at 70 Mbps to dump the SSR global data and for further forwarding via terrestrial transmission links to other users. Other links of transmission are the
HRPT and LRPT providing continuous data transmission to ground in VHF (72 kbps) and L-bands (3.5 Mbps) respectively for local users.

On-board encryption of LRPT and HRPT are provided on demand using the Triple Data Encryption Standard (TDES) to prevent access by un-authorized users. The encryption keys are generated in the EPS Key Management Centre (KMC) and distributed via secure terrestrial links to registered local mission users. The on-board encryption is based on the principle shown in Figure 3-8 [16]. The encryption itself is performed by doing an exclusive OR between each clear data and a pseudo-noise pattern. As this operation is fully reversible, the data at ground are decrypted by using the same pseudo-noise pattern with an exclusive OR with data.

The pseudo-noise pattern that is the basis for the encryption is created from the secret Master Satellite Key (MSK), which is stored on board and cannot be transmitted to ground via telemetry and the Public Satellite Keys (PSK), which can be uploaded from the ground periodically to ensure sufficient secrecy and to control data access. On MetOp, there is a table of 64 different possible PSKs. These PSKs are stored in encrypted form and there is a suitable telecommand in order to select the appropriate key. One of the PSK is selected depending on the telecommand from the ground station and it is decrypted using the MSK to get the message key. Triple Data Encryption Standard (TDES) algorithm (decryption part) is used to get the message keys. The seed together with the message key is the basis for pseudo noise key or pattern (PNK). The OFB mode of TDES is used to generate the PNK. The seed is composed of header of data unit called Virtual Channel Data Unit (VCDU) and insert zone. This PNK is exclusive-ORed with the plain data to generate the cipher stream. The data handling electronics unit that includes the encryption unit is implemented using ASIC technologies.
3.2.4 RASAT

The EO satellite RASAT is being developed by Turkish research institute Tubitak-Bilten and is scheduled for launch in 2008. GEZGIN-2 is a real-time multispectral image processing subsystem developed for the RASAT microsatellite. The main functionality of this subsystem is to compress in real-time multi-spectral images received concurrently from imagers, using JPEG2000 Image Compression. The real-time encryption/decryption features are implemented on a separate board called GÖLGE, which is mounted on GEZGIN-2 as a daughter-board [75]. GÖLGE accommodates in-house designed encryption/decryption chips for public-key and private-key encryption of image/data streams [76].

GÖLGE uses two different algorithms for encryption and decryption. First one is the public-key RSA (Rivest-Shamir-Adleman) algorithm and second one is the private-key AES (Advanced Encryption Standard) Algorithm. In GÖLGE the RSA algorithm is used to encrypt and decrypt the session keys. These session keys are used by the AES crypto module to encrypt and decrypt block data, such as the images captured by the cameras on RASAT. The encryption rate of the AES algorithm is 160 Mbps at 25 MHz. It will use two ASICs for encryption/decryption using AES and RSA.
algorithms are implemented in 0.35μm standard CMOS technology. Having a data encryption and decryption throughput of 160 Mbps, GOLGE will encrypt the images captured by the cameras on RASAT in real-time.

3.2.5 RADARSAT-2

RADARSAT-2 is Canada's next-generation commercial satellite and scheduled for launch in the summer of 2007 [77]. RADARSAT-2 has been designed with significant and powerful technical advancements which include 3m high-resolution imaging, flexibility in selection of polarization, left and right-looking imaging options, superior data storage and more precise measurements of spacecraft position and attitude. The encryption of this high quality images will be carried out using the DES algorithm.

3.3 On-Board Security Architecture for Earth Observation Small Satellites

As discussed in section 3.2, only very few EO satellites are equipped with on-board security services, in particular only encryption services are used to protect the data transmitted to the ground station. Security services such as authentication and data integrity, which are required for the overall protection of satellite data, are not addressed at present. This has been highlighted in the United States General Accounting Office report (GAO-02-781) [4] on mitigating the risk of satellites being taken over by unauthorized users. In order to secure the communication between the satellite and the ground station, both the uplink and the downlink need to be protected [4,5,78]. In addition, similar to secure terrestrial architectures, all security services like authentication, integrity and encryption should be used for complete protection of the satellite communication links. Key management in satellites also plays a crucial role as in terrestrial applications. Possible key management strategies for satellite applications are discussed in [107].

Figure 3-9 [11] shows a block diagram of the proposed on-board security architecture for a small EO satellite. For the sake of simplicity Figure 3-9 includes only the main
connections between the subsystems. The following security blocks are introduced in the small satellite block diagram, Figure 3-3, to protect the communication links: an authentication and integrity check block, an encryption block and a real-time high-speed encryption block.

3.3.1 Security Services for Uplink Commands

The uplink or telecommand should be checked for integrity and authentication in order to protect the satellite from being taken over by unauthorized people. Thus, authentication and integrity block in Figure 3-9 provides protection to the satellite by ensuring that the on-board data handling subsystem receive unmodified telecommands from authorized ground station. Any telecommands that do not pass the authentication process are rejected. To achieve authentication and integrity of telecommand the security mechanisms discussed in section 2.1 can be used. For instance, if the telecommands are encrypted by the ground station using the secret key of the satellite then the successful decryption of the commands by the satellite will itself serve as the authentication of the source (ground station). In general the telecommands are encoded by the ground station [18]. Hence the successful decoding of commands on-board the satellite will serve the purpose of integrity check. The telecommands may also be encrypted by ground station depending on the level of security required [4]. In case of encrypted telecommands are sent to the satellite, they are decrypted on-board before being checked for authentication and integrity.
3.3.2 Security Services for Downlink Data

Both high rate and low rate downlinks of EO satellite should be encrypted to protect the valuable and sensitive data transmitted to the ground station. Low rate downlink consists of sensitive information such as satellite health and control information (attitude and orbit information), on-board voltage and temperature measurements and time stamps. Thus the encryption block in Figure 3-9 encrypts the low rate telemetry downlink. High rate downlink is usually the imaging payload data stored in the mass memory units. In order to protect the valuable data, the high rate downlink needs to be encrypted. Thus the real-time high-speed encryption block in Figure 3-9 [11] protects the high rate downlink. For the high rate downlink, data need to be encrypted on demand by the ground station during the contact period and therefore the encryption process should be high-speed to achieve real-time transmission. Also in spacecrafts, data will be encoded with forward error correction code before transmission in order to avoid errors during transmission. The use forward error
correction codes to the encrypted data will also serve the purpose of providing the integrity.

3.4 Encryption of Satellite Images

The encryption algorithms used in present satellite missions are typically proprietary algorithms or outdated algorithms like DES, as listed in Table 3-2, rather than using the latest encryption standards. The Rijndael algorithm approved as the AES by the NIST in October 2000 is being adopted by many organizations across the world. It is used across a wide range of platforms ranging from smart cards to big servers. AES is gradually emerging as the preferred algorithm in the aerospace industry because of its simplicity and flexibility in implementation. The CCSDS is considering recommending AES as the standard encryption algorithm for use on satellites. Recently, the Turkish satellite RASAT is also planning to use AES for on-board encryption.

3.4.1 Encryption of Satellite Images Using AES Modes

The encryption and decryption of satellite multispectral images were implemented using a software program written using the Java programming language [79]. The standard Java language features together with java image I/O Application Programmer Interface (API) (Version 1.0) have been used to encrypt and decrypt the satellite multispectral images. The image I/O API package used for reading and writing the images are java.lang, java.io and java.util. The java programs are compiled using the JDK1.4.2 compiler on windos 2000 operating system. The AES software implementation is divided into core modules and feedback modules. The core modules consist of - ShiftRows, SubBytes, MixColumns and the corresponding inverse modules for decryption. The feedback modules consist of encryption and decryption routines for ECB, CBC, CFB, OFB and CTR modes. The Sun's Java API for JPEG images [79] is used for image encoding and decoding during the encryption and decryption process. Multispectral satellite images from SSTL [12] and Internet [80] have been employed to demonstrate the satellite image encryption.

The multispectral satellite image in Figure 3-10 (a) [80] is employed as a test image to carry out encryption using the modes of AES. It is found that images encrypted with
the ECB mode reveal patterns in the input data as shown in Figure 3-10 (b), which makes the ECB mode insecure. This is due to the fact that in the ECB mode the same plain data input results in the same cipher data output. Figure 3-10 (c) shows the encrypted image using the CBC, OFB, CFB and CTR modes, where no data patterns are revealed [26].

As discussed above, ECB is not suitable for encryption of satellite images as patterns are observable in the encrypted image. Pre-processing of the keystream is one of the main advantages of the OFB and CTR modes. This OFB / CTR feature could enable high-speed and near real-time encryption of data with less processing time and computing resources in satellites. The plain data could just be XOR-ed with the pre-computed keystream to generate the encrypted data when they are to be transmitted to ground. However, the storage of the keystream requires a huge amount of memory, as the keystream length is equal to the data stream length, which makes pre-processing in OFB / CTR not so attractive for on-board use. The remaining AES modes, CBC and CFB, are also viable options for on-board use. These two modes have identical characteristics except that CBC is a block cipher mode whereas CFB is a stream cipher mode.
3.5 Conclusions

Satellites are classified according to their weight and they are broadly divided as large and small satellites. In this chapter, small satellites, in particular EO satellites have been discussed in detail. Also brief introduction to small satellite platform and on-board block diagram has been presented.

A review of on-board encryption architectures, algorithms and services used in existing satellites has been done and summarised. In this chapter the necessary security services required to protect the satellite links are identified and a security block diagram for small EO satellites is presented. In order to protect the valuable information generated by the sophisticated on-board payloads, the AES algorithm has been chosen to perform the encryption of high data rate downlink, which is of the order of hundreds of Mbps.

Also in this chapter analysis of satellite image encryption using popular modes of AES such as ECB, CBC, OFB, CFB and CTR is carried out using a purpose-built software simulator developed in JAVA. Advantages and disadvantages of each of these modes for on-board use is discussed.
Chapter 4

4 Design Space Exploration of the AES Algorithm

4.0 Introduction

A detailed survey of the AES implementations on various platforms has been carried out in Chapter 2 and in Chapter 3 we discussed the encryption of satellite images using the AES. This chapter discusses the suitability of the AES for on-board use in terms of high processing speed, small area, power and energy consumption. Various implementations of the AES using architectural and algorithmic optimizations have been carried out and evaluated. Section 4.1 discusses the suitable platform and technology for the implementation of the AES for on-board use. Section 4.2 addresses the effect of various AES optimization techniques on the throughput, device area and power of Static Random Access Memory (SRAM) based FPGA implementations targeting the space application domain.

4.1 AES Implementation: Platform and Technology

As discussed in Chapter 2, AES has been implemented on a wide variety of platforms for various applications ranging from smart cards to big servers. It has been implemented targeting both software and hardware platforms. Software implementations of the AES is carried out using programming languages such as C, C++, Java or assembly and executed on a general purpose processor [34,39,40,41,42]. In hardware implementations, a dedicated purpose-built processor using ASICs or FPGAs has been used. A comprehensive survey of various software and hardware implementations have been carried out and summarised in Chapter 2. This section addresses the suitable platform and technology for the AES implementation to use in space applications.
4.1.1 Hardware or Software platform?

In general, cryptographic algorithms are implemented using dedicated hardware to achieve a higher speed than the software implementations. The additional requirements of smart and sophisticated applications however, demand other properties of hardware implementations such as low power, energy and small device area. There are two main scenarios where hardware implementations are advantageous over software implementations.

The main advantage of hardware implementations is that they achieve high-speed processing compared to software implementations [19]. This is because software implementations run on a general purpose processor, which is shared by many other applications. Also general purpose processors are designed to execute wide variety of applications and it may not necessarily be an optimised platform to execute cryptographic algorithms which use specific set of arithmetic operations. On the other hand, a dedicated hardware processor will achieve very high speed processing as it is designed and allocated for the sole use of certain cryptographic algorithm. Typically, a cryptographic processor is used in conjunction with the general purpose processor. The cryptographic processor is then responsible for the cryptographic operations in order to relieve the main processor to attend the rest of the operations [42].

In addition to high-speed processing, present day applications demand low power and energy. In sophisticated and smart applications power constraints are highly stringent. The advantages of hardware implementations over software are low-power and high speed. Hardware implementations are faster and consume less energy and power compared to that of software. This is because, as discussed above, dedicated hardware specifically is designed to perform a given computation, and is thus can be very efficient in terms of area, power and energy consumption. Hence hardware implementations of cryptographic algorithms are preferred over software where high speed as well as low power processing is needed.
4.1.2 FPGA or ASIC?

As discussed above, there are two common methods in conventional computing for execution of algorithms. The first is to use a purpose-built hardwired technology, for example an ASIC to perform the operations in hardware. ASICs are specifically designed to perform a given computation, and are thus very efficient. The term efficient can have several meanings, for instance, the design can be high speed processing, or very small or can require only very little power. However, the circuit cannot be altered after production. This forces a redesign and remanufacturing of the chip if any part of the circuit needs to be modified.

The second method is to use a software-programmed processor or microprocessor, a far more flexible solution. Processors execute a set of instructions to perform a computation. By changing the software instructions, the functionality of the system is altered without changing the hardware. The downside of this flexibility is that the performance suffers and is far below that of an ASIC.

Reconfigurable computing intends to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Reconfigurable devices, FPGAs, contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, the so-called logic blocks, are connected using a set of routing resources that are also programmable. Synthesis and implementation tools allow the high level description of a design to be translated into the programming language for an FPGA. The reconfigurability of FPGAs offers several advantages when using them for cryptographic applications. Some of the main advantages are described below [22,88].

Algorithm Agility: This term refers to switching the cryptographic algorithms during operation of the targeted application, in our case spacecraft. One can observe that the majority of modern security protocols, such as SSL or IPsec, are algorithm independent and allow for multiple encryption algorithms. The encryption algorithm is negotiated on a per-session basis and a wide variety may be required.
Algorithm Upload: Using FPGAs it is possible to upgrade with a new encryption algorithm. A new cryptographic algorithm upload can be necessary because of the reason such as current algorithm was broken, a new standard was created and/or that the list of ciphers in an algorithm independent protocol was extended. FPGA based encryption devices can upload the new configuration code. Notice that the upgrade of ASIC based implemented algorithms is practically infeasible if the systems are not easily accessible, for instance in satellites.

Throughput rate: Although typically slower than ASIC implementations, FPGA implementations have the potential of running substantially faster than software implementations.

Cost Efficiency: There are two cost factors that have to be taken into consideration, when analyzing the cost efficiency of FPGAs: cost of development and unit prices. The costs to develop an FPGA implementation of a given algorithm are much lower than for an ASIC implementation, because one is actually able to use the given structure of the FPGA (e.g. look-up table) and one can test the reconfigured chip endless times without any further costs. This results in a shorter time-to-market period, which is nowadays an important cost factor. The unit prices are not so significant when comparing them with the development costs. However, for high-volume applications, ASIC solutions are the more cost-efficient choice.

4.1.3 Antifuse or SRAM FPGAs?

As discussed above, FPGA technology offers number of advantages including a highly compact solution, high integrity, flexibility, reduced cost, faster and cheaper prototyping and reduced time to market. The capacity and performance of FPGAs suitable for space flight applications have been steadily increasing for more than a decade. The application of FPGAs has moved from simple glue logic to complete platforms that combine several real-time system functions on a single chip. FPGAs can be split into two categories namely re-programmable and one time programmable (OTP). In an antifuse programmable device, special “anti-fuses” are included at each customisation point. These OTP FPGAs use antifuse for storing its configuration, either using oxide-nitride-oxide (ONO) or metal-to-metal (M2M) antifuse structures.
Anti-fuse FPGAs and high reliability/military products are widely used as components for satellite on-board processing.

The other category FPGAs are re-programmable FPGAs. Reprogrammable technology offers volatile SRAM or non-volatile EEPROM/Flash cells to hold the device configuration. SRAM based technology is evolving at a faster pace than OTP technology, and now features a million system gates or more on a single chip, and hence has become an attractive choice for high performance applications. The SRAM bits are connected to configuration points in the FPGA, and programming the SRAM bits configures the FPGA. Thus, these chips can be programmed and reprogrammed as easily as a standard static RAM. The methods to program or re-programme these FPGA are called run-time reconfiguration and partial reconfiguration [71,82].

Run-time (or dynamic) reconfiguration is used to swap different configurations in and out of the reconfigurable hardware as they are required during program execution. Partial run-time reconfiguration allows that part of the reconfigurable device is modified while the rest of the device is still on operation. Partial run-time reconfiguration is also called dynamic partial reconfiguration or active partial reconfiguration. Currently available partial run-time reconfigurable FPGA devices are: the Xilinx Virtex/E/2/2 Pro/4 families, the Atmel AT40K family, the Lattice Semiconductors ORCA2/3/4 and ispXPGA families. They are all SRAM-based FPGAs. However, Xilinx FPGAs have the largest capacity compared to others. The use of SRAM-Based FPGA, combined with reconfigurable computing technology, is very promising in space applications. In this research, to reduce the initial cost during the proof-of-concept stage, our FPGA designs will target to one particular device of the Xilinx Virtex 2 family FPGAs. The next section will review the Virtex series FPGAs’ architecture.

4.1.4 Structure of SRAM Based FPGAs

This section discusses Virtex 2 Xilinx family FPGAs, one of the most widely used FPGA families. Xilinx FPGAs consist of I/O blocks (IOB), internal configurable logic and programmable routing matrix as shown in Figure 4-1 [22]. The internal
configurable logic includes four major elements organized in a regular array. They are CLBs, BRAM, Multiplier blocks and DCM as described below.

- Configurable Logic Blocks (CLBs) provide functional elements for combinational and synchronous logic, including basic storage elements.
- Block SelectRAM memory modules provide large 18Kbit storage elements of dual-port RAM.
- Multiplier blocks for dedicated multipliers.
- DCM (Digital Clock Manager). This block provides self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division etc.

The Xilinx Virtex 2 FPGAs consists of an array of CLBs. The FPGA XC2V1000 has a size of 40 X 32 (= 1280). The CLBs provide functional elements for combinatorial and synchronous logic. Each CLB includes four identical slices (1280 X 4 = 5120 slices). Each slice contains

- Two 4-input function generators (5120 X 2 = 10,240 function generators)
- Carry Logic
- Arithmetic logic gates
- Multiplexers and
- Two storage elements (5120 X 2 = 10,240 storage elements)
Each 4-input function generator is programmable as a 4-input LUT, 16 bits of distributed SelectRAM memory, or a 16-bit variable-tap shift register as shown in Figure 4-2 [22]. Input/output blocks (IOBs) encircle the CLB array. Each IOB can be used as single-ended input and/or output. In addition to CLBs and IOB the FPGA has some other components. There are digital clock managers, 18-bit × 18-bit multipliers and Block RAM. All components are linked together via the global routing matrix.

![Figure 4-2 Slice of Virtex 2 FPGA](image)

The CLBs are interconnected through a general routing matrix (GRM) that comprises an array of routing switches located at the intersections of horizontal and vertical routing channels. The Xilinx Virtex 2 matrix also has 40 dedicated memory blocks called block RAM (BRAM) of 18 Kbits each, clock DLLs for clock-distribution delay compensation and clock domain control, and two tri state buffers (BUFT) associated with each CLB.

### 4.1.5 Suitable Platform for the Implementation of AES for On-Board Use

A typical EO small satellite has weight of approximately 100 kg and the orbit average power generated by solar panels is 30 to 60 W [2]. The imaging payload units of such satellites comprise imagers, mass memory and high-rate data transceivers and
consume up to 70% of the average orbit power. For example, the recently launched small EO satellite TopSat [12] has a weight of 110 kg with average orbit power of 55 W. The capacity of the memory data recorders of TopSat is 1 Gbytes and the down-link transmission rate through X-band is 25 Mbps.

But the demand to improve monitoring of the Earth's resources and its dynamic processes drives scientists to require high spatial and high-resolution images from Earth-orbiting satellites. In [87], the authors projected that the demand for data rates might be as high as 1 Gbps by 2010. To protect these huge amounts of valuable data during transmission to ground, high-speed on-board encryption using the latest encryption algorithms like AES needs to be carried out. In order to meet this requirement, hardware implementation of the AES should be considered.

Advantages of SRAM-based FPGAs such as flexibility of design, shorter time-to-market, lower cost, remote configurability etc., make them particularly suitable for use in small satellite on-board systems.

4.1.6 Overview of the Radiation Environment and Effects on Integrated Circuits

The space environment is significantly different from the terrestrial environment. The lack of atmospheric protection increases the incident radiation on satellite electronic components. Satellite electronic systems include a large variety of analog and digital components that are potentially sensitive to radiation and must be protected or at least qualified for space operation. The interaction of such radiation with electronic devices can cause failure, degradation or malfunctioning in their performance [24].

There are three primary radiation components of the natural space environment affect electronic devices. Firstly, planetary magnetic fields trap belts of high-energy protons and electrons, thus subjecting satellites to large fluxes of these particles when they pass through the radiation belts. Second, galactic cosmic rays, highly energetic particles, exist in space. Third, solar flares produce varying quantities of electrons, protons, and lower energy charged particles. Solar flare activity varies widely at different times. During periods of high solar activity, very high fluxes of particles may occur over time periods of hours or days. In low earth orbit (LEO), the main radiation
source comes from electrons and protons and in geo-stationary earth orbit (GEO); the primary source comes from electrons and solar flares. Hence the impact of radiation on on-board semiconductor devices depends on orbit altitude, orientation, and time [82].

The total ionizing dose (TID), refers to the amount of energy that ionization processes create and deposit in a material (such as semiconductor or insulator), when energized particles pass through it, causing ionization. TID, mostly due to electrons and protons, can result in device failure. TID is measured in terms of the absorbed dose, which is a measure of the energy absorbed by matter. Absorbed dose is quantified using either a unit called the rad (an acronym for Radiation Absorbed Dose) or the SI unit which is the gray (Gy). In the space environment, the total ionizing dose can cause device failure. Satellites typically encounter TID between 10 krad(Si) and 100 krad(Si).

Many current COTS parts are very susceptible to total dose damage, and may fail at total-dose of 5 krad(Si) or even less. Total dose is therefore an issue in spaceflight where long mission lifetimes, and/or exposure to the high doses of radiation means that these dose levels can be easily exceeded [71,82]. TID effects may be mitigated using radiation hardened devices and shielding. Electrons and low energy protons can be partially mitigated with shielding. Radiation hardening is achieved at different levels starting from process level, cell level to gate level etc. In addition today's technology is moving towards smaller feature size devices i.e. scaling. As feature size decrease, charge trap become less significant, leading to a trend toward improved total-ionising dose performance. However, whilst TID effects are becoming less of an issue, new threats have emerged.

A Single Event Effect (SEE) is the main concern in space, with potentially serious consequences for the application, including loss of information and functional failure. SEE occurs when charged particles hit the silicon transferring enough energy in order to provoke a fault in the system. SEE can have a destructive or transient effect, according to the amount of energy deposited by the charged particles and location of the strike in the device. Single Event Latchup (SEL), type of SEE effect, is a condition which causes loss of device functionality due to a single event induced high current state. Normally, SEL is measured by linear energy transfer (LET) which is a
measure of the energy deposited per unit length as an energetic particle travels through a material. The main consequences of the transient effect, also called Single Event Upset (SEU), are bit flips in the memory elements [82]. SEU has been constantly magnified in the past years, caused by the continuous technology evolution that has led to more complex architectures, with a large amount of embedded memories, followed by an amazing scaling down process of transistor dimensions.

The fabrication technology process of semiconductor components is in continuous evolution in terms of transistor geometry shrinking, power supply, speed and logic density. As stated in [81], drastic device shrinking, power supply reduction and increasing operating speeds significantly reduce the noise margins and thus the reliability that ICs face from the variance of internal sources of noise.

The fabrication process is now approaching a point where it will be unfeasible to produce ICs that are free from upset effects. A more significant problem is related to SEU. The necessity to protect integrated circuits against upsets has become more and more eminent. Experiments presented in [81] indicate that neutron particles present in the atmosphere are capable of producing SEU in avionics. Recent studies also show that memory cells composed of transistors with channel length smaller than 0.25 μm and combinational logic composed of transistors with length smaller than 0.13 μm may be subject to upsets while operating in space environment or at sea level. Terrestrial applications that are determined as critical such as bank servers, telecommunication servers and avionics are more and more considering the use of fault-tolerant techniques to ensure reliability.

The space market interest of using COTS and military devices in space applications and the constant increase in the radiation sensitivity of integrated circuits driven by the process scaling, have brought the necessity of researching fault-tolerating techniques for ICs able to cope with the radiation effects at sea level, and also qualifying the design for space applications [82]. Based on the definition of fault-tolerance, the goal is to maintain the IC operating correctly despite the existence of upsets. Although many techniques have been developed in the last few years attempting to avoid SEU, efficient fault-tolerant solutions are still a challenge for the
next generation semiconductor industry, especially because of the complexity of the new architectures.

4.1.7 Radiation Effects in SRAM Based FPGA

When an FPGA is used in space, the effects of radiation must be considered and accounted for. The lack of atmospheric protection increases the incident radiation, which can produce soft and hard circuit faults. The advantages of using SRAM-based FPGAs for space applications have been discussed in section 4.1.3. As reprogrammable technology has been evolving at a rapid pace, a number of mitigation techniques have been proposed to cope with radiation issues.

The Virtex family from Xilinx is one of the most popular SRAM-based programmable devices used in the market nowadays, because of its high density and high-performance. It supports a wide range of configurable gates, from 50,000 to more than 6 million gates. As discussed in Section 4.1.4, the Virtex architecture consists of a flexible and regular matrix composed of an array of CLBs surrounded by programmable input and output blocks (IOB), all interconnected by a large hierarchy of fast and versatile routing resources. Virtex components are programmed by loading a configuration bitstream into the FPGA [81]. The device functionality can be changed anytime by loading in a new bitstream. The bitstream is divided into frames and it contains all the information to configure the programmable storage elements in the matrix located in the look-up tables (LUT) and flip-flops, CLBs configuration cells and interconnections and embedded memories. All these bits are potentially sensitive to SEUs.

4.1.8 Fault-Tolerant Approaches for SRAM Based FPGA Design

There are two ways to implement fault-tolerant circuits in SRAM-based FPGAs, as shown in Figure 4-3 [81]. The first possibility is to design a new FPGA matrix composed of fault-tolerant elements. These new elements can replace the old ones in the same architecture topology or a new architecture can be developed in order to improve robustness. The cost of these two approaches in very high and it can differ according to the development time, number of engineers required to perform the task.
and the foundry technology used. Another possibility is to protect the high-level description by using some sort of redundancy, targeting the FPGA architecture. In this way it is possible to use a commercial FPGA part to implement the design and the SEU mitigation technique is applied to the design description before the description is synthesized in the FPGA. The cost of this approach is less compared to the previous one because, in this case, the user is responsible for protecting his/her own design, and the solution does not require new chip development and fabrication. In this way the user has the flexibility of choosing the fault-tolerant technique and consequently, the overheads in terms of area, performance and power dissipation.

For a given digital circuit described in a high-level description language

How to implement a fault-tolerant digital circuit in SRAM-based FPGA?

Designing a new FPGA matrix composed of fault tolerant elements by:

Replacing elements in the same architecture topology

Developing a new architecture topology

Full hardware redundancy

Combination of hardware and time redundancy

Protecting the circuit description by redundancy, targeting the FPGA architecture

Figure 4-3 Fault Tolerant Approaches Used in SRAM Based FPGA Designs

4.1.9 SEU Mitigation Techniques in SRAM Based FPGAs

In 2003, Xilinx released its third series of radiation hardened FPGAs – the QPro Virtex 2. The capacity of the largest FPGA of this family is up to 6 million system gates. The space-related features of the Xilinx QPro Virtex and Virtex 2 radiation hardened FPGAs are summarized from Xilinx data sheets and can be found at [22].
TIDs of the Xilinx QPro Virtex and Virtex 2 FPGAs are 100k and 200 krad(Si) respectively, which indicates that they are very tolerant to total ionizing dose. From the data sheets [22], it is concluded that both the QPro Virtex and Virtex 2 FPGAs are latch-up immune. The SEL immunity of the QPro Virtex 2 can reach to LET = 160 MeV·cm²/mg.

As discussed above, the two main categories of radiation effects are TID and SEEs. Regarding to SRAM-based FPGAs, there are two types of SEEs—SEIs and SEUs. Test results of TID and SEL for the radiation-tolerant Virtex FPGAs show that these devices have excellent TID and SEL performance satisfying the requirements for use in space [22,77]. However, these radiation-tolerant FPGAs are still sensitive to SEUs. SEUs in the Virtex FPGAs can be grouped into three categories [81]: configuration upsets, user logic upsets and architectural upsets or SEFIs (Table 4-1).

<table>
<thead>
<tr>
<th>Upset Modes</th>
<th>Damage Objects</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Upsets</td>
<td>Configuration memory</td>
<td>Readback</td>
</tr>
<tr>
<td>User Logic Upsets</td>
<td>BlockRAM</td>
<td>Not feasible</td>
</tr>
<tr>
<td></td>
<td>CLB Flip-Flops (CLB-FF)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I/O Block Flip-Flop (IOB-FF)</td>
<td></td>
</tr>
<tr>
<td>Architectural Upsets</td>
<td>Controlelements of the FPGAs (e.g. configuration circuits, reset control)</td>
<td>Indirectly measurement</td>
</tr>
</tbody>
</table>

4.1.9.1 SEU Mitigation for Configuration Upset

Readback and partial reconfiguration of the Virtex series FPGAs allow a system to detect and repair SEUs in the configuration memory without disrupting its operation or completely reconfiguring the FPGA. This feature facilitates two simple techniques for maintaining coherency of the bitstream and correcting the configuration upsets—Partial Reconfiguration and Scubbing. Only the SelectMap and JTAG modes support partial reconfiguration of the Virtex FPGAs. Investigations by Xilinx have shown that partial reconfiguration in Virtex can be used for the purpose of correcting SEUs to the configuration memory array induced by cosmic rays [71,81,86]. Another efficient
method of SEU correction is Scrubbing. Scrubbing is a much simpler correction method, which omits readback and detection and simply reloads the configuration at a chosen interval. Scrubbing simply rewrites the device bitstream, so the time to repair an error is the scrub cycle time. The cycle time can be on the order of a few milliseconds and varies with device density. Continuous readback in conjunction with a detection algorithm (bit compare, CRC, etc) provides data on upsets encountered, time, and frequency. Partial reconfiguration repairs any section of the device where an error is detected.

4.1.9.2 SEU Mitigation for User Logic Upsets

The user logic contains elements not directly available in the bitstream for the purpose of upset detection. The data for the user logic in the bitstream are subject to changes in the normal function of the user-implemented logic. These include BlockRAM, CLB-FF and I/OB-FF. This kind of upset is not feasible to be detected because the state of each bit needs to be known a priori, and data in these locations change state in the normal function of the user implemented logic. To mitigate the user logic upsets mitigation techniques such as Triple Modular Redundancy (TMR) is necessary, which will be discussed in the next section.

4.1.9.3 SEU Mitigation for Architectural Upsets

Single Event Functional Interrupts (SEFIs) are architectural upsets incurred in the control elements of the FPGAs. It only can be detected through a unique fault "signature" [82]. The typical SEFIs are Power-On Reset (POR) upsets and the JTAG tap controller upset. POR upsets would re-initialise the FPGA. Fortunately, the probabilities of upsets to POR are small – on the order of one POR upset per 85.6 years in LEO [111]. The probabilities of upsets to the JTAG tap controller are even lower than POR – 1 upset > 700 years. The only way to completely eliminate the effect of SEFIs is to use hardware redundancy.

4.1.10 Redundancy Methods for SEU Mitigation

The traditional method for SEU mitigation is Triple Redundancy (TR) with voting, including TMR and Triple Device Redundancy (TDR). TMR is to replicate redundant instances of an entire module and mitigate the final outputs of the modules (Figure 4-
4 (a)). TDR is to use triple FPGA devices (Figure 4-4 (b)) [111]. Using TMR to the gate level is also necessary to protect the user logic on the gate level. Xilinx offers a TMR tool, which can work with any hardware description language (HDL) and any synthesis tool to automatically build TMR, called Xilinx triple modular redundancy (XTMR), technology into any Xilinx FPGA design.

![Figure 4-4 Triple Redundancy Mitigation (a) Triple Modular Redundancy (TMR) (b) Triple Device Redundancy (TDR)](image)

The SEE consortium was founded in 2002 by the JPL and Xilinx to evaluate reconfigurable FPGAs for aerospace applications. This consortium has enlarged to 14 members up to now, including the Aerospace Corporation, Air Force Research Laboratory, Lockheed Martin, Los Alamos National Lab, etc. The investigation shows that the combination of TMR and scrubbing is the most reliable and effective SEU mitigation method for the Xilinx Virtex and Virtex 2 devices [112].

The obvious disadvantage of TMR is the limitation on the design size (less than 1/3 of the total device). However, the latest Xilinx FPGAs have grown to densities of 15 million gates, which make it possible to implement the complex SOC design. Xilinx has implemented four PowerPC 405 processors into the Virtex 2 FPGA fabric and embedding high-speed multi-gigabit serial I/Os around it. Meanwhile, the radiation-hardened QPro FPGAs’ density can reach 6 million gates. Although TMR has the highest reliability for filtering single and multiple event upsets, this is also the most costly solution, which is in contradiction with the design concept of small satellite. SEU mitigation is an aspect taken into account when the SRAM-Based FPGAs are used in space. The discussion on SEU mitigation is taken into account in Chapter 5.
4.2 FPGA Development Tools & Flow

This section discusses the flow and the tools used in the FPGA implementations of the AES algorithm. The implementation flow diagram is shown in Figure 4-5. Hardware Descriptive Language (HDL) is used for the behavioural description of the AES algorithm.

**ModelSim**

ModelSim is a well-engineered logic simulator from Mentor Graphics for the simulation of hardware designs written in VHDL, Verilog or SystemC or mixture of these three languages [89]. It compiles the sources and simulates them. ModelSim is used for the simulation of the HDL design. The HDL designs were tested using the test vectors. Once the simulations are verified and running as expected then synthesis of the design is carried out. During this research ModelSim SE V6.0 was used.

**Synplify**

Synplify from Synplicity is a synthesis tool that generates gate-level netlist for the specified target FPGA from the VHDL/Verilog design source files [90]. The netlist can be optimized under various constraints, such as minimum area or maximum possible clock frequency. HDL based behavioral descriptions usually serve as an input format for the synthesis process. Further more this tool creates schematics of the design on RTL level or on logic level. In this project Synplify V 7.7 was used.

**Xilinx ISE**

Xilinx ISE is a software environment with many tools to provide all steps of the hardware design from editing the sources up to download the design into a Xilinx’s FPGA. User constraints such as frequency of operation, area requirements can be set before starting the implementation flow [22]. Design implementation is the process of translating, mapping, placing, routing and generating a bitstream file for the design. At each point of the design flow it can create a simulation model of the design. This simulation file can be verified using Modelsim. Xilinx ISE 8.1 was used during this project.
XPower
Xilinx XPower tool was used for power estimation. Power was measured using XPower tool from Xilinx ISE package. XPower tool is provided in ISE packages of version 4.1 and above [91].

4.2.1 Hardware Design Flow

Figure 4-5 FPGA Design Implementation Flow

Figure 4-5 gives an overview about the steps of the hardware design flow. AES implementations were written using the Verilog HDL language and Modelsim was used for the functional simulation of the design. The HDL designs are tested extensively using the Known Answer Test (KAT) and Monte Carlo Test (MCT) vectors described by NIST [19,21]. Once the design verification is over the next step is synthesis. Synplify is used for synthesis which converts the HDL design to a netlist for a target FPGA. Here the design can be constrained for area or timing. A simulation file is created after the synthesis and is verified using the Modelsim, again using the KAT & MCT test vectors. After the successful completion of this step, implementation is carried out using the Xilinx ISE. It takes the netlist, which is
created by synthesizing the design as one input and area and timing constraints as another input. Xilinx ISE maps, places and routes the design and generates a bit file for the HDL design which will used to configure the FPGA. The static timing analysis checks the design after the place and route against the user defined timing constraints. As the final simulation step, dynamic timing simulations are carried out. Place and route design phase creates a HDL simulation model of the design along with a database (Standard Delay Format - SDF) that contains the timing information of the routed design. This HDL simulation model with the timing database can be simulated in the Modelsim. The timing simulation verifies that the design runs with the desired speed on the target device. At each step, during the implementation flow, a simulation file is created by the ISE and it is used in Modelsim for the verification of the design using again the KAT & MCT test benches.

The XPower tool works on the principle of ‘activity rates’ or ‘toggle rates’. Activity rates are defined as the rate at which a net or logic element capacitance switches. For dynamic power calculation and display, activity rates are expressed as a function of frequency. An activity rate may be relative to a clock, in that the net or logic element switches at some percentage of the clock frequency. This is often referred to as toggle rate. Expressed as a percentage, an activity rate of 100% means a signal state change happens on average once every clock cycle with the resultant frequency being half the associated clock. For nets and logic that are not synchronized with a clock, the activity rate is just the switching rate. In order for XPower to be able to determine the power consumption of a given design, every net in the design must have an activity rate assigned to it. The activity rate is assigned using a Value Change Dump (VCD) file generated after back annotated simulations as shown in Figure 4-5. Inputs to XPower tool are the Post route netlist (NCD), the Physical Constraint File (PCF) and the VCD file. PCF file includes both the physical constraints created by the mapper and physical constraints entered by the user. XPower uses the VCD file to set toggle rates and frequencies of all the signals in the design to estimate power consumption.

As the last step in the design flow the design can be downloaded into the target FPGA and finally the design can be tested under typical operating conditions.
4.2.2 Design Parameters

The design parameters, which are used in the evaluation of the FPGA implementations, such as throughput, power, area and latency are briefly discussed in this section.

Power

The total power consumption of a CMOS circuit is the sum of static and dynamic power consumption. The static power consumption caused by the leakage current, mainly depends on the size of the chip. It is very small and can be more or less ignored here. The dynamic power consumption consists of loading and unloading the total capacitance (C_L) of the chip. Equation 4-1[88] presents the influences on dynamic power consumption. The design measures for lowering the power consumption result from minimizing the factors in this equation.

\[
P_{\text{dyn}} = C_L V_{\text{DD}}^2 f_{\text{CLKeff}} E_{\text{SW}}
\]

(4-1)

where \( C_L \) is the load capacitance, \( V_{\text{DD}} \) is the supply voltage, \( f_{\text{CLKeff}} \) is the effective clock frequency of the design, \( E_{\text{SW}} \) is the switching probability or activity of the design nets. The load capacitance on the chip \( C_L \) increases as more gates are placed on the die. This means that lowering the die size as well as reducing the supply voltage (\( V_{\text{DD}} \)) to a minimum directly reduces power consumption. These two coefficients are somehow predetermined by the low die-size constraint and the operating conditions of the chip. Assuming a fixed supply voltage, the best option for a low-power design is reducing the effective clock frequency \( f_{\text{CLKeff}} \) of the circuit. It reduces the power consumption linearly [92].

The switching activity \( E_{\text{SW}} \) of the circuit can be reduced by using a method called sleep logic. Whenever the output of a combinational circuit is not needed changes of the input data will nevertheless cause switching activity and hence power consumption inside the module although the computed data is not needed. In order to prevent this undesired switching activity the inputs of the combinational circuit are masked using AND gates. A sleep signal that disables the AND gates prevents all
switching activities of the combinational logic behind the gate because the input is constantly zero.

**Energy**

Energy consumption is calculated using (4-2). Energy consumed during encryption is calculated by multiplying the total power consumption with the simulation time.

\[
\text{Energy} = \text{power} \times \text{time} \tag{4-2}
\]

**Throughput**

Throughput of the AES implementations is calculated using the following expression:

\[
\text{Throughput} = \left\lfloor \frac{128}{n} \times f \right\rfloor \tag{4-3}
\]

where \( n \) is the number of clock cycles required to encrypt 128-bit data block and \( f \) is the frequency of operation.

**Latency**

Latency of AES implementations is measured in terms of number of clock cycles. Latency is the delay, in clock cycles, between sending a command to start encryption and the moment the first piece of encrypted data is available on the output.

**4.3 Characterisation of FPGA-based AES Implementations**

In present FPGA implementations researchers have thoroughly investigated the effect of various optimization techniques on throughput and device area. In addition to throughput and area, power consumption is a critical parameter for satellite applications. This section explores the throughput, power and area trade-offs of AES implementations targeting SRAM-based FPGAs by exploiting the algorithmic and architectural optimization techniques outlined in sections 2.6.1 and 2.6.2.
4.3.1 AES Implementations Using Algorithmic Optimizations

Three AES designs, which are based on the algorithmic optimization Options 1, 2 & 3 (Table 2-5) are used in the experimental work. The designs are realized using three different intellectual property (IP) soft cores written in the Verilog hardware description language (HDL) as follows:

Option 1 - An existing open-source AES IP core (*AES (Rijndael) Core*) is used for the FPGA implementation and power measurements for Option 1 as it uses an LUT approach for the *SubBytes* transformation and a non-LUT approach for the *MixColumns* transformation. The core is provided at the OPENCORES website [93].

The top level block diagram of Option 1 encryption core is shown in Figure 4-6 and the encryption data path is shown in Figure 4-7. The primary inputs to the encryption data path are 128 bit input data and the key and output is 128-bit encrypted data. Input clock, reset signals are used by the control logic. The input data and the key are loaded into the input registers when the load signal is asserted. Once the encryption is completed the control circuitry will assert the done signal.

![Figure 4-6 Block Diagram of the AES Encryption Core](image)

87
As discussed in Section 2.3.1, the 128-bit input data is grouped into 16 bytes and arranged into a 4 x 4 matrix called a state matrix. At the start of encryption, when load signal is asserted, during the initial round the input state matrix is XORed with the input key. The input multiplexer is used to select either the initial state matrix or the subsequent transformed round matrix. The whole process reaches the end when 10 rounds of transformation are completed. The input register is used to keep the transformed state after every round of operation. In each round, SubBytes, ShiftRows, MixColumns and AddRoundKey transformations are carried out on the state matrix. The final round doesn’t have the MixColumns transformation. The input key is expanded to generate the appropriate round keys using the KeyExpansion algorithm as discussed in Section 2.3.2. KeyExpansion transformations such as SubWord, RotWord, RCon are used to generate the intermediate round keys for the initial and each of the 10 rounds of the AES cipher.

**Option2** - The Option2 core is also an open-source AES IP core (128-bit AES core), from the OPENCORES website [93]. This IP core is entirely based on combinational logic, as it uses a non-LUT approach for both the SubBytes and MixColumns transformations. The block diagram for the non-LUT approach of Subbytes and MixColumns can be found at [110] and are described at Appendix C.
Option3 - The Option3 core is developed in-house and is based on a look-up table implementation of both the SubBytes and MixColumns transformations. This core is written in Verilog HDL and the simulation tools used for the FPGA implementation flow are Modelsim, Synplify and Xilinx ISE. The data path of AES encryption algorithm is shown in Figure 4-8. Both the SubBytes and MixColumns transformations are implemented using the T-Box and S-Box look-up tables.

![Figure 4-8 AES Data Path of the Option3 AES Core](image)

4.3.2 Effect of Algorithmic Optimization

To study the effect of algorithmic optimizations on throughput, power and area, the three AES IP cores, Option1, 2 & 3, have been implemented on XC2V1000, Xilinx Virtex 2 family FPGA. All the three options have been implemented using the FPGA implementation flow described in section 4.2 and throughput, area, power & energy consumption estimations have been carried out based on the back-annotated simulations. Table 4-2 lists throughput, dynamic and total power consumption at 25 MHz frequency of operation. The device utilization is listed as percentage of the total available 5120 slices and 40 block RAMs (BRAM). Table 4-2 also presents the estimated execution time and energy required to encrypt the test image which is shown in Figure 3-10 (a). The image has 871(W) x 868 (H) pixels and each pixel is of 24 bits, representing three spectral bands with 8 bits per band. Thus, the number of 128-bit blocks for this image is 141755. Plots 4-9 and 4-10 show the throughput and power consumption respectively, for all the three IP cores at different frequencies. All the plots have been extended until the f<sub>max</sub>, maximum frequency of operation, of their respective Options. FPGA resources utilized by these implementations are plotted in Figure 4-11. In all these three Options, key scheduling has been carried out on-line.
From Table 4-2, we can observe that Option 1 & 3 are delivering similar throughput of 267 Mbps whereas Option 2 is delivering just 25 Mbps. This is because Option 2 is implemented using 8-bit architecture for data path and operates on a single state byte at a time whereas Option 1 & 3 uses 128-bit architecture and operates on the whole state matrix (16 bytes) at once. Hence Option 1 & 3 cores take less clock cycles for encryption of a block than Option 2. Option 1 & 3 take 12 clock cycles whereas Option 2 takes 127 clock cycles for encryption of one block of data. Using equation 4-3, at 25 MHz, Option 1 & 3 IP cores achieve a throughput of 267 Mbps and the Option 2 IP core 25 Mbps.

From plots in Figure 4-9 & 4-10, it can be observed that even though Option 1 & 3 achieve identical throughput, Option 3 consumes more power and area compared to Option 1. This is because Option 3 implementation uses look up tables (T-Box & S-Box), it consumes more power and occupies more device area (slices & BRAMs) compared to Option 1. This is reflected in the energy consumption as well.

Between Option 1 & 2, Option 2 consumes lesser power and delivers lower throughput. This is mainly because Option 2 uses 8-bit architecture path. Option 2 delivers 10 times lower throughput than Option 1. However power consumption, even though low, is comparable to Option 1. This is because, in Option 2 SubBytes are implemented using combinational logic and hence the switching activity will be more compared to Option 1 which uses S-Box look-up table. Also combinational logic is mapped onto the configurable logic block (CLB) slices of the FPGA whereas look-up tables are mapped onto the embedded Block RAMs. In Xilinx FPGAs, the design mapped onto the embedded resources (BRAM) consumes less power compared to the design mapped onto the CLB slices [94]. In addition to voltage, frequency of operation, load capacitance, power consumption is proportional to switching activity too. From Table 4-2 it can be observed that Option 2 consumes more energy than Option 1, even though it takes less power. From the above analysis, it is observed that Option 1 consumes less energy and occupies less area than other Options and hence is the better choice among the three.
Chapter 4. Design Space Exploration of the AES Algorithm

Figure 4-9 Throughput Vs frequency for AES implementation Option 1, 2 & 3

Figure 4-10 Dynamic Power Consumption Vs Frequency for the AES Implementation Option 1, 2 & 3

Figure 4-11 Slice Utilization of the Virtex 2 XC2V1000 FPGA for AES Implementation Options 1, 2 & 3
Chapter 4. Design Space Exploration of the AES Algorithm

Table 4-2 AES Implementations using Xilinx Virtex 2 Device XC2V1000 (On-line Key Expansion)

<table>
<thead>
<tr>
<th>Option No. (data path width)</th>
<th>Throughput (Mbps)</th>
<th>Dynamic Power (mW)</th>
<th>Total Power (mW)</th>
<th>Execution time (ms)</th>
<th>Energy (mJ)</th>
<th>Device Utilization</th>
<th>Max. Freq f_{\text{max}} (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option1 (128-bit)</td>
<td>267</td>
<td>534</td>
<td>885</td>
<td>68</td>
<td>60.2</td>
<td>452 (8%)</td>
<td>20 (50%)</td>
</tr>
<tr>
<td>Option2 (8-bit)</td>
<td>25.1</td>
<td>197</td>
<td>548</td>
<td>720</td>
<td>394.5</td>
<td>994 (19%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Option3 (128-bit)</td>
<td>267</td>
<td>779</td>
<td>1130</td>
<td>68</td>
<td>76.8</td>
<td>1226 (23%)</td>
<td>40 (100%)</td>
</tr>
</tbody>
</table>

* Estimated execution time = (no. of blocks in the test image \times no. of cycles to encrypt one block) / freq.

4.3.3 AES Implementations Using Architectural Optimizations

In section 4.3.1, we have used algorithmic optimizations which are specific to the AES algorithm transformations. As discussed in section 2.6.1, in addition to algorithmic optimisations, architectural optimisations are also adopted in AES implementations. Unlike algorithmic optimisations, architectural optimisations such as pipelining, sub-pipelining and loop unrolling are independent of the transformations of the AES algorithm.

Three different AES designs, which are based on the architectural optimization, PP-10, SP-1-10 and SP-2-10 are developed. The designs are realized by implementing these in the Verilog HDL and following the subsequent implementation flow as follows:

**PP-10** – This is the fully pipelined implementation of the AES. In a pipelined implementation of the AES, the iterative loop is unrolled and registers have been inserted at the end of every round as shown in Figure 4-12. The architecture is called fully pipelined where the number of pipelined stages equals the number of rounds. As all the ten rounds are unrolled and pipelined, it is referred to as fully-pipelined implementation of the AES, PP-10.
SP-1-10 & SP-2-10- The SP-1-10 & SP-2-10 are implemented using the sub-pipelining within the AES datapath. Sub-pipelining inserts pipeline stages within the round transformation itself as shown in Figure 4-13. Once the pipeline is full, this implementation will produce encrypted output at every clock cycle. By adding sub-pipeline stages, the round function of each pipeline stage is sub-divided into smaller functional blocks. This results in a decrease of the delay between the pipeline stages and hence, increases the maximum frequency of operation. However, each subdivision of the round function increases the latency, the number of clock cycles required to perform an encryption, by a factor equal to the number of sub-divisions. One sub-pipeline within a round of fully pipelined AES (PP-10) is referred as SP-1-10. If there are two sub-pipeline stages in a PP-10 round then it is referred as SP-2-10.

Figure 4-13 Block Diagram of the Sub-Pipelined implementation of the AES

Here the implementations PP-10, SP-1-10, SP-2-10 are expanding the key off-line and storing the key in registers. For iterative implementation of the AES the key is expanded and stored in the registers. Key selection for the appropriate Roundkey during the encryption is carried out using control logic. For pipelined implementations
also the key is expanded before hand and stored in the registers but there is no need for the key multiplexing as loop is fully un-rolled as in Figures 4-12 and 4-13. However, the key can be expanded on-line for pipelined implementations provided the key expansion unit is also divided into the same number of pipelined and sub-pipelined stages as in the main data round unit.

4.3.4 Effect of Architectural Optimization

Throughput, device utilization and power estimations of PP-10, SP-1-10 and SP-2-10 are carried out based on back-annotated simulations and tabulated in Tables 4-3 & 4-4. These implementations are compared with the iterative implementation of the AES, Option1.

From Table 4-3, it can be observed that the pipelined implementations, PP-10, SP-1-10, SP-2-10, require more slices and BRAM compared to the iterative architecture of Option1. However, they consume less power compared to iterative implementation of the AES. This is because each pipeline handles only one round and passes the result to the next round and also the output of each pipeline round connects only to the input of the next round. This not only eliminates shared data path, control and storage, but also shortens the length of the connections between stages. The highly localized wiring reduces propagation delays, wire capacitance and loading so the power is reduced [95,96]. Iterative option consumes 1522 mW of dynamic power whereas PP-10 consumes 858 mW. However, SP-1-10 & SP-2-10 consume more power compared to PP-10 but still less than the power consumption of the iterative architecture. The increase in power consumption is because of additional registers used for sub-pipelining.

The iterative architecture takes 10 clock cycles for the encryption of 128-bit data block and hence delivering a throughput of 320 Mbps at 25 MHz frequency of operation. In the PP-10 implementation once the pipeline is full, after 10 clock cycles of latency; encrypted output is produced at every clock. Hence the throughput is 10 times more compared to the iterative architecture. The throughput is 3.2 Gbps in the case of fully pipelined implementation at 25 MHz. Fully pipelined implementation will produce the highest throughput of the design. However, with additional sub-
pipelined stages it is possible to reduce the delay of the pipelined round unit and hence the increase in the maximum frequency of operation as shown in Table 4-3. However, each sub-pipeline division of the round function increases the latency by a factor equal to the number of sub-divisions as shown in Table 4-4. In case of PP-10 the latency is 10 clock cycles whereas in SP-1-10 and SP-2-10 implementations latency is 20 and 30 clock cycles respectively.

It can also be observed from Table 4-3 that the sub-pipelined implementations (SP-1-10 & SP-2-10) occupy a similar number of slices as the pipelined implementation (PP-10). PP-10 occupies 62% of the 14336 total available slices and SP-1-10 also occupies the same number of slices. This can be explained by observing the map reports generated by the Xilinx ISE. Even though these implementations occupy similar number of slices, their logic utilization (number of slice flip-flops and number of 4 input LUTs) is different as listed in Table 4-4. The Xilinx ISE tool maps the additional registers used for sub-pipelining to flip-flops and LUTs within the slices. Hence the number of slices remains the same however the utilization of slice flip-flops and LUTs will increase with the number of sub-pipeline stages.

Table 4-3 The AES Implementations with Pipelining (128-bit Data Path & Off-Line Key Expansion)

<table>
<thead>
<tr>
<th>AES Architecture</th>
<th>Throughput (Mbps)</th>
<th>Dynamic Power (mW)</th>
<th>Total Power (mW)</th>
<th>Execution Time* (ms)</th>
<th>Energy (mJ)</th>
<th>Device Utilization</th>
<th>Max. Freq f_max (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative</td>
<td>320</td>
<td>1522</td>
<td>1900</td>
<td>56.7</td>
<td>107.7</td>
<td>4347 (30%)</td>
<td>16 (16%)</td>
</tr>
<tr>
<td>PP-10</td>
<td>3200</td>
<td>858</td>
<td>1236</td>
<td>5.67</td>
<td>7</td>
<td>8892 (62%)</td>
<td>96 (100%)</td>
</tr>
<tr>
<td>SP-1-10</td>
<td>3200</td>
<td>1007</td>
<td>1385</td>
<td>5.67</td>
<td>7.85</td>
<td>8893 (62%)</td>
<td>96 (100%)</td>
</tr>
<tr>
<td>SP-2-10</td>
<td>3200</td>
<td>1133</td>
<td>1511</td>
<td>5.67</td>
<td>8.57</td>
<td>9208 (64%)</td>
<td>96 (100%)</td>
</tr>
</tbody>
</table>

*Estimated execution time = (no. of blocks in the test image* no. of cycles to encrypt one block) / freq.
From Table 4-4, it can be summarized that architectural optimization techniques such as pipelining and sub-pipelining reduce the energy consumption of the AES considerably but at the expense of a slight increase of the area. So depending on the application requirements various optimization techniques should be adopted.

### 4.3.4 Effect of FPGA Technology

The previous section analyzes throughput, power and area consumption of AES using optimizations at design level. The analysis carried out in sections 4.3.2 and 4.3.3 identified that Option 1 implementation of the AES algorithm is the optimal choice for satellite on-board use in terms of power, speed and device area. The other dimension to analyze in the design space of AES implementations is the target FPGA technology. In order to observe the technology dependence of design space parameters, we implemented the iterative Option 1 AES on different Xilinx families of FPGAs such as Virtex, Virtex 2, Virtex 4 and Spatran 3 [22]. Table 4-5 gives details about the FPGA devices under consideration and the voltages used in these devices. Core supply voltage, Vccint, is the main source of supply voltage to the design under consideration. I/O voltage, Vcco, supply voltage for the inputs and outputs of the FPGA.

#### Table 4-5 Different Xilinx Family FPGA Devices Used for the AES Implementation

<table>
<thead>
<tr>
<th>Xilinx FPGA Family (CMOS technology)</th>
<th>Device</th>
<th>Core Vol. Vccint (V)</th>
<th>I/O Vol. Vcco (V)</th>
<th>f&lt;sub&gt;max&lt;/sub&gt; for option 1 (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex (0.22 μm)</td>
<td>XCV800 BG560-6</td>
<td>2.5</td>
<td>3.3</td>
<td>50</td>
</tr>
<tr>
<td>Virtex 2 (0.15 μm)</td>
<td>XC2V1000 FF896-6</td>
<td>1.5</td>
<td>3.3</td>
<td>111</td>
</tr>
<tr>
<td>Spatran 3 (0.09 μm)</td>
<td>XC3S1500 FG676-5</td>
<td>1.2</td>
<td>2.5</td>
<td>90</td>
</tr>
<tr>
<td>Virtex 4 (0.09 μm)</td>
<td>XC4VLX25 FF668-11</td>
<td>1.2</td>
<td>2.5</td>
<td>150</td>
</tr>
</tbody>
</table>
Chapter 4. Design Space Exploration of the AES Algorithm

Figure 4-14 Dynamic Power Consumption Vs Frequency for Option 1 on Different Family FPGAs

Figure 4-15 Total Power Consumption Vs Frequency for Option 1 on Different Family FPGAs
Figure 4-15 presents a plot of the total power consumption versus frequency relationship for the IP core implementing Option 1 with respect to different FPGA families. It can be seen from Figure 4-15 that power consumption not only depends on the implementation style, but it also depends on the FPGA family chosen (technology used) and voltage levels used in the FPGA. Figure 4-14 presents a plot of the dynamic power consumption versus frequency relationship for the IP core implementing Option 1 with respect to different FPGA families. It is seen in Figures 4-14 & 4-15 that FPGA families with lower supply voltages consume lower dynamic power, as expected. It is observed that, the Option 1 implementation consumes almost half the power when implemented on Spartan 3 XC3S1500 FPGA compared to Virtex XCV800 FPGA.

The static power consumption of an FPGA remains constant irrespective of the implementation and frequency of operation as it mainly depends on the technology used for the fabrication of the device. At low frequency of operation static power is dominant and is comparable to dynamic power. But at higher frequencies, static power has a lower share in total power consumption. The split between static and dynamic power consumption for the AES implementation Option 1 on a Spartan3 XC3S1500 Xilinx FPGA at different frequency is shown in Figure 4-16. The static power consumption in Figures 4-16 (a) and (b) is the same (178 mW) but it forms different portions of the total power.
It can be concluded from these observations that in order to get low power, it is not only necessary that low power optimisation techniques are followed at design level, but the FPGA should be chosen so that it uses low power technology and low supply voltages. Also selecting the right frequency of operation and switching off the FPGA device when not in use will reduce static power consumption.

4.3.5 Discussion

AES can be implemented using various algorithmic and architectural optimisation techniques. Algorithmic optimisations are divided into different options such as Option1, 2 and 3. From the analysis in section 4.3.1, it is concluded that Option1 is the optimal implementation in terms device area, power, energy and throughput. This is in line with the observation from Table 2-8 that most of the published FPGA implementations of AES have adopted Option1. Architectural optimisations such as pipelining and sub-pipelining are applied to the Option1 AES. It is observed that pipelining and sub-pipelining reduce power and energy consumption but at the expense of FPGA area. So the architectural implementations may be considered for on-board implementations when device area is not a big issue. It is also illustrated that using low power FPGAs such as Spatran 3 or Virtex 4 could further reduce the power consumption of the algorithm.

4.4 Conclusions

Implementations of the AES algorithm using both the algorithmic and architectural optimization techniques have been carried out and design parameters are explored in order to identify a suitable implementation approach for space applications.

Algorithmic optimisations are divided into different options such as Option1, 2 & 3. All these three options have been implemented on Xilinx family FPGAs using soft IP cores.

AES implementations using architectural optimizations such as pipelining and sub-pipelining have also been implemented. It is demonstrated that architectural optimizations reduce power and energy consumption considerably but at the expense
of FPGA area. From the analysis of the implementations it is observed that Option 1 is the optimal choice for applications, including satellite on-board use, where throughput needs to be in the order of hundreds of Mbps but requires less device area. In addition, Option 1 consume a very small portion of the power available to the payload unit. It is also recommended to use low power FPGAs such as Spatran 3 or Virtex 4 to further bring down the power consumption of the algorithm.
Chapter 5

5 Fault Tolerant Model of the AES Algorithm

5.0 Introduction

In the previous chapter various implementations of the AES algorithm have been discussed targeting FPGA platforms. Various optimization techniques are adopted and design parameters such as throughput, area, power, energy have been explored. Suitability of these implementations for on-board use has been discussed. The present FPGA implementations have achieved the throughput demanded by present EO satellites and consuming low area and power.

In addition to high throughput and low power consumption, fault detection and tolerance is very important particularly in satellites. This is because satellites are in contact with ground station for very short durations of time. If faulty data is transmitted to the ground station, the user's request for data re-transmission has to wait until the next satellite revisit period, with revisit time varying from a couple of hours to weeks.

As discussed in Chapter 4, satellites operate in a harsh radiation environment and consequently any electronic system used on-board, including the encryption processor, is susceptible to radiation-induced faults. Most of the faults that occur in satellite on-board electronic devices are radiation induced bit flips called SEUs. SEUs can corrupt the data during on-board encryption. The other source of faults is noise in the transmission channel. Satellite data can get corrupted during transmission to ground due to this noise. All implementations targeted for on-board use should be robust to faults to tolerate harsh radiation environment.
Chapter 5. Fault Tolerant Model of the AES

Extensive research has been carried out and published on the impact of faults during transmission; however, no work has been carried out on the combined effect of the fault occurrence and the choice of the AES mode during encryption for both radiation induced faults and transmission faults. In this chapter latter issue is addressed. A detailed analysis is carried out to study the impact of faults during encryption and during transmission and the results are compared for all the AES modes. Also suitable modes of AES are recommended for satellite on-board use.

The chapter is organized as follows. Section 5.1 describes the impact of faults during on-board encryption caused by SEUs and faults during transmission caused by noise. Section 5.2 discusses the propagation of transmission faults and SEU faults in each of the five modes, ECB, CBC, CFB, OFB and CTR. Multispectral satellite images from SSTL have been employed to demonstrate the SEU propagation in the AES modes. Section 5.3 presents a parity based fault detection model of the AES. And section 5.4 presents the fault detection and correction model of the AES based on the Hamming Codes. Section 5.5 concludes the chapter.

5.1 Faults in Satellite Data

Bit-flip faults can occur during encryption as satellites operate in a harsh radiation environment and therefore any electronic systems used on-board, such as processors, memories etc., are very susceptible to faults induced by radiation. There is no exception for an encryption processor used on-board, which should be robust enough to faults in order to avoid transmission of corrupted data to ground. Most common and frequent radiation faults in satellite on-board electronics are bit flips called SEUs. SEUs are soft or temporary faults and correcting them can restore the normal operation of the device.

A study measuring the fault propagation in one block of AES has reported that even a single-bit fault during the encryption process can result in many faults in the final encrypted data and on average 50 % of the bits in the final encrypted data block will be corrupted [97]. In this research, we have extended the study in [97] a step further from fault propagation within a block to within multiple blocks as modes of operation involve multiple blocks during encryption [26].
Bit-flip faults can occur in the satellite channels due to noise during transmission of encrypted data to ground. There are techniques like Forward Error Correction (FEC) in place to detect these faults and correct them. Using FEC technique extra bits are added to the data to allow the receiver to correct some errors without having to request a retransmission of data [99,101,102]. The maximum fraction of errors that can be corrected is determined in advance by the design of the code, so different forward error correcting codes are suitable for different conditions.

In the feedback modes the faults in one block can propagate to other blocks because of the feedback. We have investigated how a single-bit fault occurring during encryption and during transmission can propagate to subsequent blocks. An elaborate study has been carried out to measure the fault propagation in the feedback modes in order to propose a suitable mode of encryption for satellite on-board use. We believe that this is the first attempt to study the impact of faults with the choice of modes in order to recommend the most suitable mode for on-board satellite application of AES.

5.2 Fault Propagation in AES Modes

**ECB:** If an SEU occurred during encryption due to radiation, then the corresponding cipher data block and hence the subsequent entire plain data block will be garbled when decrypted.

If a single bit of the cipher data block is corrupted due to noise in the transmission channel, then the entire corresponding plaintext block will also be corrupted. These faults are not propagated to other blocks, as there is no feedback. The faults are just confined to the concerned block only.

**CBC:** The effect of SEU during encryption in the CBC mode is illustrated in Figure 5-1 [26], where the SEU occurrence is marked by the star symbol • and the corrupted data blocks are represented by black boxes. If an SEU occurs while encrypting the plain block P1, the cipher block C1 will be corrupted and hence the decrypted block P1 will also be corrupted. However, this corrupted data is not propagated to the subsequent blocks despite the feedback. The reason for this is that the corrupted
cipher block C1 is XOR-ed twice - with the plain block P2 before encryption and with the cipher block C2 after decryption - as shown in Figure 5-1 [26]. Performing the XOR operation twice with this corrupted cipher block C1 neutralizes the fault and prevents propagation of faults to subsequent blocks as shown below:

\[ P \oplus X \oplus X = P \]  

(5-1)

Where X is the faulty data and P is the plain data.

In contrast, a fault occurring in an encrypted block during transmission propagates to the next block, as shown in Figure 5-2 [26], where the transmission fault is shown by the star symbol * during the transmission of the cipher block C1. The decrypted block P1 is completely garbled and the subsequent decrypted block P2 will have bit errors at the same positions as the original erroneous block C1 [37]. The decrypted blocks following the second block will not be affected by the fault. Hence, the CBC mode is self-recovering (self-synchronizing).

Figure 5-1 Fault Propagation during Encryption in CBC Mode

Figure 5-2 Transmission Fault Propagation in CBC Mode
OFB: If a SEU occurs during encryption in the OFB mode then all the subsequent blocks will be corrupted starting from the point where the fault has occurred. This is because the keystream required for encryption and decryption is independent of the plain and cipher data in the OFB mode, and hence the feedback propagates the faults from one block to another until the end of the encryption process.

This is demonstrated by introducing an SEU during the encryption of a plain multispectral satellite image. The satellite image in Figure 5-3 (a) [12] is a part of an SSTL multispectral image of North Sumatra taken on the 4th January 2005 in the aftermath of the Tsunami disaster. The image has 500 X 500 pixels and each pixel is of 24 bits, representing 3 spectral bands with 8 bits per band. Thus, the number of 128-bit blocks for this image is 46875. Figure 5-3 (b) shows the fault propagation for a single-bit error, which was introduced during the encryption of 20,000th block at the SubBytes transformation of the 4th byte in the third round. Figure 5-3 (c) shows the propagation a single-bit error that was introduced during the encryption of 40,000th block at MixColumns transformation of 7th byte in the 6th round.

Figure 5-3 (a) Plain Image (b) Decrypted image with SEU at 20,000th Block (c) Decrypted Image with SEU at 40,000th Block
If, in contrast, a bit is corrupted during transmission, only a single bit in the plain data is affected and the error does not propagate to other parts of the message again for the same reason that the keystream doesn’t depend on the plain or cipher data. So the transmission fault is not propagated. This property is very useful to applications such as satellites where the transmission channels are very noisy. Hence the OFB mode has an advantage over the CBC and CFB modes in that any bit errors that might occur inside cipher data are not propagated to affect the decryption of subsequent blocks.

**CFB:** Due to an SEU during encryption in CFB mode, the corresponding plain data block will be garbled and the faults are not propagated to subsequent blocks. This is again because of the XOR property as described by Equation 5-1. The keystream used during encryption and decryption depends on the cipher data of previous block as in CBC mode. So performing XOR two times with the corrupted data neutralize the fault and prevents propagation of faults to subsequent blocks.

However in contrast to CBC, in the CFB mode a transmission fault in an encrypted data block propagates to the next block, which is corrupted completely. This is because during decryption first the XOR operation is carried out followed by encryption as shown in Figure 5-4 [26]. Also the blocks following the second block will not be affected by the error. Therefore, CFB is also known as self-recovering (self-synchronising).

![Figure 5-4 Propagation of Transmission Fault in CFB Mode](image-url)
Chapter 5. Fault Tolerant Model of the AES

**CTR**: In CTR mode either the SEU fault or the transmission fault propagates to only one block as in the ECB mode as there is no feedback here to propagate the faults. An SEU fault during encryption corrupts one complete data block whereas a transmission fault corrupts only the corresponding single bit in the block.

### 5.2.1 AES Modes for On-Board Use

It has been observed that SEU inflicted single-bit errors can propagate from one block to multiple blocks depending on the mode of operation. In the case of the ECB, CBC, CFB and CTR modes an SEU corrupts one block of data whereas in case of the OFB mode it can propagate to the whole data starting from the point where the SEU has occurred. Hence the CBC, CFB and CTR modes are suitable for use on board as they propagate faults to just one block. On the other hand ECB is not suitable for satellite applications, even though it propagate faults to just one block, as it reveals patterns in the encrypted data, as discussed in section 3.4.

Faults occurring during transmission can also propagate from one block to multiple blocks depending on the mode of operation. It has been observed that in the case of the ECB mode the faults propagate to one block, whereas in the CBC and CFB modes the faults can propagate to two blocks. In contrast, in the OFB and CTR modes, only a single bit in the plain data is affected and the error does not propagate to other parts of the image. Based on this analysis, we conclude that the OFB and CTR modes are more favourable for noisy channels, because unlike other modes, single bit transmission errors in the cipher data are not expanded in the received plain data. The OFB mode is also recommended in [98] as a more suitable option for satellite communications compared to other modes of DES, however, it is very sensitive to SEUs as demonstrated above. Hence the OFB mode could be used on board only if an error-free AES encryption scheme as the one proposed in the next section is employed.

Table 5-1 summarizes the amount of data corrupted due to single bit faults during encryption and transmission depending on the AES mode used. It can be seen from Table 5-1 that the occurrence of single bit errors during on-board encryption in all AES modes results in fault propagation. This makes a very strong case for
development of a self-repairing AES scheme as it will prevent faulty data transmissions in satellites. The above fault propagation analysis is carried out by taking both imaging and non-imaging (telemetry) data. Examples of non-image data are listed in Appendix B.

Table 5-1 Fault Propagation Due to Single Bit Errors During Encryption and Transmission

<table>
<thead>
<tr>
<th></th>
<th>ECB</th>
<th>CBC</th>
<th>OFB</th>
<th>CFB</th>
<th>CTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Data Corrupted Due to SEU During On-board Encryption</td>
<td>One block</td>
<td>One block</td>
<td>Complete data from the point where fault has occurred</td>
<td>One block</td>
<td>One block</td>
</tr>
<tr>
<td>Amount of Data Corrupted Due to Faults During Transmission</td>
<td>One block</td>
<td>Two blocks</td>
<td>No fault propagation</td>
<td>Two blocks</td>
<td>No fault propagation</td>
</tr>
</tbody>
</table>

So far, in this chapter, fault propagation in five popular modes of the AES such as ECB, CBC, OFB, CFB and CTR have been analyzed in detail. SEUs are the most common faults that occur on-board due to radiation. The impact of SEU faults occurring during on-board encryption has been analyzed. In addition, an analysis of faults that occur during transmission due to noise has been carried out, as satellite channels are very noisy. Since none of the AES modes are free from faults, error detection and correction is very important in satellites in order to prevent faulty data transmissions. Rest of the chapter discusses the fault detection and correction models for the AES algorithm.

5.3 Fault Detection

Various methods have been proposed for fault detection of AES, which are mainly aimed at avoiding cryptanalysis of AES by injection of faults. For instance, Karri et al. [103] proposed a redundancy-based technique, where a decryption module is running in parallel with the encryption module and its output is compared with the input to the encryption module to detect a fault. The fault detection can be carried out at algorithm, round or even at transformation level to improve the detection latency at the cost of extra hardware. In [97] Bertoni et al. proposed a fault-detection scheme
based on the parity error detection code (EDC). The fault is detected by comparing the predicted parity with the calculated parity at the end of each transformation. In [104], Karpovsky et al. proposed a method to reduce the number of intentional undetected faults using systematic nonlinear (cubic) robust error detection codes. In [105], Wu et al. proposed a low cost concurrent error detection technique for the AES using parity checking based on Substitution Permutation Networks (SPN) with the aim of reducing the overhead for fault detection. Breveglieri and Koren suggested in [106] that error detecting codes are a reasonable alternative to the duplication technique due to reduced hardware overhead, optimum detection rate and many degrees of freedom in choosing the desired error coverage/cost trade-off.

Fault detection alone is not enough for space applications but fault correction is equally important. Faults must be detected and corrected on-board before sending the data to ground to avoid redundant transmission and use of erroneous data. Also, if faulty data is transmitted to the ground station, the user’s request for data retransmission has to wait until the next satellite revisit period, with revisit time varying from a couple of hours to weeks. Most of the faults that occur in satellite on-board electronic devices are radiation induced single bit flips called SEU.

5.3.1 Parity Based Fault Detection AES Model

This section presents a fault detection model based on the parity bits and section 5.4 presents a fault detection and correction model based on the Hamming codes.

The fault detection AES model involves predicting the parity at the end of each transformation from the pre-calculated parity tables. The parity bit of each byte of the S-Box look-up table $S_{RD}$, $(S_{RD} \otimes \{02\})$ and $(S_{RD} \otimes \{03\})$ is pre-calculated and stored in the form of parity tables and is referred to as parity memory. The size of each of these parity tables is $16 \times 16 = 256$ bits. So the total parity memory is $256 \times 3 = 768$ bits. The pre-calculated parity tables are referred as $P_{RD}$, $P_{2RD}$, $P_{3RD}$, where $P_{RD}$ is parity of S-Box look-up table $S_{RD}$, $P_{2RD}$ is parity of $(S_{RD} \otimes \{02\})$ and $P_{3RD}$ is parity of $(S_{RD} \otimes \{03\})$ and are represented by equation 5-2.
Chapter 5. Fault Tolerant Model of the AES

\[ p(S_{RD}[a]) \rightarrow p_{RD}[a] \]
\[ p((S_{RD}[a] \otimes \{02\}) \rightarrow p_{2RD}[a] \]
\[ p((S_{RD}[a] \otimes \{03\}) \rightarrow p_{3RD}[a] \]

(5-2)

where \( a \) represents the state byte.

The parity matrix of the \textit{SubBytes} transformation is predicted by referring to the \( P_{RD} \) table. The parity matrix prediction for \textit{ShiftRows} consists of simple cyclic rotation of the \textit{SubBytes} parity bits. The parity matrix for \textit{MixColumns} is predicted with the help of \( P_{RD}, P_{2RD}, \) and \( P_{3RD} \) parity tables and it can be expressed as in 5-3. The parity for the \textit{AddRoundKey} transformation is predicted by XORing the \textit{MixColumns} predicted parity with the expanded round key parity.

\[
\begin{align*}
p_{0,j} &= P_{2RD}[a_{0,j}] \oplus P_{3RD}[a_{1,j}] \oplus P_{RD}[a_{2,j}] \oplus P_{RD}[a_{3,j}] \\
p_{1,j} &= P_{RD}[a_{0,j}] \oplus P_{2RD}[a_{1,j}] \oplus P_{3RD}[a_{2,j}] \oplus P_{RD}[a_{3,j}] \\
p_{2,j} &= P_{RD}[a_{0,j}] \oplus P_{RD}[a_{1,j}] \oplus P_{2RD}[a_{2,j}] \oplus P_{3RD}[a_{3,j}] \\
p_{3,j} &= P_{3RD}[a_{0,j}] \oplus P_{RD}[a_{1,j}] \oplus P_{RD}[a_{2,j}] \oplus P_{2RD}[a_{3,j}] \\
0 \leq j < 4
\end{align*}
\]

(5-3)

where \( p_{0,j}, p_{1,j}, p_{2,j}, p_{3,j} \) are the predicted parity bits of \textit{MixColumns} transformation derived from \( P_{RD}, P_{2RD}, \) and \( P_{3RD} \) tables.

The parity bits \( p_{2RD} \) is given by the parity check bits of \( (S_{RD} \otimes \{02\}) \). The Galois field multiplication of a state byte, \( a \), with \( \{02\} \) is defined as follows [19,97].

\[
\{02\} \otimes a = \{02\} \cdot a(x) \mod m(x) \\
= x \cdot a(x) \mod m(x) \\
= (x \sum_{i=0}^{n-1} a_i x^i) \mod m(x) \\
= a_{n-1} \sum_{i=0}^{n-1} m_i x^i + \sum_{i=0}^{n-2} a_i x^{i+1} \\
= a_{n-1} m_0 + \sum_{i=0}^{n-2} (a_{n-1} m_{i+1} + a_i) x^{i+1}
\]

(5-4)

110
where \{02\} is represented as \(x\) in polynomial form, \(m_i\) represent the coefficients of the irreducible polynomial \(m\) defined in the AES algorithm as discussed in section 2.2.3 and \(n=8\).

The parity of the above product \(p_{2RD}\) is calculated as follows.

\[
p_{2RD} = p(\{02\} \otimes a) = p(a_{n-1}m_0 + \sum_{i=0}^{n-2}(a_{n-1}m_{i+1} + a_i))
\]  \hspace{1cm} (5-5)

Using the irreducible polynomial given by (2-3) and \(n=8\), equation (5-5) can be rewritten as

\[
p_{2RD} = p(a_7 + \sum_{i=0}^{6}(a_7m_{i+1} + a_i))
\]  \hspace{1cm} (5-6)

Equation (5-6) can be further simplified to [97]

\[
p_{2RD} = a_7 + p(a)
\]  \hspace{1cm} (5-7)

The parity bits \(p_{3RD}\) is given by the parity check bits of \((S_{RD} \otimes \{03\})\). The Galois field multiplication of a state byte, \(a\), with \{03\} is defined as follows:

\[
p_{3RD} = p(\{03\} \otimes a) = p(\{02\} \otimes a \oplus a)
\]  \hspace{1cm} (5-8)

\[
p_{3RD} = a_7 + p(A) + p(A) = a_7
\]

Figure 5-5 shows the flowchart diagram of fault detection in AES using parity tables. For each transformation of AES, the parity predicted from input state is compared with the calculated parity from transformation output. If the parity bits are different then it indicates fault detection and necessary action need to be taken before continuing with the rest of the encryption process.
5.4 Fault-Tolerant AES Model

This section presents a novel fault-tolerant model for the AES algorithm, which is immune to radiation induced SEUs occurring during encryption and can be used in hardware implementations on board small OE satellites [27]. The model is based on a self-repairing error detection and correction scheme, which is built in the AES algorithmic flow and utilizes the Hamming error correcting code [101].

The proposed Hamming code based fault-tolerant model of AES can be adapted to all the five modes of AES to correct SEUs on-board. Even though the calculation of the Hamming code is carried out within the AES it does not alter any of the transformations of the algorithm and does not affect in any way the operation of AES. Also as the Hamming parity data are not sent to ground, they are not available to leak any information about the AES algorithm. Therefore it is believed that the fault-tolerant AES model does not require a cryptanalysis.
5.4.1 Model Description

The proposed fault-tolerant model is based on the single error correcting Hamming code (12, 8), the simplest of the available ECC. The Hamming code (12, 8) detects and corrects a single bit fault in a byte and it is a good choice for satellite applications, as most frequently occurring faults in on-board electronics are bit flips induced by radiation. However, the AES correction model can be extended to correct multiple bit faults by using other ECC such as the modified Hamming code (16, 8), the Reed-Solomon codes, etc. [99,100].

The Error Detection and Correction (EDAC) capability is based on predicting the Hamming code bits (also referred to as parity check bits) at the end of each transformation from the pre-calculated Hamming code. The procedure to calculate the parity check bits is discussed below.

5.4.1.1 Calculation of the Hamming Code

The parity check bits of each byte of the S-Box look-up tables are pre-calculated. These Hamming code bits can be formally expressed as below:

\[
\begin{align*}
    h(S_{RD}[a]) & \rightarrow h_{RD}[a] \\
    h((S_{RD}[a] \otimes \{02\}) & \rightarrow h_{2RD}[a] \\
    h((S_{RD}[a] \otimes \{03\}) & \rightarrow h_{3RD}[a]
\end{align*}
\]  

(5-9)

where \(a\) is the state byte and \(h\) represents the calculation of the Hamming code.

As it can be seen from (5-9), \(h_{RD}\) is given by the parity check bits of the S-Box look-up table \(S_{RD}\), \(h_{2RD}\) is given by the parity check bits of \((S_{RD} \otimes \{02\})\) and \(h_{3RD}\) is given by the parity check bits of \((S_{RD} \otimes \{03\})\).

The procedure to derive the \(h_{RD}\) parity bits is described below by taking one state byte, \(a\), represented by bits \((b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)\) as an example. The Hamming
code of the state byte, \( a \), is a four-bit parity code, represented by bits \((p_3, p_2, p_1, p_0)\), which are derived as follows:

\[
\begin{align*}
p_3 & \rightarrow \text{is parity of bit group } b_7, b_6, b_4, b_3, b_1 \\
p_2 & \rightarrow \text{is parity of bit group } b_7, b_5, b_4, b_2, b_1 \\
p_1 & \rightarrow \text{is parity of bit group } b_6, b_5, b_4, b_0 \\
p_0 & \rightarrow \text{is parity of bit group } b_3, b_2, b_1, b_0
\end{align*}
\]  

(5-10)

The Hamming bits of all the bytes of table \( S_{RD} \) are pre-calculated and stored in the form of a memory table referred to as \( h_{RD} \) table. Calculation of hamming code is illustrated in Appendix A.

The Hamming code \( h_{2RD} \) is given by the parity check bits of \((S_{RD} \otimes \{02\})\). The Galois field multiplication of a state byte, \( a \), with \( \{02\} \) is defined as in (5-4).

The Hamming code of the above product \( h_{2RD} \) is calculated as follows.

\[
h_{2RD} = h(\{02\} \otimes a) = h(a_{n-1}m_0 + \sum_{i=0}^{n-2}(a_{n-i}m_{i+1} + a_i))
\]  

(5-11)

Using the irreducible polynomial given by (2-3) and \( n = 8 \), equation (5-12) can be rewritten as

\[
h_{2RD} = h(a_7 + \sum_{i=0}^{6}(a_7m_{i+1} + a_i))
\]  

(5-12)

Unlike the calculation of parity bit of a byte, the calculation of Hamming parity bits depends on the position of the bits in a byte and therefore it is not possible to further simplify equation (5-12). Hence the \( h_{2RD} \) parity bits are calculated beforehand, and are stored in the form of a memory table, which is referred to as \( h_{2RD} \) table.

The Hamming table \( h_{3RD} \) is given by the parity check bits of \((S_{RD} \otimes \{03\})\). The Galois field multiplication of a state byte \( a \) by \( \{03\} \) can be described as follows:
\{03\} \otimes a = (\{02\} \oplus \{01\}) \otimes a \\
= x \cdot a(x) \mod m(x) \oplus a(x) \mod m(x) \tag{5-13}

Similar to the parity function, the Hamming function is also a linear operator. The Hamming code of the above product \(h_{3RD}\) is written as follows.

\[
h_{3RD} = h(\{03\} \otimes a) = h_{2RD} \oplus h_{RD} \tag{5-14}
\]

Hence, the \(h_{3RD}\) parity bits can be calculated from the \(h_{RD}\) and \(h_{2RD}\) parity bits and therefore it is not necessary to store them in the form a parity memory table. Once we have all the parity bits, the next step is to detect and correct the faults by predicting the Hamming code bits using the pre-calculated Hamming code bits.

### 5.4.1.2 Detection and Correction of Fault Using Hamming Code Bits

The Hamming code matrix of the \textit{SubBytes} transformation is predicted by referring to the \(h_{RD}\) table. The Hamming code matrix prediction for \textit{ShiftRows} involves a simple cyclic rotation of the \textit{SubBytes} Hamming code bits. The Hamming code state matrix for \textit{MixColumns} is predicted with the help of the \(h_{RD}, h_{2RD}\) and \(h_{3RD}\) parity bits and it can be expressed by the equations below:

\[
\begin{align*}
\text{h}_{0,j} &= h_{2RD}[a_{0,j}] \oplus h_{3RD}[a_{1,j}] \oplus h_{RD}[a_{2,j}] \oplus h_{RD}[a_{3,j}] \\
\text{h}_{1,j} &= h_{RD}[a_{0,j}] \oplus h_{2RD}[a_{1,j}] \oplus h_{3RD}[a_{2,j}] \oplus h_{RD}[a_{3,j}] \\
\text{h}_{2,j} &= h_{RD}[a_{0,j}] \oplus h_{RD}[a_{1,j}] \oplus h_{2RD}[a_{2,j}] \oplus h_{3RD}[a_{3,j}] \\
\text{h}_{3,j} &= h_{3RD}[a_{0,j}] \oplus h_{RD}[a_{1,j}] \oplus h_{2RD}[a_{2,j}] \oplus h_{2RD}[a_{3,j}] \\
\end{align*}
\tag{5-15}
\]

By substituting (5-15) in the equation (5-16), the Hamming code matrix for \textit{MixColumns} can be predicted with just two tables, \(h_{RD}\) and \(h_{2RD}\).
As shown in Figure 5-6, for each transformation, the Hamming code is predicted using the input data state to the transformation by referring to the parity check bit tables and also the parity check bits are calculated from the output of the transformation. The predicted and calculated check bits are compared to detect and correct the fault as discussed below.

Let the predicted check bits of the transformation input be represented by \((x_3, x_2, x_1, x_0)\) and the calculated check bits of the transformation output be represented by \((y_3, y_2, y_1, y_0)\). The location of the faulty bit is detected by comparing the predicted and calculated Hamming check bits following the bit match patterns in Table 5-2. Once the faulty bit position is identified the fault correction is performed by simply flipping that bit. The encryption is then continued without any interruption to the encryption process. Here we assume that the Hamming code tables will be protected from SEUs by traditional memory protection techniques in satellite applications like memory scrubbing and refreshing [86].
Table 5-2 Hamming Code Bit Match Table To Locate A Faulty Bit

<table>
<thead>
<tr>
<th>Hamming Code Bits</th>
<th>Faulty Bit Position in Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x_3, y_3) &amp; (x_2, y_2))</td>
<td>0</td>
</tr>
<tr>
<td>((x_3, y_3) &amp; (x_1, y_1))</td>
<td>2</td>
</tr>
<tr>
<td>((x_3, y_3) &amp; (x_0, y_0))</td>
<td>5</td>
</tr>
<tr>
<td>((x_2, y_2) &amp; (x_1, y_1))</td>
<td>3</td>
</tr>
<tr>
<td>((x_2, y_2) &amp; (x_0, y_0))</td>
<td>6</td>
</tr>
<tr>
<td>((x_1, y_1) &amp; (x_0, y_0))</td>
<td>7</td>
</tr>
<tr>
<td>((x_1, y_1))</td>
<td>1</td>
</tr>
<tr>
<td>((x_0, y_0))</td>
<td>4</td>
</tr>
</tbody>
</table>

5.4.2 Software Simulation

The AES fault-tolerant model was verified using a purpose-built software simulator written in the JAVA programming language. The model is tested through injecting faults randomly at different round, transformation, byte and bit levels. A Graphical User Interface (GUI) was also developed to effectively simulate fault injection and correction. The GUI, as shown in Figure 5-7, has three sub-frames. The input sub-frame is meant to display the input data block, encryption key, cipher block and decryption block etc. The inject error sub-frame is to simulate the error injection at different round, transformation, byte and bit position. The details sub-frame shows the intermediate state of output for every transformation and for every round in AES. The details sub-frame also shows the predicted and calculated Hamming code.
Figure 5-7 JAVA GUI To Simulate AES Encryption

Figure 5-8 JAVA GUI To Simulate AES Decryption
Figure 5-9 and 5-10 shows the fault detection and correction window of AES algorithm, where the error is injected randomly. For instance in Figure 5-9, the error is injected at at round number 4, in \textit{ShiftRows} transformation, in the 5\textsuperscript{th} byte position of state at 7\textsuperscript{th} bit position. The status bar at the lower end of the screen displays the result of fault detection and correction technique based on hamming code as discussed above.

The proposed model was tested extensively using the Known Answer Test (KAT) and Monte Carlo Test (MCT) vectors described by NIST [19,21]. The testing with the software simulator has shown that the fault-tolerant scheme using the Hamming codes (12, 8) is able to detect and correct all the faults up to bit level as expected.

Figure 5-9 JAVA GUI to simulate of fault injection and detection at 'bit' level
Figure 5-10 JAVA GUI to simulate fault injection and detection at 'bit' level

5.4.3 Hardware Implementation of the Fault-Tolerant AES Model

FPGA implementation of the AES algorithm and the proposed fault-tolerant model is carried out in order to explore the design space and to calculate the hardware overhead incurred by the AES EDAC.

As discussed in section 2.6., different approaches to the algorithmic realization of the SubBytes and MixColumns transformations can be adopted depending on the design goals. For instance, the SubBytes transformation can be implemented using a LUT approach or a non-LUT approach. In the non-LUT approach SubBytes can be calculated on the fly for each state byte using Galois field inversion. Alternatively, the SubBytes transformation is computed in advance and the results are stored in the S-Box.

Similarly, the MixColumns transformation can be implemented using a LUT or a non-LUT approach. In the non-LUT approach every column in the state is multiplied
with a predefined polynomial \{2 3 1 1\}. Alternatively, the two pre-calculated tables 
\((S_{RD} \oplus \{02\})\) and \((S_{RD} \oplus \{03\})\) can be used to carry out this transformation, which is 
called a T-Box approach. Instead of storing only the value of \textit{SubBytes} in the S-box 
approach, the T-box approach stores the values of \((S_{RD})\), 
\((S_{RD} \oplus \{02\})\) and \((S_{RD} \oplus \{03\})\). The other two transformations \textit{ShiftRows} and 
\textit{AddRoundKey} are implemented by just a cyclic left shift and an XOR operation 
respectively.

The realisation of the \textit{SubBytes} and \textit{MixColumns} transformations using LUT and non-
LUT approaches gives rise to three AES implementation options as detailed in section 
2.6.

From the analysis in 4.3.1, it can be concluded that the Option1 IP core consumes less 
energy and occupies less area than the other options and hence it is the best choice 
among the three.

In order to implement and calculate the overhead of the fault-tolerant AES model, the 
Hamming code based EDAC is incorporated into the Option1 AES IP core, which is 
identified as the optimal implementation in terms of power, throughput and area in 
section 4.3.4. Two additional Hamming tables \(h_{RD}\) and \(h_{2RD}\) are calculated using 
expression (5-10) and \(h_{3RD}\) using (5-15). For comparison purposes the EDAC 
function is also incorporated in the Option3 AES IP core which uses LUTs for both 
\textit{SubBytes} and \textit{MixColumns}.

The Hamming code prediction path is implemented as follows. For the \textit{SubBytes} 
transformation the Hamming code is predicted using the \(h_{RD}\) table. For \textit{ShiftRows} it is 
just a cyclic left shift operation of the Hamming code and it is realized as a shift-by-
wire. The Hamming code prediction for the \textit{MixColumns} transformation is carried out 
using equation (5-16) with the help of the Hamming code tables \(h_{RD}\) and \(h_{2RD}\) and 
\(h_{3RD}\). An XOR gate is used for the prediction of the parity check bits for the 
\textit{AddRoundKey} transformation. At each transformation the parity check bits are 
calculated and compared with the predicted one. Fault detection comparators are
included at each step to compare the predicted and the calculated Hamming codes as shown in Figure 5-11. If they differ an error flag is set and the error correction logic is used to identify the corrupted bit as described in Table 5-2.

![Block Diagram of the Fault Tolerant AES Datapath](image)

Figure 5-11 Block Diagram of the Fault Tolerant AES Datapath

The FPGA implementations of the AES fault tolerant model for Option1 and Option3 are quantified in terms of area, power overhead and maximum frequency of operation as shown in Table 5-3 and Table 5-4, respectively.

<table>
<thead>
<tr>
<th>Table 5-3 FPGA Implementation of the Fault Tolerant Model with Option1 AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC2V1000 &amp; $f = 25$ MHz</td>
</tr>
<tr>
<td>Option1 IP Core</td>
</tr>
<tr>
<td>FPGA Utilization</td>
</tr>
<tr>
<td>Power (mW)</td>
</tr>
<tr>
<td>$f_{max}$ (MHz)</td>
</tr>
<tr>
<td>Option1 AES</td>
</tr>
<tr>
<td>452 (8%)</td>
</tr>
<tr>
<td>20 (50%)</td>
</tr>
<tr>
<td>885</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>Option1 AES + EDAC</td>
</tr>
<tr>
<td>675 (13%)</td>
</tr>
<tr>
<td>39 (97%)</td>
</tr>
<tr>
<td>2234</td>
</tr>
<tr>
<td>91.43</td>
</tr>
<tr>
<td>Overhead</td>
</tr>
<tr>
<td>49 %</td>
</tr>
<tr>
<td>152 %</td>
</tr>
<tr>
<td>-17 %</td>
</tr>
</tbody>
</table>
Table 5-4 FPGA Implementation of the Fault Tolerant Model with Option3 AES

<table>
<thead>
<tr>
<th></th>
<th>XC2V1000, f = 25 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FPGA Utilization</td>
</tr>
<tr>
<td>Option3 IP Core</td>
<td></td>
</tr>
<tr>
<td>Option3 AES</td>
<td>1226 (23%)</td>
</tr>
<tr>
<td></td>
<td>40 (100%)</td>
</tr>
<tr>
<td>Option3 AES + EDAC</td>
<td>1695 (33%)</td>
</tr>
<tr>
<td></td>
<td>40 (100%)</td>
</tr>
<tr>
<td>Overhead</td>
<td>38 %</td>
</tr>
</tbody>
</table>

For both fault-tolerant AES implementations, Option1+EDAC and Option3+EDAC, the throughput remains 267 Mbps at 25 MHz. However, as it can be seen from Table 5-3 & Table 5-4 the maximum frequency of operation is lower, which is due to the addition of Hamming code comparators in the data path. It is clear from Table 5-3 & Table 5-4 that the Option3+EDAC implementation consumes more power compared to the Option1+EDAC. It can also be observed from Table 5-3 and Table 5-4 that the FPGA device utilization overhead for Option3+EDAC is lower by 11% and the power consumption overhead is lower by 21%. Even though the overhead of the Option3+EDAC implementation is smaller compared to Option1+EDAC, it has higher area and power consumption. Hence the Option1 based fault-tolerant implementation is a more favorable option for on-board use. Compared with the TMR technique for SEU mitigation [14], the proposed approach provides better results in terms of area and power.

5.5 System-on-a-Chip Approach to On-Board Encryption

As discussed in Section 3.1, the main components of EO satellite are subsystems like command and data handling, attitude determination, power, propulsion, imaging payload etc. All these subsystems are interconnected to each other through an on-board bus or network.
In conventional satellite design, subsystems are physically separate from one another and each of the subsystem is composed of a combination of circuit boards and components. But the latest trend in satellite design is towards miniaturization of the satellite platforms. One of the contributing factors to miniaturization of the computing system is implementing OBC and the computing units in the other subsystems on a single chip using System-on-a-Chip (SOC) design.

![Figure 5-12 Generic Architecture of Embedded IP Core-Based SOC](image)

The research team at the Surrey Space Centre of University of Surrey has a long-term research goal, codenamed ChipSat, which aims to apply advanced micro and nanotechnologies to small satellites [71,113,114]. As part of this programme a generic on-board computing platform is developed, which is implemented as a programmable SOC through the use of high density FPGAs. An FPGA-based SOC platform offers many potential advantages including low cost, very large scale integration, low power, short time to market, and easy field upgrades of entire systems.

Computing on-board satellites is undertaken by a number of embedded microcontrollers, which are connected via a local data network. These controllers perform data processing and control functions related to different sub-systems. A
Chapter 5. Fault Tolerant Model of the AES

typical SOC system consists of a number of functional modules such as one or more processors, high performance peripherals, DMA controllers and interfaces, represented by IP cores from different vendors and implemented on a single chip. Figure 5-12 shows a generic architecture of an IP core based SOC for embedded systems.

The central processing unit of the SOC is the LEON microprocessor, which is a SPARC V8 soft IP core and written in VHDL [109]. The SPARC architecture is RISC architecture with typical features like linear 32-bit address space and few and simple instruction formats. A memory controller and some on-chip peripherals such as UARTs, timers, interrupt controller and 16-bit I/O port are integrated. The LEON IP core uses the AMBA-2.0 advanced high-performance bus (AHB) and advanced peripheral bus (APB) as on-chip bus. The AHB is for interfacing of high-performance system modules.

The SOC is an AMBA bus centric design. Therefore, any components with AHB and APB interface can be added to the chip. For highly intensive computation, one or more additional LEON processors can be plugged in. An on-board computer SOC based on the LEON2 processor was successfully implemented on a Virtex800 FPGA [71]. Data processing IP cores such as AES can also be connected to AMBA bus as shown in Figure 5-12.

LEON is a highly configurable 32-bit processor IP core developed by Gaisler Research [109]. In order to speed-up the on-board computations LEON processor can be configured to provide generic interface to a special-purpose Co-Processor (CP) and an interface to the Meiko floating-point unit (FPU) core as shown in Figure 5-12. The Co-Processor was developed in an earlier project at SSC. The Co-Processor is based on the CO-ordinate Rotation DIgital Computer (CORDIC) algorithm [116]. The other way to speedup the floating-point computations is to make use of FPU interface provided by the Leon with Meiko FPU core available from Sun Microsystems [115]. The Meiko FPU supports all single and double precision floating-point instructions as defined in the SPARC Architecture V8.
A set of peripheral block which are particularly relevant to the space application are a high level data link controller (HDLC) interface, a controller area network (CAN) interface, a SpaceWire interface etc. The SpaceWire standard is developed for usage in spacecraft applications. SpaceWire is a serial interface for a point-to-point connection. It is used to connect other spacecraft components to the OBC. It supports full-duplex data transmission with data rates from 2 Mbit/s to 400 Mbit/s [114]. The High-level Data Link Control (HDLC) protocol is used on SSTL small satellite missions for up- and downlink transfers from the spacecraft to the ground station. As the name suggests, the HDLC is a second-layer protocol in the OSI reference model. Data rates up to 10 Mbit/s are used for the up- and down-link. A further interface standard used on SSTL satellites is the Controller Area Network (CAN). It is a serial bus interface and was developed by Bosch for car applications. Its maximum data transfer rate is up to 1 Mbit/s and it is also used to link the OBC with other spacecraft components [12,114].

The fault-tolerant AES IP core developed as a result of this research will be integrated with the LEON processor to implement an on-board SOC-based crypto-processor. A feasibility study on the integration of AES IP cores onto the AMBA bus has been undertaken [108]. The AES86 IP core available from [117] is used in the study. A wrapper is developed in order to connect the AES86 core to the AMBA bus. AES86 occupies 15% of the available slices in the Xilinx Spartan-3 FPGA device XC3S1500 and utilizes 152 mW of power.

A conceptual view of an SOC approach to AES implementation in a DMC payload is presented in Figure 5-13. The DMC payload data handling block diagram is shown in Figure 3-6. The SOC based encryption module is introduced just before the transmitters. The speed of the downlink in AIsat DMC is 40 Mbps [12]. Hence in order to encrypt and transmit the images in real-time the SOC needs to encrypt the data at the rate of 40 Mbps.
5.6 Conclusion

In this chapter, SEU propagation in the popular modes of the AES such as ECB, CBC, OFB, CFB and CTR have been discussed in detail and their advantages and disadvantages for encryption of satellite images. In addition, an analysis of faults that occur during transmission due to noise is carried out, as satellite channels are very noisy.

In order to avoid data corruption due to SEUs, a fault detection and correction model of AES is proposed based on the Hamming code (12, 8). The model provides an SEU self-recovering capability, which is built in the AES data path. Also FPGA implementation is carried out to calculate the area and power consumption overhead of the proposed fault correction model. The fault-tolerant model of the AES provides adequate processing speed of 267 Mbps on a Xilinx Virtex 2 FPGA, which is in excess of the typical data rate in small EO satellites of 25 Mbps. Also it consumes a very small portion of the power available to the payload unit. The estimated hardware overhead of the optimal fault-tolerant AES IP core is 49% in terms of area and 152%
in terms of power. The model can be extended for detection and correction of multiple bit faults by using other more-sophisticated error-correcting codes such as modified Hamming code, Reed-Solomon codes etc.

The proposed fault detection and correction AES model targets the satellite application domain, however it can also be used in other applications aimed at hostile environments such as nuclear reactors, interplanetary exploration, unmanned aerial vehicles, etc.. Terrestrial applications, which require a high level of reliability, such as bank servers, telecommunication servers, etc. can benefit from the use of AES fault-tolerant techniques too.

Finally, a conceptual view of using a SOC approach to AES implementation is discussed. A block diagram of SOC based encryption for DMC satellite imaging payloads is also presented.
Chapter 6

6 Conclusions & Future Work

6.1 Conclusions

This thesis presented a novel research study of compact and robust implementations of cryptographic algorithms for EO satellites on-board use. The recent cases of unauthorized access to satellite data and the increasing demand for on-board security measures are the motivation for this research study.

The amount of data during downlink is very huge in EO satellites and at the same time satellites, particularly small satellites, has very limited power and computational resources as in terrestrial embedded systems. In addition, satellites operate in a harsh radiation environment and consequently any processor used on board, including the encryption processor, is susceptible to radiation-induced faults. So the encryption algorithm used on-board should be robust to radiation induced faults and at the same time capable of providing high-speed encryption without consuming much power and processing resources, without compromising in security (tough to break). With these constraints in mind a novel compact fault tolerant model of the latest encryption algorithm has been proposed in this thesis.

In order to protect the valuable information generated by the sophisticated on-board payloads, the AES has been identified as the suitable algorithm to perform the encryption of high data rate downlink. The novel contributions of this research are listed below in the order of importance.
Novel Fault Propagation Analysis using AES Modes

Most of the faults that occur in satellite on-board electronic devices are radiation induced bit flips called single event upsets (SEUs). This thesis presents a novel fault tolerant model to mitigate SEUs occurring on-board to avoid faulty data transmission to ground station.

A detailed novel analysis of the impact of faults during on-board encryption for five most commonly used AES modes such as ECB, CBC, OFB, CFB and CTR is presented and their advantages and disadvantages for encryption of satellite images. From the analysis it has been observed that SEU inflicted single-bit errors can propagate from one block to multiple blocks depending on the mode of operation. In case of the ECB, CBC, CFB and CTR modes a single SEU corrupts one block of data whereas in case of the OFB mode it can propagate to the whole data starting from the point where the SEU has occurred. Hence OFB mode is not suitable to satellite on-board use. ECB is also not suitable as it reveals patterns in the encrypted output. Out of the remaining three modes CBC is also not suitable for satellite applications as it is a block cipher mode and hence the speed of the processing is limited. The remaining modes CFB and CTR are more suitable for on-board use as they both are stream ciphers and hence high speed can be achieved by using parallel processing and also the propagation of the SEU faults is limited to just one block.

In addition, the impact of faults in the data occurring during transmission to ground due to noisy channels is also discussed and compared for all the five modes of AES.

Novel Fault Tolerant Model of the AES

In order to avoid data corruption due to SEUs, a fault detection and correction model of AES is proposed based on the Hamming code (12, 8). The model provides an SEU self-recovering capability, which is built in the AES data path. Also FPGA implementation is carried out to calculate the area and power consumption overhead of the proposed fault correction model. The fault-tolerant model of the AES provides adequate processing speed of 267 Mbps on a Xilinx Virtex 2 FPGA, which is in
excess of the typical data rate demanded by small EO. Also it consumes a very small portion of the power available to the payload unit. The estimated hardware overhead of the optimal fault-tolerant AES IP core is 49\% in terms of area and 152\% in terms of power. The model can be extended for detection and correction of multiple bit faults by using other more-sophisticated error-correcting codes such as modified Hamming code, Reed-Solomon codes etc.

The proposed fault detection and correction AES model targets the satellite application domain, however it can also be used in other applications aimed at hostile environments such as nuclear reactors, interplanetary exploration, unmanned aerial vehicles, etc. Terrestrial applications, which require a high level of reliability, such as bank servers, telecommunication servers, etc. can benefit from the use of AES fault-tolerant techniques too.

**Design Space Exploration of the AES**

Various implementations of the AES algorithm using both the algorithmic and architectural optimization techniques have been carried out and design parameters are explored in order to identify a suitable implementation approach for space applications.

Algorithmic optimisations are divided into different options such as Option1, 2 \&3. All these three options have been implemented on Xilinx family FPGAs using soft IP cores and throughput, area, power etc. have been extracted. Three other AES implementations using architectural optimizations such as pipelining and sub-pipelining have been carried out and evaluated in terms of throughput, area, power. It is demonstrated that architectural optimizations reduce power and energy consumption of the AES implementations considerably compared to algorithmic optimizations but at the expense of FPGA area.

From the analysis of the implementations it is observed that Option1 is the optimal choice for EO satellite applications, where throughput needs to be in the order of hundreds of Mbps but should consume less device area and power. Option1 consumes a very small portion of the power available to the payload unit. In addition, Option1
implementation has been implemented on various other Xilinx family FPGAs such as Spatran 3, Virtex, Virtex 4 and analysis has been carried out to explore the role of technology on design parameters. It is also recommended to use low power FPGAs such as Spatran 3 or Virtex 4 to further bring down the power consumption of the AES implementations.

**On-Board Security Block Diagram**

A review of on-board security architectures, algorithms and services used in existing satellites has been done and summarized. This research have identified the necessary security services required to protect the satellite links and presented the security architecture for small EO satellites.

**6.2 Future Work**

The main areas of future work are suggested as follows.

**Applying Sophisticated Error Correction Codes**

The proposed fault-tolerant model is based on Hamming (12,8) and it will detect and correct single bit errors during encryption. However the model can be extended for detection and correction of multiple bit faults by using other more-sophisticated error-correcting codes such as modified Hamming code, Reed-Solomon codes etc. Further study of how these codes can be effectively applied to AES needs to be carried out.

**Erasure Codes**

The proposed fault tolerant scheme is internal to the AES algorithm. It is possible to implement an EDAC function that is external to the AES algorithm using erasure codes [100]. Other than OFB mode, all other modes propagate faults occurring during encryption to just one block and hence a fault tolerant scheme based on erasure code will be more effective in terms of processing overhead. In the implementation of the erasure codes, the overhead caused by the additional encoding stage to encode the plain data blocks to codeword symbols needs to be evaluated. Also the impact of SEUs on the encoder needs to be further taken care of.
FPGA-Based SOC Approach to On-Board Encryption

In recent years lot of research is being carried out in SOC based satellite design and the details can be found at [71,108]. In [71,108], the core of the SOC is built upon the Leon SPARC V8 microprocessor IP core, a full featured 32-bit microprocessor core developed by European Space Agency [109]. This research was intended to connect the fault tolerant AES IP core to the Leon microprocessor through the on-chip bus. But because of lack of time this area of research was not completed. A step towards this task is carried out in [108], however integration of the AES IP Core into the SOC based design needs to be further carried out to completion on FPGA prototyping board.

Key Management for Satellites

Key management is very vital issue in any application either terrestrial or space. Any secure system is just as secure as its key management policy. A step in this direction has been carried out in [107]. However a complete analysis of proposed key management schemes needs to be evaluated further.
References


References


References


References


References


Appendix A

A. Error Correction Using Hamming Codes

This appendix presents examples of error correction using Hamming code (12,8).

A.1 Example 1

The following example illustrates how Hamming code (12,8) corrects a single bit flip in a byte. Byte 67 is corrupted to 63 because of single bit flip. The procedure to correct is described below.

Let data byte \( d \) is 63 which can be represented in binary form as 0110 0011. The calculated Hamming code or parity bits \( x_1,x_2,x_3,x_4 \) of byte 63 are tabulated as follows

<table>
<thead>
<tr>
<th></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>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Data word (without parity) 0011 0011
p0 0 0 0 1 0 0 1
p1 0 0 0 1 1 0 0 1
p2 0 1 1 1 0 0 0 1
p3 1 1 0 0 0 1 1

But the predicted Hamming code will be different for 63 as it is a faulty byte. The predicted Hamming code \( y_1,y_2,y_3,y_4 \) is 1110 and is listed below.
Appendix C: Error Correction Using Hamming Codes

Applying the correct hamming code 1110 (Predicted one) to the faulty byte 63, the word is – 010111010011. The decoding to identify the faulty bit works as described in the following table.

<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>Parity check</th>
<th>Parity bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 + 8 = 10. That means 10th bit is faulty.

Flipping the 10th bit changes 010111010011 into 010111010111.

Removing the Hamming codes gives the original data word of 0110111 = 67

A.2 Example 2: Detecting the flip in the hamming code

Data word is 67
Hamming code is 0111

<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>Parity check</th>
<th>Parity bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Fail</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Pass</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Fail</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

146
Data word is 67
Hamming code is 0101

<table>
<thead>
<tr>
<th></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>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>p1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>p2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>p3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4th bit is flipped which is p3. So the right hamming code is - 0111
Appendix B

B. Fault Propagation in Satellite Data (Non-imaging Data)

This appendix carries out the fault propagation analysis using AES modes analysis in non-imaging data.

This is the telemetry data from UK-DMC satellite.

<table>
<thead>
<tr>
<th>V</th>
<th>Lat</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>-40.6 02.6 40.0</td>
<td>-76.2 24.1 67.3</td>
</tr>
<tr>
<td>6</td>
<td>+10.4 02.0 14.6</td>
<td>+12.4 12.3 45.6</td>
</tr>
<tr>
<td>10</td>
<td>-20.6 02.0 45.3</td>
<td>+76.0 34.5 23.8</td>
</tr>
<tr>
<td>11</td>
<td>+00.9 45.6 12.3</td>
<td>-12.2 56.7 45.0</td>
</tr>
<tr>
<td>18</td>
<td>+12.1 22.4 20.5</td>
<td>-23.5 78.5 90.0</td>
</tr>
</tbody>
</table>

List 1 Plain Text (telemetry)

Error Propagation During Encryption in CBC mode

Error injected at 'State', 3rd round, 4th Byte, 5th bit in block 2.

<table>
<thead>
<tr>
<th>V</th>
<th>Lat</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>-40.6 02.6 40.0</td>
<td>LRe_wet6NkA0t 67.3</td>
</tr>
<tr>
<td>6</td>
<td>+10.4 02.0 14.6</td>
<td>+12.4 12.3 45.6</td>
</tr>
<tr>
<td>10</td>
<td>-20.6 02.0 45.3</td>
<td>+76.0 34.5 23.8</td>
</tr>
<tr>
<td>11</td>
<td>+00.9 45.6 12.3</td>
<td>-12.2 56.7 45.0</td>
</tr>
<tr>
<td>18</td>
<td>+12.1 22.4 20.5</td>
<td>-23.5 78.5 90.0</td>
</tr>
</tbody>
</table>

List 2 Decrypted Text With Error Using CBC
**Appendix E: Fault Propagation in non-Image Satellite Data**

List 3 Decrypted Text With Error in OFB Mode

V = 12, Lat = -40.6 02.6 40.0, Long = -76.2 24.1 67.3
V = 6, Lat = +10.4 02.0 14.6, Long = +12.4 12.3 45.6
V = 10, Lat = -20.6 02.0 45.3, Long = +76.0 34.5 23.8
V = 11, Lat = +00.9 45.6 12.3, Long = -12.2 56.7 45.0
V = 18, Lat = +12.1 22.4 20.5, Long = -23.5 78.5 90.0

List 4 Decrypted Text With Error Using CFB Mode

V = 12, Lat = -40.6 02.6 40.0, Long = -76.2 24.1 67.3
V = 6, Lat = +10.4 02.0 14.6, Long = +12.4 12.3 45.6
V = 10, Lat = -20.6 02.0 45.3, Long = +76.0 34.5 23.8
V = 11, Lat = +00.9 45.6 12.3, Long = -12.2 56.7 45.0
V = 18, Lat = +12.1 22.4 20.5, Long = -23.5 78.5 90.0

List 5 Decrypted Text With Error Using CTR Mode

V = 12, Lat = -40.6 02.6 40.0, Long = -76.2 24.1 67.3
V = 6, Lat = +10.4 02.0 14.6, Long = +12.4 12.3 45.6
V = 10, Lat = -20.6 02.0 45.3, Long = +76.0 34.5 23.8
V = 11, Lat = +00.9 45.6 12.3, Long = -12.2 56.7 45.0
V = 18, Lat = +12.1 22.4 20.5, Long = -23.5 78.5 90.0
Appendix C

C. MixColumns & SubBytes Block Diagrams

Block Diagram of the SubBytes Using Composite Field Mathematics
Appendix E: MixColumns & SubBytes Block Diagrams

Block Diagram of the MixColumns Using Galois Field multiplication