Mark Tehranipoor
N. Nalla Anandakumar
Farimah Farahmandi

Hardware Security Training, Hands-on!

Hardware Security Training, Hands-on!

Mark Tehranipoor • N. Nalla Anandakumar •
Farimah Farahmandi

Hardware Security Training, Hands-on!

Mark Tehranipoor
Department of Electrical & Computer Engineering
University of Florida
Gainesville, FL, USA

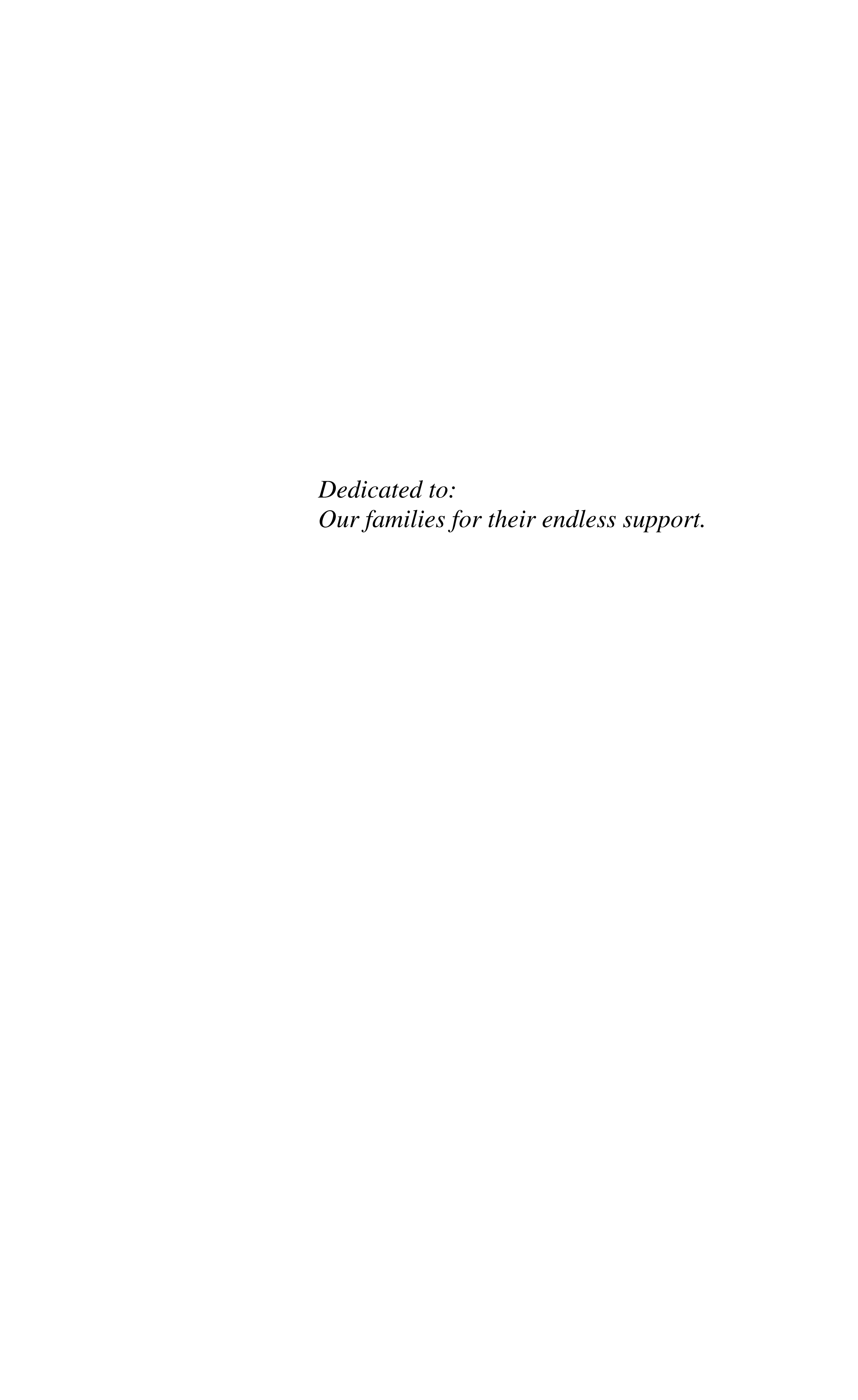
Farimah Farahmandi
Department of Electrical & Computer Engineering
University of Florida
Gainesville, FL, USA

N. Nalla Anandakumar
Continental Automotive Singapore
Security & Privacy Competence Center
Singapore, Singapore

ISBN 978-3-031-31033-1 ISBN 978-3-031-31034-8 (eBook)
https://doi.org/10.1007/978-3-031-31034-8

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland



Dedicated to:
Our families for their endless support.

Preface

All information technology systems are built using the physical hardware of the semiconductor chips found in all modern electronics, computers, communications networks, and critical cyber-physical systems. These chips are becoming cheaper, faster, and more powerful, and this has enabled the rise of the Internet of Things (IoT), autonomous systems, and artificial intelligence as key technologies of the future. As the world becomes increasingly reliant on advanced technologies for economic growth and national security, implicit trust in hardware becomes an untenable option. Moreover, trusted computing in hardware is fundamental to information security practices. The basis of security guarantees in digital systems is essentially a set of cryptographic operations executing in a hardware root of trust. Advanced cyber-attacks therefore deliberately target hardware devices in cryptosystems applications. The semiconductor chips have been targeted for reverse engineering, counterfeiting, piracy, malicious insertion, side-channel attacks, fault injection attacks, and probing attacks. These attacks, their associated vulnerabilities, root causes, and their countermeasures form the field of hardware security.

Hardware security has become an important topic recently with more and more researchers from related research domains joining this area. However, the understanding of hardware security is often mixed with computer science, electronics, cryptography, physics, material sciences, communication systems, and signal processing. It sometimes requires the construction of specialized equipment, and it usually takes some practice to acquire laboratory skills—especially where attacks involve laboratory equipment. Therefore, it is necessary to acquire laboratory skills to help both academia and industry investigate hardware attacks, countermeasures, and solutions to address hardware security problems. To address this important growing need, we embarked on this initiative for developing the first-ever textbook dedicated to hands-on hardware security training that includes different attacks deliberately targeting hardware devices and applying countermeasure techniques against them. In this book, a wide variety of topics will be covered in 16 chapters. Chapter 1 presents physical unclonable functions (PUFs), Chap. 2 provides a true random number generator (TRNG), Chap. 3 presents recycled chip detection using ring oscillator-based odometer, Chap. 4 provides a recycled FPGA detec-

tion, Chap. 5 presents techniques of hardware Trojan insertion, Chap. 6 provides hardware Trojan detection, Chap. 7 presents security verification, Chap. 8 presents power analysis attack on AES, Chap. 9 presents EM-based side-channel attack on AES, Chap. 10 presents logic locking insertion and assessment, Chap. 11 presents clock glitch fault-injection attack on AES/FSM, Chap. 12 provides voltage glitch fault-injection attack on AES/FSM, Chap. 13 presents laser fault-injection attack, Chap. 14 presents optical probing attack on logic locking, Chap. 15 presents a universal fault sensor, and Chap. 16 presents scanning electron microscope (SEM) training. This book aims to provide holistic hands-on hardware security training to upper-level undergraduate engineering students and graduate students, security researchers, practitioners, and industry professionals, including design engineers, security engineers, system architects, and chief security officers.

Any source code and other supplementary materials referenced by the authors in this book are available to readers on the hardware security lab (HSL) page at http://cad4security.org/index.php/trainings/hsl/.

Gainesville, FL, USA Mark Tehranipoor
Singapore, Singapore N. Nalla Anandakumar
Gainesville, FL, USA Farimah Farahmandi
Dec, 2022

Acknowledgments

This work would not have been accomplished without the assistance of individuals affiliated with the Florida Institute for Cyber Security (FICS) Research at the University of Florida. We appreciate their tenacious efforts, experimentation, and scientific contributions in various chapters.

Yunkai Bai and Tao Zhang, Chap. 1
Nashmin Alam, Chap. 2
Tao Zhang, Chap. 3
Tasnuva Farheen, Chap. 4
Tao Zhang, Chap. 5
Tao Zhang, Chap. 6
Nusrat Farzana and Avinash Ayalasomayajula, Chap. 7
Jungmin Park, Chap. 9
M. Sazadur Rahman, Chap. 10
Pantha Sarker and Henian Li, Chap. 13
Leonidas Lavdas and M. Shafkat M. Khan, Chap. 14
Md Rafid Muttaki, Chap. 15
Nitin Varshney and Daniel Johnson, Chap. 16

Contents

1 Physical Unclonable Functions (PUFs) 1
1.1 Introduction 1
1.2 Background 2
1.2.1 RO PUF 4
1.2.2 MiniZed Board Introduction 4
1.2.3 FPGA Development Procedure 5
1.3 PUF Performance Metrics 6
1.4 Implementation Details of the RO PUF 7
1.5 Performance Analysis and Discussion 11
1.5.1 Randomness, Uniqueness, and Reliability 11
1.5.2 NIST Statistical Test 13
1.5.3 Entropy Estimation 13
1.6 Conclusion 15
References 16

2 True Random Number Generator (TRNG) 19
2.1 Introduction 19
2.2 Background 20
2.2.1 Sources of Entropy 20
2.2.2 Ring Oscillator-Based TRNG 22
2.3 RO-Based TRNG Implementation 23
2.4 Measures of the Quality of Randomness 29
2.4.1 Entropy Estimation 29
2.4.2 Restart Experiment 30
2.4.3 Statistical Evaluation of the Output 30
2.5 Conclusion 31
References 32

3 Recycled Chip Detection Using RO-Based Odometer 35
3.1 Introduction 35
3.2 Background 36
3.2.1 Motivations and General Flow 36

3.2.2 Counterfeit Threats 37
3.3 Recycled FPGA Detection 39
3.4 FPGA Development Procedure 41
3.5 Recycled Chip Detection Experiments 42
3.5.1 Experimental FPGA Platform 42
3.5.2 Experimental Flow 43
3.6 Conclusion 49
References 49

4 Recycled FPGA Detection 53
4.1 Introduction 53
4.2 Background 54
4.2.1 Look-Up Table Structure 54
4.2.2 RO Path Formation Using XNOR and XOR Logic 55
4.2.3 Aging Mechanism 55
4.3 Classification Using Supervised and Unsupervised Methods 57
4.3.1 Supervised Classification Method 57
4.3.2 Unsupervised Classification Method 57
4.4 The Setup for the Experiment 58
4.4.1 Bitstream Generation 60
4.4.2 Bitstream Loading 62
4.4.3 Capturing Output 62
4.5 Capturing RO Frequencies and Recycled FPGA Detection 63
4.5.1 Visualization of RO Frequencies 64
4.5.2 Analysis Using Machine Learning 65
4.6 Conclusion 71
References 71

5 Hardware Trojan Insertion 73
5.1 Introduction 73
5.2 Hardware Trojan Attacks 74
5.2.1 Modern Chip Design Flow and Threat Model 74
5.2.2 Hardware Trojan Insertion 76
5.3 Trojan-Infected Implementation on FPGA 79
5.3.1 FPGA Development Flow 80
5.3.2 Experimental Setup 80
5.3.3 Trojan-Infected Design 81
5.3.4 Compiling Target Design and Trigger Trojan 82
5.4 Bitstream Tampering for Trojan Triggering 85
5.4.1 FPGA Bitstream Format Preliminaries 85
5.4.2 Bitstream Tampering Enabling Trojan Trigger 88
5.5 Conclusion 90
References 91

6 Hardware Trojan Detection 93
6.1 Introduction 93

6.2 Hardware Trojan Detection ... 94
6.2.1 Overview of Hardware Trojan ... 94
6.2.2 Pre-silicon Hardware Trojan Detection ... 96
6.2.3 Post-silicon Hardware Trojan Detection ... 101
6.2.4 Destructive Method ... 101
6.2.5 Nondestructive Method ... 102
6.3 Hardware Trojan Detection Experiment ... 105
6.3.1 Experimental Setup ... 105
6.3.2 Experimental Steps ... 107
6.4 Conclusion ... 111
References ... 111

7 Security Verification ... 115
7.1 Introduction ... 115
7.2 Background: Writing Properties ... 116
7.3 SoC Security Verification Using Property Checking ... 117
7.3.1 Security Asset Identification ... 117
7.3.2 Threat Model Identification ... 119
7.3.3 Generating Security Properties ... 121
7.4 Experimental Setup ... 121
7.4.1 AES Design ... 122
7.4.2 Security Property Development for Verification ... 123
7.4.3 Property-to-Assertion Conversion ... 124
7.4.4 Compiling Target Design and Property Verification ... 124
7.4.5 Tool 1: JasperGold Security Path Verification (SPV) ... 126
7.4.6 Tool 2: JasperGold Formal Property Verification ... 128
7.5 Conclusion ... 132
References ... 134

8 Power Analysis Attacks on AES ... 137
8.1 Introduction ... 137
8.2 Power Analysis Attacks ... 138
8.2.1 Power Consumption Characteristics of CMOS ... 138
8.2.2 Simple Power Analysis (SPA) ... 139
8.2.3 Differential Power Analysis (DPA) ... 140
8.2.4 Correlation Power Analysis (CPA) ... 142
8.3 AES Implementation on FPGA ... 143
8.3.1 Field-Programmable Logic Arrays ... 143
8.3.2 AES Algorithm Overview ... 144
8.4 Experiment Setup ... 145
8.4.1 Hardware and Software ... 146
8.4.2 Firmware Setup ... 146
8.4.3 Hardware Setup ... 147
8.5 Power Measurements on the AES Chip ... 149
8.5.1 AES Bitstream Generation ... 150
8.5.2 Capture a Power Trace ... 151

8.6 Performing AES CPA Attack ... 155
8.6.1 CPA Attack Steps ... 155
8.7 Conclusion ... 159
References ... 160

9 EM Side-Channel Attack on AES ... 163
9.1 Introduction ... 163
9.2 Background ... 164
9.2.1 Measuring EM Radiation ... 164
9.2.2 Typical EM Side-Channel Attacks ... 165
9.3 Implementation Details of Investigated AES Design ... 166
9.4 Measurement Setup ... 167
9.5 EM Measurements on the AES Chip ... 170
9.5.1 Tool Setup ... 170
9.5.2 Capture an EM Trace ... 171
9.6 Performing Correlation Electromagnetic Analysis (CEMA) Attack ... 175
9.7 Conclusion ... 178
References ... 179

10 Logic-Locking Insertion and Assessment ... 183
10.1 Introduction ... 183
10.2 Background ... 184
10.2.1 Logic Locking ... 184
10.2.2 The Threat Model for Logic Locking ... 185
10.3 Review of Existing Logic-Locking Solutions ... 186
10.3.1 Combinational Locking ... 186
10.4 Experimental Demonstration ... 190
10.4.1 Experimental Setup ... 190
10.4.2 Locking Gate Insertion ... 191
10.4.3 Random Locking Gate Insertion ... 191
10.4.4 Equivalency Checking ... 194
10.5 Conclusion ... 195
References ... 195

11 Clock Glitch Fault Attack on FSM in AES Controller ... 199
11.1 Introduction ... 199
11.2 Background ... 200
11.2.1 Fault Models ... 200
11.2.2 Clock Glitching ... 201
11.2.3 Brief Description of AES ... 203
11.2.4 Clock Glitch Attack on FSM in AES Controller ... 204
11.2.5 ChipWhisperer CW305 Board ... 206
11.3 Experimental Setup ... 207
11.4 Performing Clock Glitch Attacks ... 208
11.4.1 Performing Clock Glitch Attack ... 208

11.4.2 Glitch Explorer 213
11.4.3 Results 214
11.5 Conclusion 216
References 216

12 Voltage Glitch Attack on an FPGA AES Implementation 219
12.1 Introduction 219
12.2 Background 220
12.2.1 Voltage Glitches 220
12.2.2 Fault Models 222
12.2.3 Brief Description of AES 223
12.2.4 Voltage Glitch Attack on FSM in AES Controller 224
12.2.5 ChipWhisperer CW305 Board 225
12.3 Experimental Setup 225
12.3.1 Hardware Setup 225
12.3.2 Software Setup 226
12.4 Performing Voltage Glitch Attacks 227
12.4.1 Steps in Performing Voltage Glitch Attacks 227
12.4.2 Starting the Voltage Glitch Attack 230
12.4.3 Results 231
12.5 Conclusion 232
References 232

13 Laser Fault Injection Attack (FIA) 235
13.1 Introduction 235
13.2 Laser Fault Injection Attacks 236
13.2.1 Analysis of Laser Beams on MOSFETs 236
13.2.2 Exploitation of Laser Attacks 237
13.3 Device Under Test (DUT) Circuit on FPGA 238
13.3.1 Field Programmable Logic Arrays 238
13.3.2 Device Under Test (DUT) 239
13.4 Experimental Setup 240
13.4.1 Hardware and Software 240
13.4.2 Hardware Setup 241
13.4.3 DUT Bitstream Generation 243
13.4.4 Hardware Connection 244
13.4.5 Placement of the FPGA 244
13.4.6 Fault Injection Attack 246
13.4.7 Bitflip Observation 252
13.5 Conclusion 253
References 256

14 Optical Probing Attack on Logic Locking 259
14.1 Introduction 259
14.2 Background 261
14.2.1 Optical Probing Overview 261

14.2.2 Logic Locking 263
14.3 Experiment Setup 265
14.3.1 Programming the Sample 265
14.3.2 Sample Preparation 266
14.3.3 Measurement Setup 266
14.4 Performing the Attack 266
14.4.1 Attack on Combinational Logic Locking 266
14.4.2 Attack on Sequential Logic Locking 269
14.5 Conclusion 270
References 270

15 Universal Fault Sensor 273
15.1 Introduction 273
15.2 Background 274
15.3 FTC Sensor 275
15.4 Hardware Implementation Setup 277
15.4.1 Hardware and Software 278
15.4.2 Bitstream Generation 280
15.4.3 Capturing Output 282
15.5 Results and Analysis 283
15.5.1 EM Attack Analysis 285
15.5.2 Voltage Glitch Attack Analysis 287
15.5.3 Clock Glitch Attack Analysis 288
15.5.4 Proximity Analysis 289
15.6 Conclusion 291
References 291

16 Scanning Electron Microscope Training 293
16.1 Introduction 293
16.2 Background 294
16.2.1 Scanning Electron Microscopy 294
16.2.2 Beam Interaction 295
16.2.3 Display and Record System 298
16.2.4 Specimen Preparation 299
16.3 Setting Up the Experiment for Image Acquisition with the SEM 299
16.3.1 Sample Preparation 300
16.3.2 Sample Loading Inside the SEM 300
16.3.3 SEM Image Acquisition 301
16.4 Hardware Trojan (HT) Detection in ICs Using SEM Images 307
16.4.1 Equipment and Software Needed for This Work 307
16.4.2 Prerequisites 308
16.4.3 Experimental Setup for HT Detection in ICs Using SEM 308
16.4.4 FIB and SEM Imaging 309
16.4.5 Trojan Detection System 311
16.4.6 Cell Extraction 313

16.4.7 Synthetic Cell Image Generation 315
16.4.8 Logical Cell Recognition 317
16.5 Conclusion 318
References 318

Index 319

About the Authors

Mark Tehranipoor is currently the Intel Charles E. Young Preeminence Endowed Chair Professor and the Chair of the Department of Electrical and Computer Engineering (ECE) at the University of Florida. His current research projects include: hardware security and trust, supply chain security, IoT security, VLSI design, test, and reliability. He is a recipient of a dozen best paper awards and nominations, as well as the 2008 IEEE Computer Society (CS) Meritorious Service Award, the 2012 IEEE CS Outstanding Contribution, the 2009 NSF CAREER Award, and the 2014 AFOSR MURI award. He received the 2020 University of Florida Innovation of the year as well as teacher/scholar of the year awards. He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), IEEE International Conference on Physical Assurance and Inspection of Electronics (PAINE). He serves on the program committee of more than a dozen leading conferences and workshops. He has also served as *Program* and *General Chair* of a number of IEEE and ACM sponsored conferences and workshops (HOST, ITC, DFT, D3T, DBT, NATW, and more). He is currently serving as a founding EIC for *Journal of Hardware and Systems Security* (HaSS) and served as Associate Editor for TC, JETTA, JOLPE, TODAES, IEEE D&T, and TVLSI. He is currently serving as a founding director of Florida Institute for Cybersecurity Research (FICS) and a number of other centers with focus on microelectronics security. Dr. Tehranipoor is a Fellow of the IEEE, a Fellow of the ACM, a Fellow of the National Academy of Inventors (NAI), a Golden Core Member of IEEE CS, and Member of ACM SIGDA.

N. Nalla Anandakumar is currently a Senior Specialist on Security and Privacy at Continental Automotive, Singapore. He received his Ph.D. from IIIT-Delhi, India, in 2020, and his M.E. from Anna University, Chennai, in 2006. Prior to joining the Continental in 2023, he was a postdoctoral fellow at the Florida Institute of Cyber Security Research (FICS), University of Florida, USA. Earlier he worked as a Scientist at the Society for Electronic Transactions and Security (SETS), India from 2008 to 2020. He was also a Visiting Researcher at the Nanyang Technological

University (NTU), Singapore, in 2013. His current research interests include hardware security with a specific focus on secure software/hardware architecture design, physically unclonable functions, implementations of classical and post-quantum cryptographic algorithms, side-channel and fault injection attacks, and hardware IP protection. He has authored more than 25 technical papers in leading journals and conferences. He has served as Principal and Co-Principal Investigator of five funded R&D projects in the hardware security area. He also has served on many technical program committees as well as organizing committees of premier conferences and workshops. He served as a peer reviewer for more than 70 technical journal and conference papers. Currently, he serves as a Secretary of Hardware Security Standards Committee (HSSC) with CAS Chapter for IEEE SASD. He is a senior member of IEEE.

Farimah Farahmandi is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Florida. She received her Ph.D. from the Department of Computer and Information Science and Engineering at the University of Florida, 2018. She received her B.S. and M.S. from the Department of Electrical and Computer Engineering at the University of Tehran, Iran, in 2010 and 2013, respectively. Her research interests include design automation of System-on-Chips and energy-efficient systems, formal verification, hardware security validation, and post-silicon validation and debug. Her research has resulted in two books, seven book chapters, and several publications in premier ACM/IEEE journals and conferences, including *IEEE Transactions on Computers*, *IEEE Transactions on CAD*, Design Automation Conference (DAC), and Design Automation and Test in Europe (DATE). Her research has been recognized by several awards, including IEEE System Validation and Debug Technology Committee Student Research Award, Gartner Group Info-Tech Scholarship, a nomination for the Best Paper Award in ASPDAC 2017, and DAC Richard Newton Young Student Fellowship. She has actively collaborated with various research groups (IBM, Intel, and Cisco) that have led to several joint publications. She currently serves as an Associate Editor of *IET Computers & Digital Techniques*. She also has served on many technical program committees as well as organizing committees of premier ACM and IEEE conferences. Her research has been sponsored by SRC, AFRL, DARPA, and Cisco. She is a member of IEEE and ACM.

Acronyms

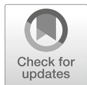
3PIP	Third-Party Intellectual Property
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AISS	Automatic Implementation of Secure Silicon
APUF	Arbiter-based PUF
ASICs	Application-Specific Integrated Circuits
BDD	Binary decision diagram
BEOL	Back-End of Line
BER	Bit Error Rate
BGA	Ball Grid Array
BMC	Bounded Model Checking
BSE	Back-Scattering Electrons
BTI	Bias Temperature Instability
CBC	Cipher Block Chaining
CEMA	Correlation Electromagnetic Analysis
CLBs	Configurable Logic Blocks
CMOS	Complementary Metal-Oxide-Semiconductor
CNC	Computer Numerical Control
CPA	Correlation Power Analysis
CPU	Central Processing Unit
CR	Correlation Ratio
CRC	Cyclic Redundancy Check
CRPs	Challenge-Response Pairs
DAG	Directed Cyclic Graph
DARPA	Defense Advanced Research Project Agency
DCM	Digital Clock Manager
DEMA	Differential Electromagnetic Analysis
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DFF	D Flip-Flops
DFT	Design-for-Test

DoS	Denial-of-Service
DPA	Differential Power Analysis
DUT	Design Under Test
ECC	Elliptic curve cryptography
ECC	Error Correction Code
ECO	Engineering Change Order
EDA	Electronic Design Automation
EM	Electromagnetic
EMA	Electromagnetic Analysis
EMFI	Electromagnetic Fault-Injection
EOFM	Electro-Optical Frequency Mapping
EOP	Electro-Optical Probing
ES	Evolution Strategies
FA	Failure Analysis
FANCI	Functional Analysis for Nearly Unused Circuit Identification
FAR	Frame Address Register
FEOL	Front-End of Line
FFs	Flip-Flops
FFT	Fast Fourier Transform
FIA	Fault Injection Attacks
FIB	Focused Ion Beam
FICS	Florida Institute for Cybersecurity
FLL	Fault Analysis-based Logic Locking
FOV	Field of View
FPGAs	Field-Programmable Gate Arrays
FPV	Formal Property Verification
FSM	Finite State Machine
FTC	Fault-to-Time Converter
GAN	Generative Adversarial Network
GDSII	Graphic Database System
GND	Ground Pin
GUI	Graphic User Interface
HCI	Hot Carrier Injection
HD	Hamming Distance
HDL	Hardware Description Language
HT	Hardware Trojan
HW	Hamming weight
I/O	Input/Output
IC	Integrated Circuit
ID	Identity
IDM	Integrated Device Manufacturers
ILA	Integrated Logic Analyzer
IoT	Internet of Things
IP	Intellectual Property
LBs	Logic Barriers

LFSR	Linear Feedback Shift Register
LNA	Low-Noise Amplifier
LR	Logistic Regression
LSBs	Least Significant Bits
LUTs	Look-Up Tables
ML	Machine Learning
MMCM	Multi-Mode Clock Manager
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
MSBs	Most Significant Bits
NAROs	Noise-Augmenting Ring Oscillators
NBTI	Negative BTI
NBTI	Negative-Bias Temperature Instability
NIR	Near-Infrared
NIST	National Institute of Standards and Technology
NMOS	N-Channel Metal-Oxide Semiconductor
NOP	No Operation
NVMs	Non-Volatile Memories
OBIRCH	Optical Beam-Induced Resistance Change
OCMs	Original Component Manufacturers
OTP	One-Time Programmable
PBTI	Positive BTI
PC	Personal Computer
PCBs	Printed Circuit Boards
PCC	Proof-Carrying Code
PDLs	Programmable Delay Lines
PEA	Photon Emission Analysis
PKC	Public-Key Cryptography
PLLs	Phase-Locked Loops
PMOS	P-Channel Metal-Oxide Semiconductor
PoC	Proof of Concept
PRNG	Pseudo Random Number Generator
PSN	Power Supply Noise
PSNIB	Power Supply Noise Inducing Block
PUFs	Physically Unclonable Functions
PV	Process Variation
QRNGs	Quantum Random Number Generator
RE	Reliability
RFID	Radio Frequency Identification
RLL	Random Logic Locking
ROC	Receiver Operator Curve
RoI	Region of Interest
RO-PUF	Ring Oscillator-based PUF
ROs	Ring Oscillators
RSA	Rivest–Shamir–Adleman
RS-LPUF	RS Latch-based PUF

RTL	Register Transfer Level
SCA	Side-Channel Attack
SE	Secondary Electrons
SEM	Scanning Electron Microscope
SEMA	Simple Electromagnetic Analysis
SHA	Secure Hash Algorithm
SIPO	Serial-In to Parallel-Out
SLL	Strong Logic Locking
SMT	Satisfiability Modulo Theorem
SoC	System-on-Chip
SPA	Simple Power Analysis
SPV	Security Path Verification
SRAM	Static Random Access Memory
SRAM-PUF	SRAM-based PUF
SV	Silhouette Value
SVM	Support Vector Machine
SWAG	State-Space-Obfuscation Waveform Attack Generator
TDC	Time-to-Digital Converter
TDDB	Time-Dependent Dielectric Breakdown
TDK	Tunable delay key-gate
TI	Technology-Independent
TRNGs	True Random Number generators
UART	Universal Asynchronous Receiver-Transmitter
UCI	Unused Circuit Identification
UF	Uniformity
UQ	Uniqueness
VCC	Positive Power Input
VIO	Virtual Input/Output
XOR	Exclusive OR (logical)

Chapter 1
Physical Unclonable Functions (PUFs)



1.1 Introduction

The demand for computer security for electronic products has increased due to the rapid expansion of the electronics industry. However, there are many threats, vulnerabilities, and risks related to electronic products such as IC counterfeiting, integrated circuit (IC) overproduction, and intellectual property (IP) piracy [28]. Key-based cryptosystems have typically been employed for IP protection purposes. However, key generation and key storage make an IC vulnerable to physical attacks and tampering [5, 23, 27]. Researchers and engineers have been inspired by the need to secure secret keys to design and implement physical unclonable functions (also known as physical one-way functions), which are simple to perform in one direction but difficult in the opposite direction [22]. Moreover, these physical unclonable functions (PUFs) are inexpensive to fabricate, inherently random, intrinsically tamper-resistant, and difficult to duplicate. This makes PUF stand out as ideal candidate for providing a tamper-resistant design for secret key generation and storage [8, 26]. According to [22], PUF interacts with challenges in a complicated way, producing responses that are unpredictable and distinctive. The randomness of the PUF response comes from the device's manufacturing process variation during fabrication, which is intrinsic to the device itself. Because of their unpredictable and unique properties, PUFs are frequently used for authentication, identification, detection of counterfeit ICs, and IC obfuscation to prevent IC piracy [8, 26]. Since the introduction of PUF in [22], it has been a field of interest that has inspired researchers and engineers to develop various PUF systems to increase the robustness and reliability of those systems and to overcome the problems associated with practical implementations.

PUFs have become more prevalent recently in numerous publications on hardware-intensive cryptographic protocols and schemes, such as PUF-based block ciphers [6] and PUF-based key transfer protocols [2]. A PUF can be realized in electronic products by integrating a customized PUF circuit, e.g., as part of a

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_1

chip or field-programmable gate array (FPGA) or as a stand-alone application-specific integrated circuit (ASIC). FPGAs are preferred for designing PUF primitives because of their reconfigurability, speed of turnaround, and flexibility and because they can easily interface with applications. Two of the top FPGA manufacturers, Xilinx [21] and Intel (formerly Altera) [10], have both revealed PUF implementations in their respective market products for security reasons. Applications of FPGA-based PUFs include device authentication [1, 16], IP protection [15], IoT security [1, 16], secure key generation [24], and IC counterfeit detection [32]. In this chapter, we demonstrate how to implement a ring oscillator (RO)-based PUF instance and analyze it on an FPGA device. In particular, this chapter can help a reader to better understand and will gain hands-on experiences on how a typical ring oscillator (RO)-based PUF primitive can be implemented at RTL (register-transfer level) and collect challenge-response pairs (CRPs) and also learn how to apply appropriate constraints to fix the specific PUF structure for effectiveness and calculate metrics for evaluation.

The remaining chapters are structured as follows: Basic information on PUFs and FPGA is briefly discussed in Sect. 1.2. Section 1.3 briefly provides the important PUF quality metrics. Section 1.4 presents the specifics of the RO PUF implementation. The experimental investigation of the RO PUF and a discussion of the implementation's outcomes are provided in Sect. 1.5. The conclusions are then presented in Sect. 1.6

1.2 Background

PUF is a security primitive that generates a device fingerprint using the uncontrollable manufacturing process variances induced during chip fabrication [25, 33]. A PUF can be properly described mathematically as a function ($R = f(C)$) that maps challenges/inputs (C) to responses/outputs (R) (i.e., (R) is generated by the PUF (Fig. 1.1a)). The essential security-oriented features of PUFs are shown in (Fig. 1.1b–g) [17], and we will explain these features briefly below:

Reproducible The response R generated from the same PUF should always be the same during multiple evaluations when using the same challenges (Fig. 1.1b).

Unique In the same challenge, responses produced by several PUFs must always be specific/unique (Fig. 1.1c).

Unclonable Given only f, it is (almost) impossible to build a g (i.e., another physical entity) such that $f(C) \approx g(C)$ (Fig. 1.1d).

One-Way Given R and f, it is difficult to find C such that $f(C) = R$ (Fig. 1.1e).

Unpredictable Given a set of CRPs Q, where $R = f(C)$ and $(C, R) \in Q$, it is (almost) impossible to compute the response $R' = f(C')$, where C' is a random challenge and $(C', R') \notin Q$ (Fig. 1.1f).

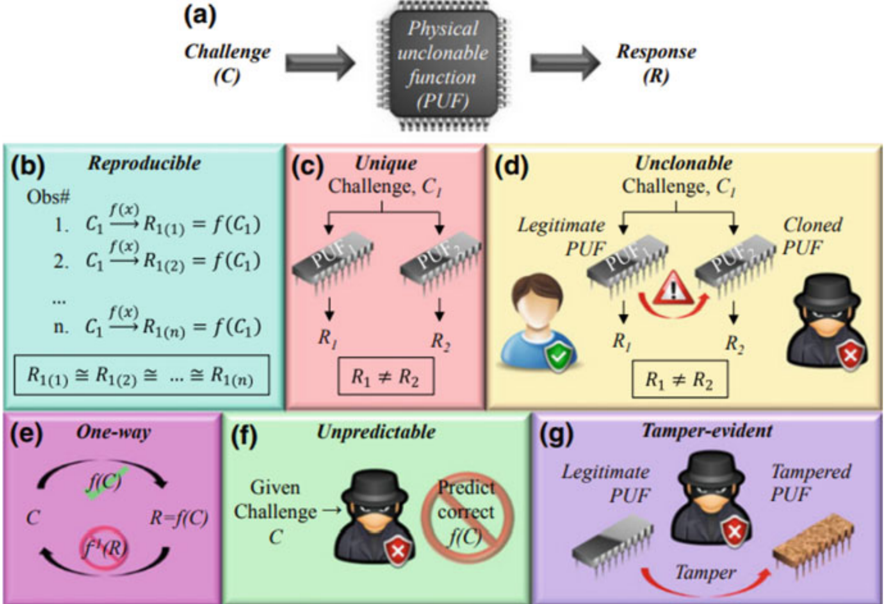


Fig. 1.1 Schematic representations of important features of PUF [20]

Tamper-Evident Physically altering of the physical copy/entity embedding f changes it to f' implying that $f' \neq f$ (Fig. 1.1g).

In practice, two main types of silicon PUF circuits have been implemented, namely, delay-based PUFs and memory-based PUFs according to various sources of manufacturing process variations. Delay-based PUFs provide the PUF response by using the propagation delay between identical circuits. Examples of delay-based PUFs are Arbiter PUF, RO PUF [24], etc. Memory-based PUFs are built around the instability of volatile memory cells. Examples of memory-based PUFs are Flip-Flop PUF [1], RS-LPUF [1], and SRAM-PUF [15]. The silicon PUF architectures can also be classified into two major categories according to the number of CRP space [3]: weak PUF and strong PUF. In general, weak PUFs are better suitable for applications like pseudorandom number generators (PRNG) and key generation since they have a relatively small number of CRPs that rise linearly with PUF size. Some examples of weak PUF are Flip-Flop PUF, RS-LPUF, SRAM-PUF, and RO PUF [3]. Alternative strong PUF architectures have a large set of CRPs that grow exponentially with PUF size and can be used directly for device authentication without the need for additional cryptographic mechanism. Some strong PUF examples are Bistable Ring PUF and Arbiter PUF [3].

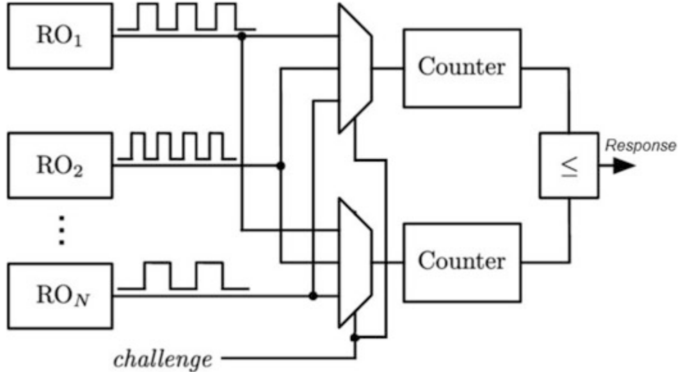


Fig. 1.2 Schematic diagram of RO PUF

1.2.1 RO PUF

Gassend et al. [11] originally proposed a concept of RO PUF based on a single configurable oscillator. Later, Suh and Devadas [24] in 2007 improved the RO PUF design which employs numerous identical ring oscillators (ROs) and takes into account oscillator pair relative frequencies rather than absolute values. The RO PUF architecture is shown in Fig. 1.2. The oscillation frequencies of any group of identically activated ROs, where each ring oscillator consists of an odd number of inverters, are used to calculate the RO PUF response. Despite having identical structures, each RO has slightly different oscillation frequencies because of random variations during the manufacturing process. The frequency is measured using rising edge counters. The counters count the number of edges of two oscillating signals over a predetermined amount of time, and the counters send their counter values to the comparator. Then, the two frequency counter values are compared, and depending on which counter has the higher value, a bit 0 or 1 is produced in response to this RO pair [4].

1.2.2 MiniZed Board Introduction

MiniZed board is shown in Fig. 1.3. It features a Zynq single-core SoC device (XC7Z007S), a low-cost prototype platform, and multiple storage devices including Micron 8 GB eMMC, Micron 512 MB DDR3L, and Micron 128 Mb Quad SPI NOR flash. In this work, we focus on its programmable hardware part, i.e., FPGA fabric for our development.

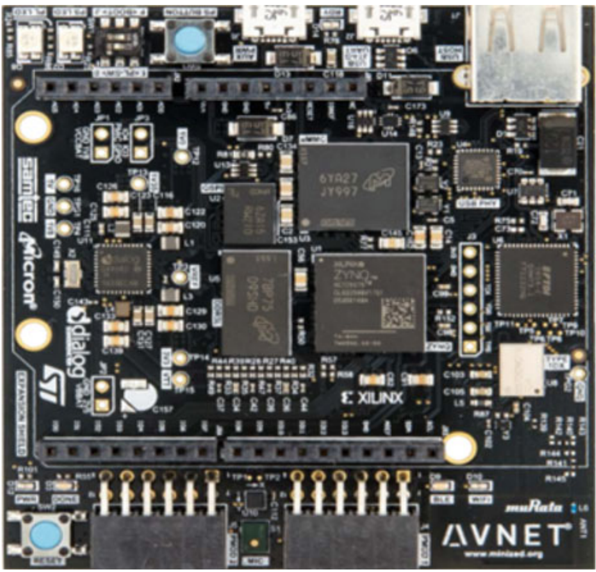


Fig. 1.3 MiniZed board

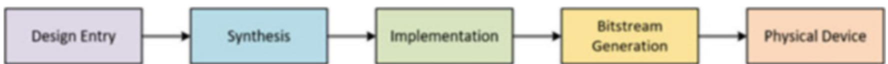


Fig. 1.4 FPGA development procedure

1.2.3 FPGA Development Procedure

FPGA is an integrated circuit that is flexible and can be configured by the user after the manufacturing phase. The typical development flow of an FPGA device involves design entry, synthesis, implementation, and bitstream generation as shown in Fig. 1.4.

Design entry can be done in various ways. The most intuitive method is drawing the schematics by connecting some predefined functional modules together. It is better and recommended to write your behavioral implementation in the form of hardware description language (HDL) like VHDL and Verilog. During the synthesis stage, the HDL code composed at the design entry stage will be converted into a circuit in the form of a netlist by the electronic design automation (EDA) tools. Then, this HDL code is going to be parsed to check syntax and then optimized to reduce redundant logic according to the specified settings. The generated netlist will contain the needed logic elements and the connectivity among them as described by the HDL code. The implementation phase will then technology map the logic elements in the netlist to the primitives available in the selected FPGA model so that the design could be implemented on your physical chip. Also, this step will place and route the primitives on the FPGA layout virtually per the constraints

from designers and physical aspects to make the final design meet the power, performance, and area requirements. Finally, the placed and routed netlist will be translated to the binary configuration data, the so-called bitstream with the vendor-specific tool, and then download to the target device to fulfill the functionality.

1.3 PUF Performance Metrics

To assess the performance of PUF circuits, several quality metrics, including *reliability*, *uniformity*, and *uniqueness* [1], have been presented [1]. These three metrics are commonly used to estimate PUFs and are also used in this work. Later, we also give some additional crucial metrics such as the entropy estimation and NIST test in Sect. 1.5:

(1) *Reliability (RE):* This metric captures how efficient a PUF is in perfectly reproducing its response bits under noise and environmental variation. For the given challenge, the PUF should give the same response under varying operating conditions. The average intra-chip HD is calculated using (1.1), and the *reliability* of a PUF chip is specified in (1.2):

$$HD_{INTRAi} = \frac{1}{m}\sum_{t=1}^{m}\frac{\mathsf{HD}(S_i, S_{i,t})}{n} \times 100\% \tag{1.1}$$

$$Reliability_i = 100\% - HD_{INTRAi} \tag{1.2}$$

where n is the length of the PUF response's sequence and m is the number of tests. S_i is taken at nominal operating conditions, and $S_{i,t}$ is measured under various operating conditions. The ideal value for *reliability* is 100% (i.e., the ideal value of HD_{INTRA} is 0%), and the *average reliability* of m chips can be estimated using the following Eq. (1.3):

$$Average\ Reliability = \frac{1}{m}\sum_{i=1}^{m} Reliability_i \tag{1.3}$$

(2) *Uniformity (UF):* It estimates how uniform the proportion of 1s and 0s is in the PUF response bits. Uniformity reflects the randomness of the response bit and is measured as a percentage of the response bit's Hamming weight (HW) according to the equation in (1.4)). 50% is the best value for *uniformity*:

$$Uniformity_i = \frac{1}{n}\sum_{j=1}^{n} u_{i,j} \times 100\% \tag{1.4}$$

where $u_{i,j}$ is the jth bit of n-bit response of ith chip.

(3) *Uniqueness (UQ):* This statistic assesses the difference of a PUF's responses to the same challenge (C) when implemented on several PUF chips (k). The uniqueness (HD_{INTER}) is calculated as the inter-chip variation of various responses using Eq. (1.5). 50% is the best value for *uniqueness*:

$$Uniqueness = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{\mathsf{HD}(S_i, S_j)}{n} \times 100\% \tag{1.5}$$

where S_i is n-bit responses of ith chip, S_j is n-bit responses of jth chip, k is the quantity of PUF chips, and $\mathsf{HD}(S_i, S_j)$ is the Hamming distance between n bit responses S_i and S_j.

PUF structure evaluation metrics may vary depending on the context of the application [9]. For instance, the metrics of uniformity, reliability, and uniqueness may have different importance in different PUF usage scenarios, such as encryption, authentication, or identification. For further information, see below:

- Identification: PUF can generate an identification number through challenge-response pairs (CRP) to identify genuine and counterfeit products. In this case, uniqueness is the most important factor. Furthermore, in identifying scenarios, reliability is not a big consideration when the bit error rate is low.
- Encryption: The PUF instance can be used to generate a random nonce that can be used to select specific public-private key pairs for asymmetric encryption and to generate a secret key for symmetric encryption techniques. Reliability, randomness, and uniqueness are important in such applications. However, a large challenge-response pairs (CRPs) space is not important in some situations where only a few keys need to be generated during the lifetime of the chip. The BER should be zero in this case, which would necessitate error correction [3].
- Authentication: PUFs are also well recognized for being frequently used to safely identify the chip in which they are inserted. The most important criteria, in this case, are randomness and reliability. The PUF should also have strong uniqueness characteristics. A very large CRP space is necessary to further deter attackers from reading all the responses and making a copy. Machine learning attacks against SCA adversaries are likewise prevented by the large CRP space.

1.4 Implementation Details of the RO PUF

In this part, we will first walk through the detailed RO PUF implementation used in this experiment. We will then talk about how to compile the RTL design into FPGA bitstreams and download it into the target FPGA (MiniZed board here). The original implementation is from https://github.com/Crimsonninja/senior_design_puf where we select their serial PUF design in their repository which uses a measure of

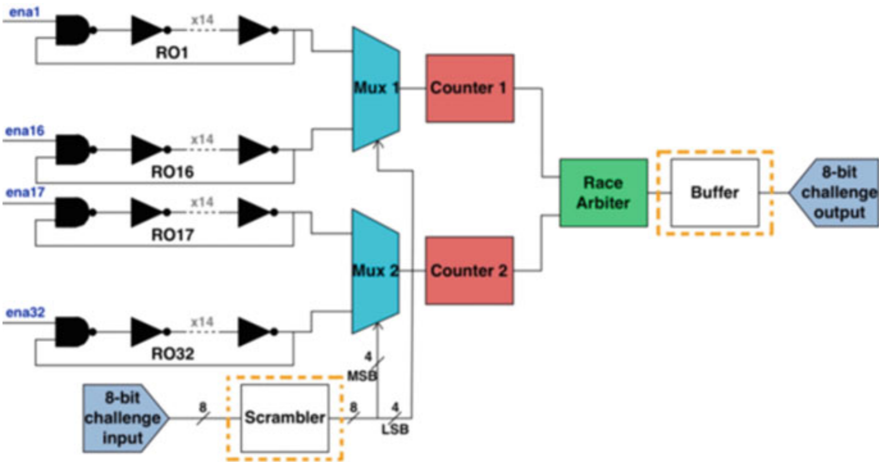


Fig. 1.5 Overall schematic of the serial RO PUF scheme

randomness, a nonlinear scrambler to increase internal entropy in the system as shown in Fig. 1.5.

An RO PUF consists of identical ROs in two multiplexers. The PUF challenge is given on the select lines of the multiplexer and selects which two ROs to compare. The multiplexer outputs are fed into a counter, each of which counts to a preset value. If either the upper or lower counter reaches a preset value, the arbiter outputs a "1" or a "0" depending on which counter ends first [4]. As shown in Fig. 1.6, the entire implementation consists of two parts, i.e., the RO PUF itself and the virtual IO (VIO) module. The VIO module here can provide the interface between the FPGA board and the host to apply the challenges and collect the PUF responses. Note that the wrapper looks very complicated, but their actual functionality is just simply providing a clock signal to the PUF implementation from the processor side since there is no available hardware clock source connected to the FPGA fabric. We mainly focus on the inst: top and its submodules.

Then, we need to synthesize, implement, and generate the consequent bitstream for the target FPGA (Fig. 1.7). Before loading the bitstream, the FPGA board must be connected by following the steps in Fig. 1.8. Then, we specify the desired bistream file and the debug probe file (.ltx) as shown in Fig. 1.9 which is used to set up the virtual IO interface for transferring the challenge input and accessing the response output.

After downloading the bitstream, in the Xilinx Vivado, you will see a dashboard called hardware vio pops up (otherwise, you might want to double click the hw_vio_1 in the Hardware window). At the first time, you might have to click the + button to add all available signals to the dashboard. Note that this RO PUF is designed to be an 8-bit challenge and response which is relatively small in the practical world if a large challenge-response pair space is wanted. Every time you apply a new challenge value in this window, the reset value should be set to be 1 and then 0 to

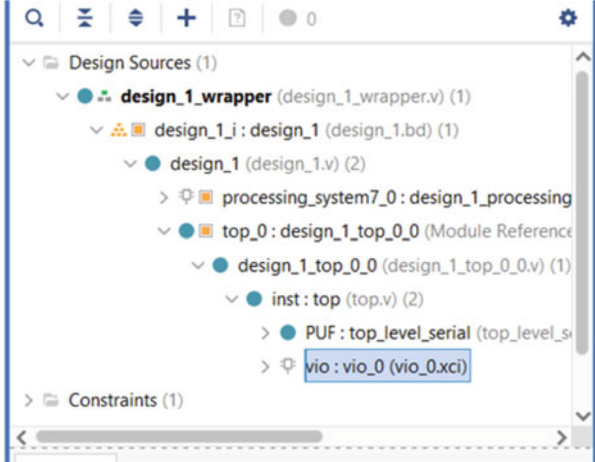


Fig. 1.6 Design hierarchy of the RO PUF implementation

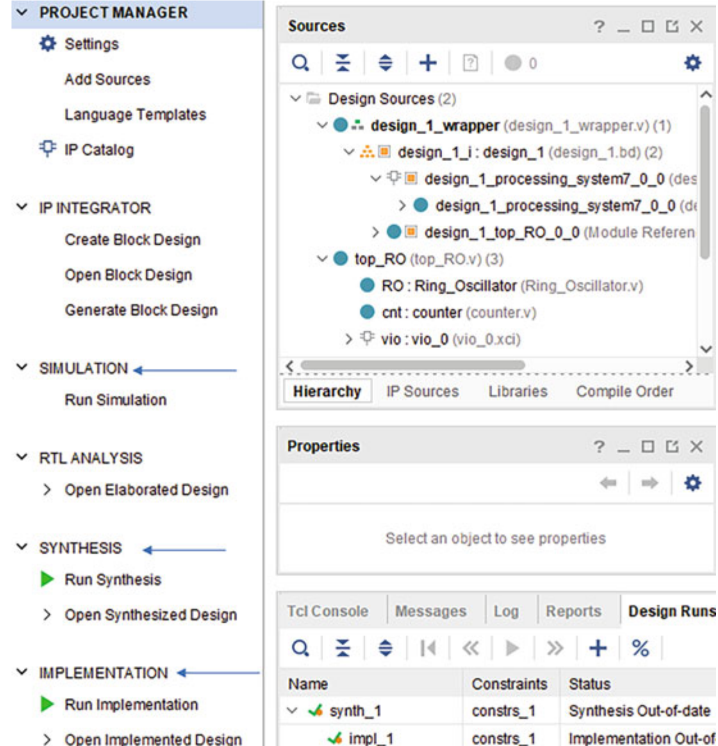


Fig. 1.7 Compiling the design in Xilinx vivado

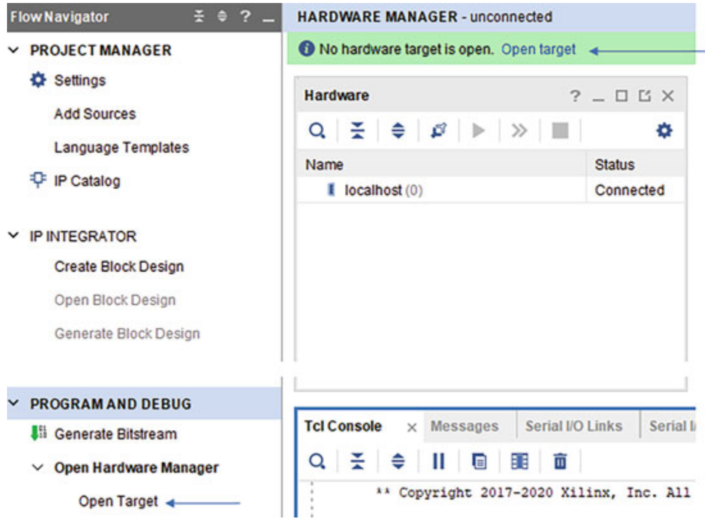


Fig. 1.8 Connect the MiniZed board

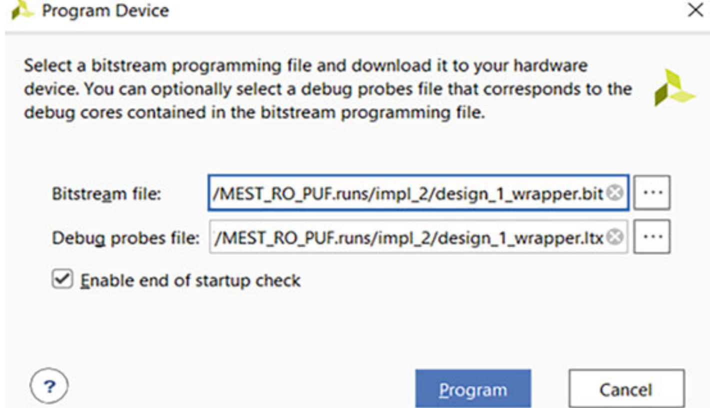


Fig. 1.9 Download bitstream

enforce the PUF design to be restarted to get the response. Due to the temperature and voltage variance, the response might not be exactly the same as the statistics in Fig. 1.10 but should be consistent at our side, i.e., one single challenge will not produce different responses in the particular environment.

Name	Value	Activ...	Directi...	VIO
design_1_i/top_0/inst/bnt	[B] 0		Output	hw_vio_1
design_1_i/top_0/inst/response[7:0]	[H] 74		Input	hw_vio_1
design_1_i/top_0/inst/challenge[7:0]	[H] 22		Output	hw_vio_1
design_1_i/top_0/inst/reset	[B] 0		Output	hw_vio_1
design_1_i/top_0/inst/enable	[B] 1		Output	hw_vio_1

Fig. 1.10 Inputs and outputs inside the hardware VIO dashboard

1.5 Performance Analysis and Discussion

In this section, we will learn about how to calculate the critical metrics for PUF performance evaluation. The number of SoC testbeds used in the literature to assess PUF performance varies widely. It has been demonstrated that good performance is possible for 10–50 [24, 34], for 5–10 [13, 31], and even for more than hundred [12, 18, 30] testbeds. In this chapter, the evaluation of PUF performance in terms of inter-chip hamming distance, uniformity, uniqueness, and reliability for the above PUF design has been carried out through implementations on XC7Z007S SoCs. The design files and all source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch1_puf/.

1.5.1 Randomness, Uniqueness, and Reliability

An important metric of PUF is the inter-chip hamming distance to measure the randomness between different PUF instances. On the same FPGA, we can create a new PUF instance by moving the original PUF module to a new location. As the PUF is mapped to a new region, physically new transistors, the new module will procure new process variations. Inter-chip variation can be calculated by applying the same challenge to the two PUF designs to see how the response can be changed. Ideally, half of the response bits from different PUFs should be flipped. The formulation is below:

$$Distance_\mathrm{intra} = \frac{(Hamming\ Distance(Response1, Response2))}{(Number\ of\ Response\ Bits)} \tag{1.6}$$

To move the PUF module, we apply different pblock constraints (pblock1 and pblock2 in Fig. 1.11). We can click the $P+$ button to create new pblocks and right-click the desired module (PUF here), floorplanning $->$ assign to pblock. Regenerate the bitstreams with the PUF instance in different locations. For example, for the same challenge 8'h66, the responses of pblock1 and pblock2 are 8'h53

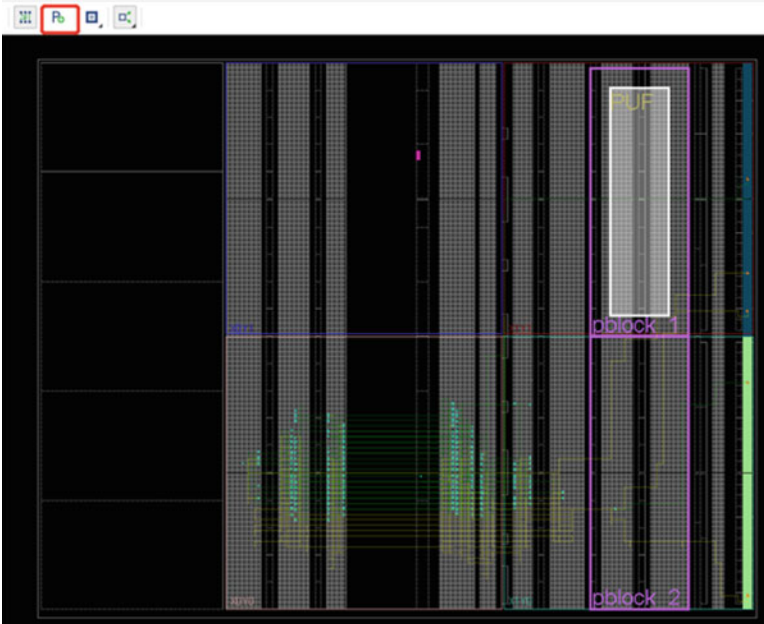


Fig. 1.11 pblock constraints in the device window

($8'b01010011$) and 8'h1A ($8'b00011010$) to produce a 37.5% inter-chip distance. Note that if we encounter the error LUTLP−1, please add the following statements in our xdc constraint file to suppress it:

```
set_property SEVERITY Warning
[get_drc_checks LUTLP-1]
set property ALLOW_COMBINATORIAL_OOPS TRUE
```

Before collecting CRPs, we need to open MATLAB to adjust the setting of the COM port. In this experiment, we use UART to transmit challenges and receive responses. Thus, the settings of the COM port need to be changed on different laptops. After running the MATLAB script, just like Fig. 1.12, the outcome of randomness, uniqueness, and reliability can be seen in the command window afterward.

For calculating the randomness, uniqueness, and reliability, the MATLAB script runs 30 times in total, and each time collects 1000 CRPs. The histogram of randomness, uniqueness, and reliability is listed in Fig. 1.13. The average randomness is 50.20%, and 30 results are presented in Fig. 1.13a. The average reliability is 01.60% and is presented in Fig. 1.13b. For calculating the uniqueness, we place aging-resilient PUF on four places of FPGA and use three FPGAs in total. Thirty comparisons between these PUFs are measured and the average uniqueness is 48.80%. The uniqueness of the PUF is presented in Fig. 1.13c.

Fig. 1.12 Experiment result of randomness, uniqueness, and reliability on the command window

1.5.2 NIST Statistical Test

In order to assess randomness, the derived PUF responses are also put to the test using the NIST 800-22 suite [7]. Long input bit sequences are needed for the majority of the NIST SP 800-22 statistical tests. Some tests, however, can be modified to examine a little amount of data. These tests include the following: the cumulative sum test, the block frequency test, the frequency test, the run test, the longest run test, and the approximation entropy test. In this chapter, we've utilized the six tests (taken from [19]) to verify the randomness of the RO PUFs (Fig. 1.5). A string must be at least 128 bits long to pass the NIST test suite's chosen subset. This evaluation's primary goal is to swiftly eliminate PUF responses that are not random. We combined all ten 128-bit responses to create a single 2560-bit string as the input sequences for these six tests. After that, the bit string is divided into ten 256-bit sequences that serve as the input sequences for the NIST tool (https://github.com/dj-on-github/sp800_22_tests). The distribution of p-values and pass rates for each test is the output of the NIST tool. We test the distribution of p-values at a significance threshold of 0.1. Because we selected the number of sequences $N = 10$, the tool determines the minimum pass rate of these tests, which in our case is 8/10. Since all six tests (based on p-values and pass rates) are passed by the aforementioned PUF implementation (Fig. 1.5), the derived PUF responses cannot be statistically separated from a true random source. Notably, passing these criteria does not ensure that the generated responses are unbiased. Instead, it serves as a sanity check, demonstrating that the statistical findings have no evident flaws.

1.5.3 Entropy Estimation

The use of entropy to evaluate the unpredictable nature of PUF responses has become a standard practice. The min-entropy measures the lower bound of entropy (i.e., worst-case scenario) so as to determine the unpredictability in the random data.

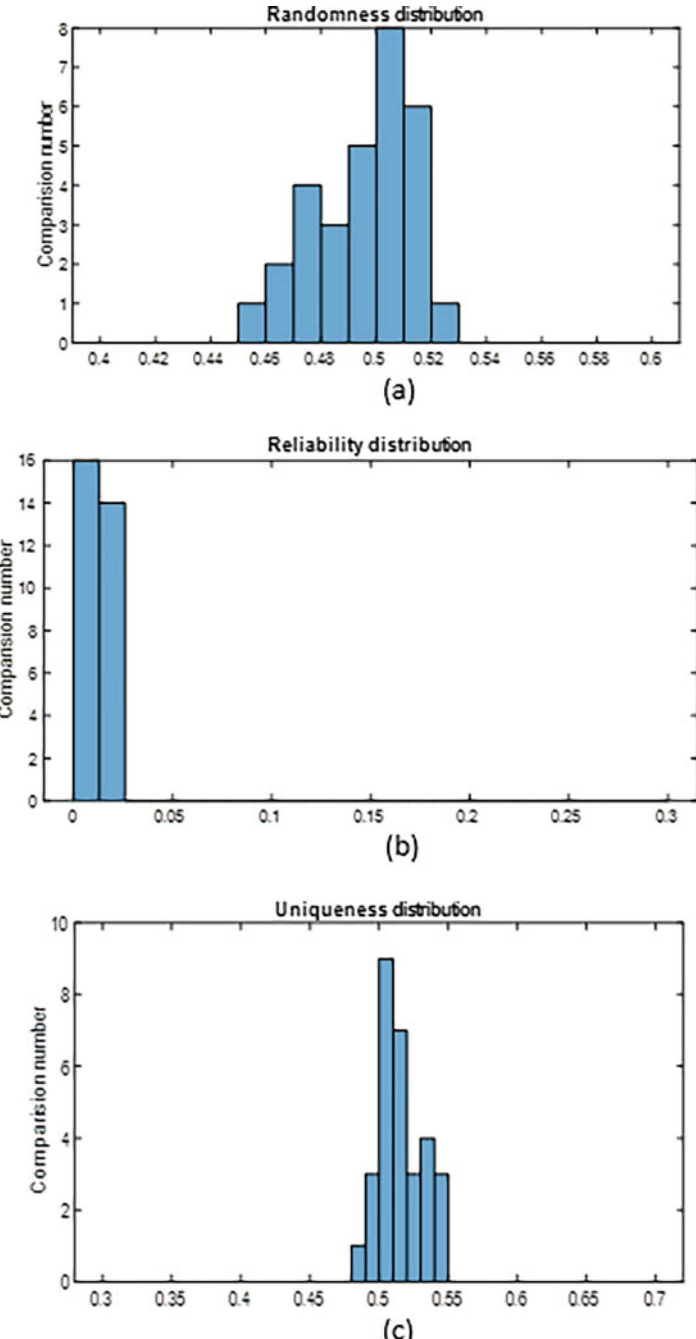


Fig. 1.13 Histogram of randomness, uniqueness, and reliability

To determine the minimum entropy of a binary source, we employ the procedure outlined in NIST specification 800–90 [29]. The occurrence probabilities are p_1 and p_0 for the values of "1" and "0," respectively, for each bit in the n-bit responses of the k devices. For $p_{i\ \max} = \max(p_1, p_0)$, the expression (1.7) provides the minimum entropy of each bit, whereas Eq. (1.9) provides the overall minimum entropy, as explained in [14]:

$$H_{\min,i} = -\log_2(p_{i\ \max}). \tag{1.7}$$

where

$$p_{i\ \max} = \begin{cases} \frac{HW_i}{k} & \text{if } HW_i > \frac{k}{2} \\ 1 - \frac{HW_i}{k} & \text{otherwise} \end{cases} \tag{1.8}$$

where $\frac{HW_i}{k}$ denotes the number of ones in k devices.

$$(H_{\min})_{average} = \frac{1}{n}\sum_{i=1}^{n} H_{\min,i} \tag{1.9}$$

By conducting experiments with ten FPGAs, we apply the expressions (1.7) and (1.9) to determine the minimum entropy for the designed PUFs. It is obvious that the PUFs in the aforementioned architecture attain an entropy close to 0.55. It is crucial to note that using more FPGAs will raise this entropy value, as shown by [14]. To determine the significant entropy value of PUF designs, a highly extensive investigation involving numerous boards is required. Despite the fact that the largest experiment ever conducted used more than 100 boards [12, 18, 30], it was not enough to reliably calculate the entropy. In general, how to accurately calculate the entropy of PUF responses is another significant open research problem.

1.6 Conclusion

PUFs are used in various security applications such as FPGA IP protection, device authentication, secret key generation, and trusted computing. The purpose of this chapter is to help readers (including researchers and practitioners) to learn about the fundamental knowledge of PUF technology, important performance metrics, and a typical ring oscillator-based PUF and FPGA development procedure. Also, readers will learn about how to apply challenges to the FPGA RO PUF and calculate the inter-chip distance, randomness, reliability, uniqueness, and entropy for performance evaluation.

References

1. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Design and analysis of FPGA based PUFs with enhanced performance for hardware-oriented security. ACM J. Emerg. Technol. Comput. Syst. **18**, 1–26 (2022)
2. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Field programmable gate array based elliptic curve Menezes-Qu-Vanstone key agreement protocol realization using physical unclonable function and true random number generator primitives. IET Circuits Devices Syst. **16**, 1–17 (2022)
3. Anandakumar, N.N., Hashmi, M.S., Tehranipoor, M.: FPGA-based physical unclonable functions: a comprehensive overview of theory and architectures. Integration **81**, 175–194 (2021)
4. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: Design, Implementation and analysis of efficient hardware-based security primitives. In: 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), pp. 198–199. IEEE, Piscataway (2020)
5. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd edn. Wiley, Hoboken (2008)
6. Armknecht, F., Maes, R., Sadeghi, A.R., Sunar, B., Tuyls, P.: Memory leakage-resilient encryption based on physically unclonable functions. In: Advances in Cryptology – ASIACRYPT 2009, pp. 685–702 (2009)
7. Bassham III, L. E., et al.: SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. National Institute of Standards & Technology, Gaithersburg (2010)
8. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Burlington (2018)
9. Che, W., Martinez-Ramon, M., Saqib, F., Plusquellic, J.: Delay model and machine learning exploration of a hardware-embedded delay PUF. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 153–158 (2018)
10. Gabriel, S.: Altera Partners with Intrinsic-ID to Develop World's Most Secure High-End FPGA (2015). https://www.intrinsic-id.com/altera-partners-with-intrinsic-id-to-develop-worlds-most-secure-high-end-fpga/
11. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: Silicon physical random functions. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, pp. 148–160. ACM, New York (2002)
12. Gu, C., Chang, C., Liu, W., Hanley, N., Miskelly, J., O'Neill, M.: A large-scale comprehensive evaluation of single-slice ring oscillator and PicoPUF bit cells on 28-nm Xilinx FPGAs. J. Cryptogr. Eng. **11**(3), 227–238 (2021)
13. Gu, C., Hanley, N., O'neill, M.: Improved reliability of FPGA-based PUF identification generator design. ACM Trans. Reconfig. Technol. Syst. **10**(3), 20:1–20:23 (2017)
14. Gu, C., Liu, W., Hanley, N., Hesselbarth, R., O'Neill, M.: A Theoretical model to link uniqueness and min-entropy for PUF evaluations. IEEE Trans. Comput. **68**(2), 287–293 (2019)
15. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Cryptographic Hardware and Embedded Systems - CHES 2007, vol. 4727, pp. 63–80. Springer, Berlin (2007)
16. Liu, W., Zhang, L., Zhang, Z., Gu, C., Wang, C., O'neill, M., Lombardi, F.: XOR-based low-cost reconfigurable PUFs for IoT security. ACM Trans. Embed. Comput. Syst. **18**(3), 25:1–25:21 (2019)
17. Maes, R., Verbauwhede, I.: Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions, pp. 3–37. Springer, Berlin (2010)
18. Maiti, A., Schaumont, P.: Improved ring oscillator PUF: an FPGA-friendly secure primitive. J. Cryptol. **24**, 375–397 (2011)
19. Marchand, C., Bossuet, L., Mureddu, U., Bochard, N., Cherkaoui, A., Fischer, V.: Implementation and characterization of a physical unclonable function for IoT: a case study with the TERO-PUF. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **37**(1), 97–109 (2018)

20. Noor, N., Silva, H.: Phase Change Memory for Physical Unclonable Functions, pp. 59–91. Springer, Singapore (2020)
21. Prophet, G.: Xilinx to Add PUF Security to Zynq Devices (2016). https://www.eenewseurope.com/news/xilinx-add-puf-security-zynq-devices-0
22. Ravikanth, P.S.: Physical one-way functions. PH.D. Thesis. Massachusetts Institute of Technology (2001)
23. Ray, S., Peeters, E., Tehranipoor, M.M., Bhunia, S.: System-on-chip platform security assurance: Architecture and validation. Proc. IEEE **106**(1), 21–37 (2017)
24. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th Design Automation Conference, DAC 2007, USA, June 4–8, 2007, pp. 9–14. IEEE, Piscataway (2007)
25. Tehranipoor, M.: Emerging Topics in Hardware Security. Springer, Berlin (2021)
26. Tehranipoor, M., Forte, D., Rose, G.S., Bhunia, S.: Security Opportunities in Nano Devices and Emerging Technologies. CRC Press, Boca Raton (2017)
27. Tehranipoor, M., Pundir, N., Vashistha, N., Farahmandi, F.: Hardware Security Primitives. Springer Nature, Berlin (2022)
28. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer Science & Business Media, Cham (2011)
29. Turan, M.S., Barker, E., Kelsey, J., McKay, K.A., Baish, M.L., Boyle, M.: Recommendation for the entropy sources used for random bit generation. NIST Special Publ. **800**(90B) (2018)
30. Wild, A., Becker, G.T., Güneysu, T.: A fair and comprehensive large-scale analysis of oscillation-based PUFs for FPGAs. In: 27th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–7. IEEE, Piscataway (2017)
31. Xu, X., Rührmair, U., Holcomb, D.E., Burleson, W.: Security evaluation and enhancement of bistable ring PUFs. In: Radio Frequency Identification, pp. 3–16. Springer, Berlin (2015)
32. Yang, K., Forte, D., Tehranipoor, M.M.: CDTA: a comprehensive solution for counterfeit detection, traceability, and authentication in the IoT supply chain. ACM Trans. Des. Autom. Electron. Syst. **22**(3), 42:1–42:31 (2017)
33. Yu, L., Wang, X., Rahman, F., Tehranipoor, M.: Interconnect-based PUF with signature uniqueness enhancement. IEEE Trans. Very Large Scale Integr. Syst. **28**(2), 339–352 (2019)
34. Zhang, J., Tan, X., Zhang, Y., Wang, W., Qin, Z.: Frequency Offset-based ring oscillator physical unclonable function. IEEE Trans. Multi-Scale Comput. Syst. **4**, 711–721 (2018)

Chapter 2
True Random Number Generator (TRNG)

2.1 Introduction

TRNGs are frequently employed in cryptographic applications such as random padding bits, key generation, and the generation of nonces in authentication protocols [35]. In addition to lottery drawings, true random number generators are also used in gambling, probabilistic algorithms, and computer games. The TRNGs must be unpredictable, produce truly random numbers using a physical source that is nondeterministic, and adhere to tight statistical constraints [36]. Generally speaking, a poor random number generator frequently causes the complexity of attacking a system utilizing that generator to reduce. For instance, a Mifare Classic tag's use of an unreliable pseudorandom number generator (PRNG) made attacks easier and gave attackers access to the secret key of the smart card [22]. Secure TRNGs can be used to generate the required random bits in cryptographic systems, which can solve these problems. In most cases, statistical test suites like NIST [6] and Diehard [21] are used to evaluate the randomness of a TRNG, and a stochastic model [31] is used to estimate the entropy of each bit.

Standard TRNGs employ a single source of entropy and a single post-processing step. The block diagram of a typical TRNG architecture is displayed in Fig. 2.1. Using a sampler, randomness is first collected from the physical noise source and then translated into a raw random bitstream (digitization). In practice, the raw random bitstream frequently demonstrates poor randomization. To enhance the output TRNG bit stream's quality and randomness, supplementary post-processing techniques like the Neumann corrector [37] or hash function [3, 17] are needed. In this regard, a variety of TRNG with post-processing designs have been presented using field-programmable gate arrays (FPGAs) [12, 15, 20, 33]. These designs draw entropy from the jitter of ring oscillators (RO) [2, 4, 12, 33] or the metastability of flip-flops [15, 20], which is brought on by setup or hold time violations of flip-flops (FFs). Numerous strategies for enhancing TRNG performance have been researched. In this chapter, we focus on true random numbers on FPGA. In

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_2

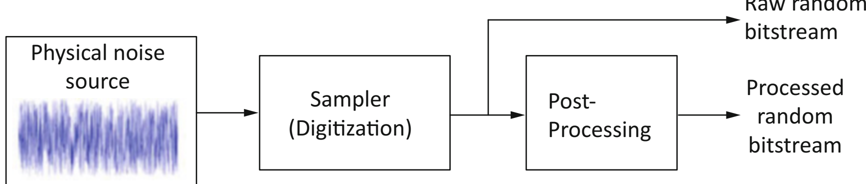


Fig. 2.1 An illustration of a true random number generator [4]

particular, this chapter can help a reader to better understand and will gain hands-on experience on how to create a technology-independent true random number generator (TI-TRNG) [26] step by step and get the random numbers on an FPGA board and also learn how to measure the quality of randomness of the generated true random numbers.

The remaining chapters are structured as follows: Sect. 2.2 briefly discusses commonly used sources of entropy and RO-based TRNGs. In Sect. 2.3, the implementation specifics of the RO-based TRNG are described. In Sect. 2.4, the quality of the bitstream generated by the TRNG is discussed. Section 2.5 presents the conclusions.

2.2 Background

2.2.1 Sources of Entropy

The entropy source that was employed in the design is a key resource for a TRNG. Timing jitter in circuits, thermal noise, metastability, chaotic circuits, quantum effects, and other factors are often employed sources of entropy. These are briefly discussed next:

(1) *Thermal Noise:* It is sometimes referred to as Johnson-Nyquist noise [23]. This is the electronic noise caused by the thermal movement of charge carriers in an electrical conductor that is at equilibrium. Regardless of whether an external voltage is provided or not, this agitation occurs. This kind of entropy generator is typically appropriate for use in ASICs. One of the family's well-known TRNGs was introduced by Intel [17]. This device uses a high-speed oscillator to first amplify and then digitize the Johnson thermal noise over a resistor. The statistical characteristics of the obtained random numbers are then improved via a von Neumann post-processing method. There are more recent TRNGs from this family in [11, 25].

(2) *Timing Jitter:* Timing jitter in electronic systems is the deviation of a signal's timing from its nominal value. For both FPGA and ASIC TRNGs, the timing jitter of phase-locked loops (PLLs) or free-running ring oscillators (ROs) is

frequently used as an entropy source. Fischer et al. [13] introduced PLL-based TRNG, while Allini et al. [1] showed the optimization of the PLL-based TRNG design. Fischer et al. [13]. Multiple RO-based TRNGs were first introduced in BSunar et al. [33] and later refined by Wold et al. in [41]. Multiple RO edges were used as the randomness source in the RO-based TRNG designs [44], which were then realized using a tetrahedral oscillator with high jitter [19]. Anandakumar et al. [4] incorporated the concept of programmable delay lines (PDL) to increase the randomness of the TRNG.

(3) *Metastability:* The most popular entropy source for both FPGA and ASIC TRNGs is metastability. Early systems used latches' metastable behavior as an entropy source [8]. In 2001, Walker et al. assessed the metastability of a DFF (D flip-flops) circuit for producing random numbers [39]. In [14], a method for generating random numbers that uses write collisions in memory blocks as a source of entropy was described. In [27], the noise source for ring oscillators is the last time they passed each other. In [20], truly random numbers were produced using the metastability of flip-flops. Programmable delay lines (PDLs), which precisely equalize the signal arrival timings to flip-flops, were used in [20] to accomplish the metastability. RS latches' metastable behavior was used by Hata et al. [15] as an entropy source to produce real random numbers. Li et al. [18] presented a TRNG that makes use of cross-coupled NAND gates.

(4) *Chaos Circuits:* A simple electronic circuit that displays typical chaotic behavior is referred to as a chaotic circuit. To produce noticeably different future outputs, they amplify modest changes at the beginning states. Typically, this kind of entropy source is appropriate for ASIC implementations. Yalcin et al. [43] presented a TRNG that used a double-scroll attractor. Rodriguez et al. [28] introduced a straightforward chaotic discrete-time systems-based TRNG. To produce chaotic signals, the suggested TRNG uses discrete maps, a straightforward mathematical model. A framework for analyzing chaotic-map TRNG performance is presented in [7]. For chaos-based TRNG, analog-to-digital converters (ADCs) are also employed. Callegari et al. in [10] make use of chaos and nonlinear signal processing to create a TRNG. Pareschi et al. in [24] proposed another TRNG design-based ADC that internally uses a pipeline ADC adapted to function as a collection of interleaved chaotic maps.

(5) *Quantum Effect:* As a source of randomness, several quantum effects can be used. John walker et al. [38] presented a TRNG based on radioactive decay as a source of entropy. A popular type of quantum random number generator (QRNGs) is based on the detection of a single photon between two outputs of a beam splitter. Lasers, single-photon emitters, and light-emitting diodes are common sources of photons. The TRNG designs in this family come in a variety of forms, from those seen in academic publications [32, 34] to those found in commercial products [30]. Jennewein et al. [16] demonstrated a physical QRNG with a throughput of 1Mbps. Recently, Massari et al. [42] developed a QRNG that makes use of a 16×16 array of detector pairs. It was decided

which detector in the pair of two would receive a photon first by implementing arbitrator modules.

2.2.2 Ring Oscillator-Based TRNG

As was already indicated, several RO-based TRNG designs have been published in the literature [2, 4, 12, 33, 41]. Jitter typically builds up in free-running ROs that have an odd number of inverters or delay elements coupled in a ring design [9]. This results in a change in the oscillator's digital output value over a period of about $2DL$, where $2L$ is the number of inverters in the oscillator and D is the delay of a single inverter. The rising and falling edges of the generated RO clocks exhibit jitter due to the oscillations' variable period, as demonstrated in Fig. 2.2. Digital circuits may experience these oscillations and jitter as a result of changes in the semiconductor noise, power supply, cross talk, propagation delays, and temperature. By sampling the output of a high-frequency oscillator using a D flip-flop (DFF)-based sampler, as shown in Fig. 2.3, these jitters can be exploited to generate a stream of truly random bits.

The use of numerous free-running ROs, such as BSunar et al. [33], can enhance the quality of the real random bits that are created. This is accomplished by feeding the outputs into a multi-input XOR tree, which is then sampled by a reference clock

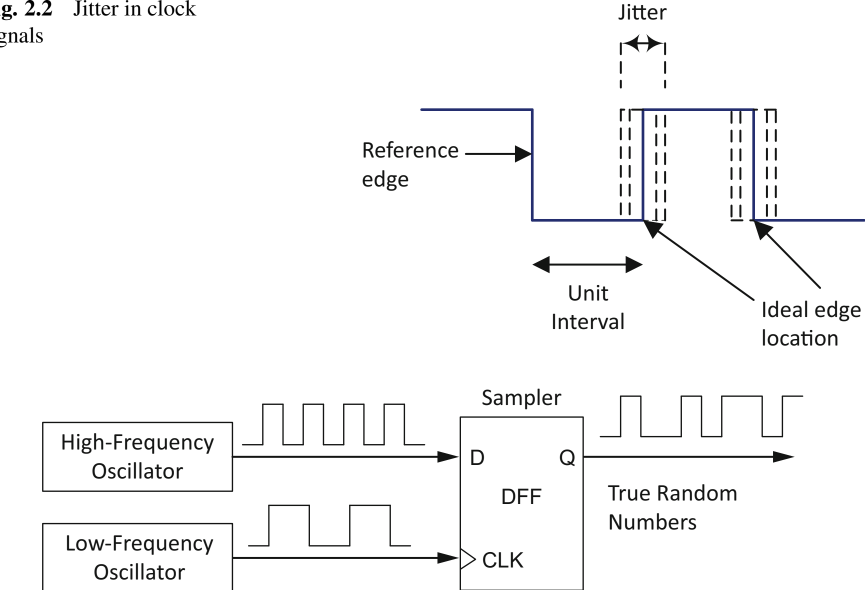


Fig. 2.2 Jitter in clock signals

Fig. 2.3 Basic TRNG based on oscillators

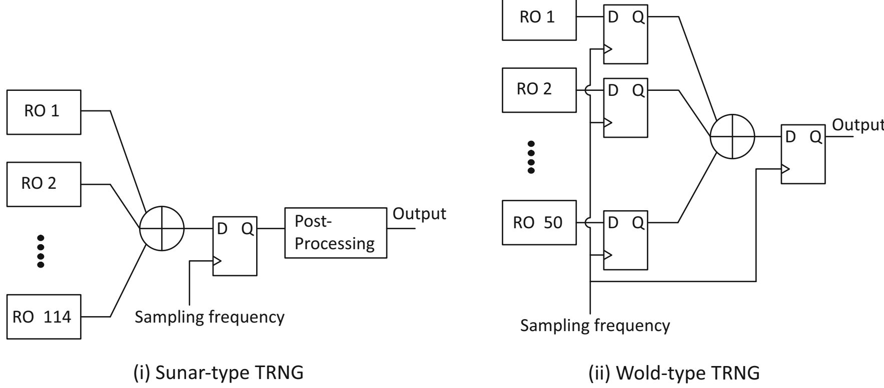


Fig. 2.4 Original TRNG block diagram (**a**) [33] and the modified TRNG (**b**) [41]

operating at a fixed frequency using a DFF to produce the random bit stream as illustrated in Fig. 2.4. However, handling a large amount of switching activity from the free-running ROs in such designs is exceedingly difficult for the XOR tree and the sample DFF [12]. Due to the numerous transitions that occur throughout a sample period as a result of parallel ROs, there are stringent setup and hold time requirements. By including a sampling DFF at the output of each free-running RO, as shown in Fig. 2.4, this feature can be somewhat addressed in [41]. This design uses fewer ROs and passes the NIST statistical tests without the need for post-processing.

2.3 RO-Based TRNG Implementation

This section will begin by going through the precise RO-based TRNG implementation that was used in the experiment. Then, we'll discuss how to download the target FPGA with the RTL design's FPGA bitstreams (Nexys A7-100T board here). In order to create and enhance the genuine random numbers on FPGA, we have used the basic implementation from [26] in this chapter. Technology-independent TRNG (TI-TRNG) design and bias detection algorithms were proposed by Rahman et al. [26]. The TI-TRNG utilizes a mechanism of self-calibration technique to lessen the biases in true random number generator (TRNG) output caused by aging and attacks and improves power supply noise for older technologies. Figure 2.5 shows how process modifications and environmental factors affect the output result of the SXOR, XOR tree, and bitstreams (R).

The jitter coming from RO is XOR-ed and sampled with the help of a synchronous flip-flop in an RO-based TRNG. Process variation (PV) increases the randomness of internal random noise. Power supply noise (PSN) increases jitter and subsequently randomness by acting like substrate noise. To raise the random

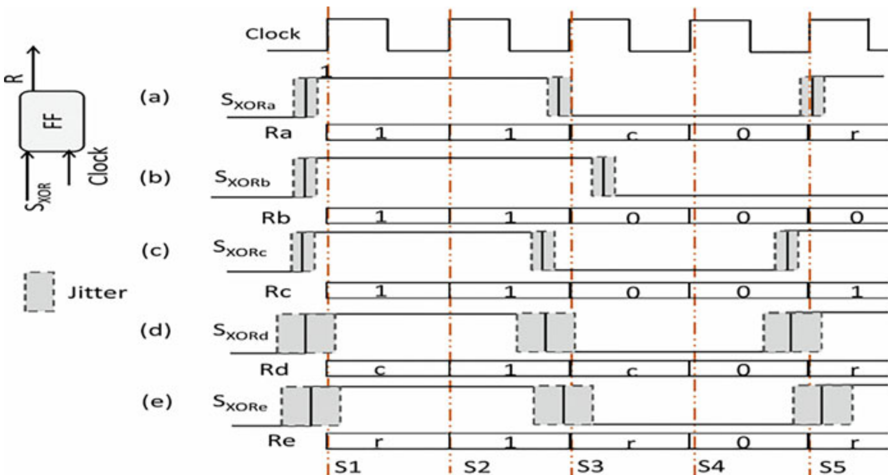
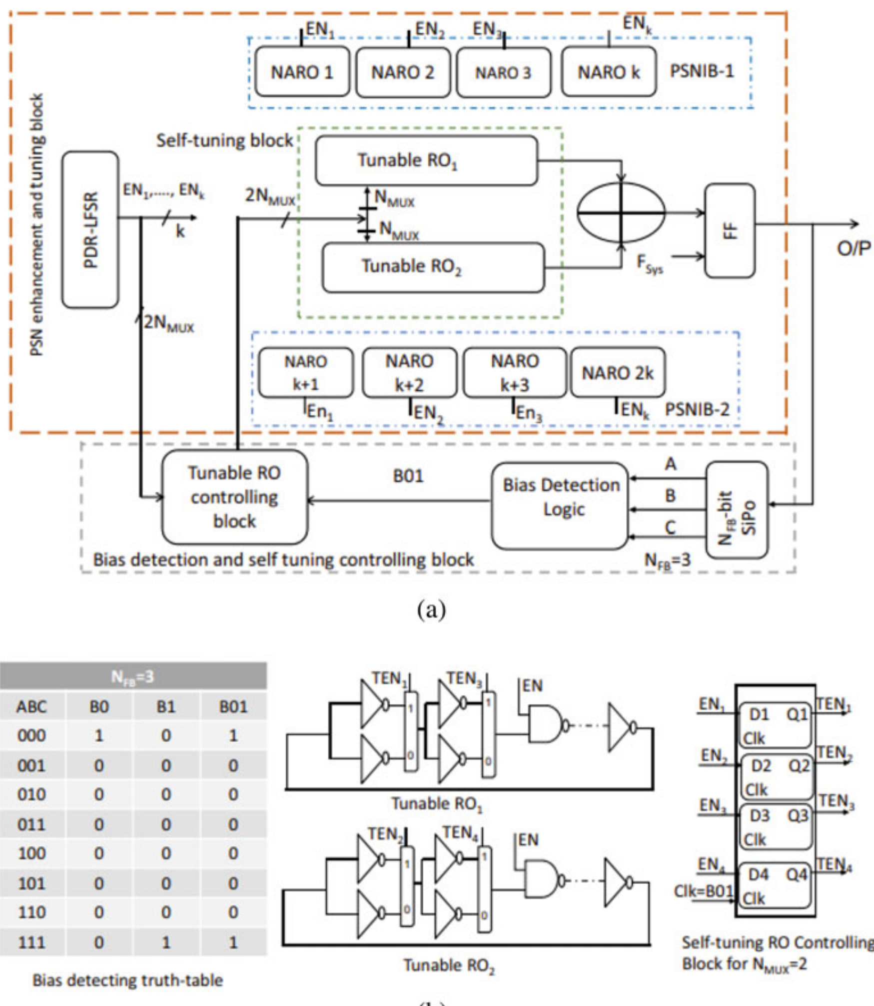


Fig. 2.5 Behavior of (**a**) traditional RO-based TRNG, (**b**) decrease of frequency because of environmental variations, and aging (**c**) increase of frequency because of operating conditions. Overcome bias by (**d**) expanding jitter and (**e**) adjusting RO delay (tunable RO)

PSN, we have introduced smaller ROs. The noise-augmenting ROs are these ROs (NAROs). The adjustable ROs are maintained near the NAROs. Two symmetric PSNIB-1 and PSNIB-2 modules with an equivalent combination of symmetric NAROs make up the random noise enhancement unit. It is vital to keep in mind that when similar ROs are utilized [40]. Even though RO-based TRNGs are exciting, they don't have a lot of randomness. ROs with equal length built in an FPGA can be correlated with one another because they have the same delays. This means that the XOR of their outputs mostly produces zeros. So, the randomness of the design isn't very good. To avoid interlocking, a tunable RO is proposed instead of the usual RO. Also, the mechanism of tuning can adjust the delay of the RO so that any bias is taken away. In this work, a self-calibration technique is used to keep a minimum security level when the entropy and randomness of a true random number generator (TRNG) are changed in bad working environments or an attack. This helps to get rid of or lower the costs of implementing post-processing.

The whole bias detection and calibration procedure to eliminate bias is illustrated in Fig. 2.6a and b. Figure 2.6b design illustrates how to use a NFB-bit series-to-parallel (SiPo) register to keep NFB successive bits from the randomized output of TRNG. The succeeding bits of NFB may be unbiased, biased to "0" or "1," or even both. A logic function, $B01$, is needed to identify the bias which depends on NFB as displayed in Fig. 2.6b. As an instance, $B01 = \overline{A}\ \overline{B}\ \overline{C} + ABC$ is used to determine the bias of the next three bits. The following NFB bits are biased to "0" or "1," respectively, if $B0 = 1$ or $B1 = 1$ when $NFB = 3$. The subsequent NFB bits are biased to either "0" or "1" when $B01 =$ "1." To prevent interlocking and significant speed variances between two tunable ROs, the detector of bias senses



$N_{FB}=3$			
ABC	B0	B1	B01
000	1	0	1
001	0	0	0
010	0	0	0
011	0	0	0
100	0	0	0
101	0	0	0
110	0	0	0
111	0	1	1

Fig. 2.6 (**a**) Architecture of TRNG and (**b**) the tuning and bias detection technique

the bias and modifies the path of delay between them. The tunable RO is shown in Fig. 2.6b, which is made up of inverters (of an odd number) with $NMUX$ MUXs in between some of them. The same $PD - LFSR$ that controls the $NAROs$ also controls the MUXs in the tunable RO. On the right side of Fig. 2.6b, readers can also see the controlling block that changes the delay of two tunable ROs to get rid of the bias. A simple tuning control unit is used to control the MUXs in the tunable RO. The controlling unit is made up of 2NMUX latches that are controlled by the bias detection unit's B01 signal. Through the tuning control unit, any 2NMUX bit from the $PD - LFSR$ output is passed. Each latch uses $B01$ as its clock. If its clock is

high ($B01$ = "1"), a latch is transparent. If its clock is low ($B01$ = "0"), it stays in the state it was in before. For $B01$ = "1," which means that bias has been found, the current state of the latches controls the MUXs, which change the delay path until the bias is removed. When there is no bias ($B01$ = "0"), latches stay in the state they were in before to keep a good TRNG bit stream. For more information, folks can look at [26].

We implement the base TRNG on the same Xilinx Artix-7 (Nexys-100T) FPGAs (xc7a100tcsg324-1). The 28-nm technology used to manufacture this FPGA makes it especially suitable for embedded applications. These designs were created using the VerilogHDL programming language and the Xilinx Vivado 2020.2 design suite. To install the Xilinx Vivado software on Windows 10 laptop or PC, use the following URL: https://www.youtube.com/watch?v=DIOll3P65hg. The entire implementation consists of two parts, i.e., the RO-based TRNG itself and virtual IO (VIO) module. Before downloading the TI-TRNG bitstream into the FPGA, we need to implement the TRNG design (as shown in Fig. 2.7) using Xilinx Vivado software. At the first time, you might have to click the + button (as shown in Fig. 2.7) to add all available signals to the dashboard. In the design source, we need to add different modules needed to generate the TRNG. For example, we shall need to include the different modules according to the TRNG architecture (as shown in Fig. 2.7) such as "TRNG," "Tunable RO control logic," "XOR," "Flip-Flop," "SIPO," "Bias Detector Logic," "LFSR," "PSNIB Blocks for NAROs," etc. The Verilog TRNG design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch2_trng/.

Since we intend to see the TRNG output by ILA probing on FPGA board, therefore, we have to add ILA ports in our design. To do this, click IP catalog under project manager as shown in Fig. 2.8. Then, type ILA in the search box and then select integrated logic analyzer. We assigned the design component name as ila_0. Next, we have selected two probing ports (see Fig. 2.8) and assigned a sample data depth size of 2048. Now, in the probe_ports window, select the probe width of

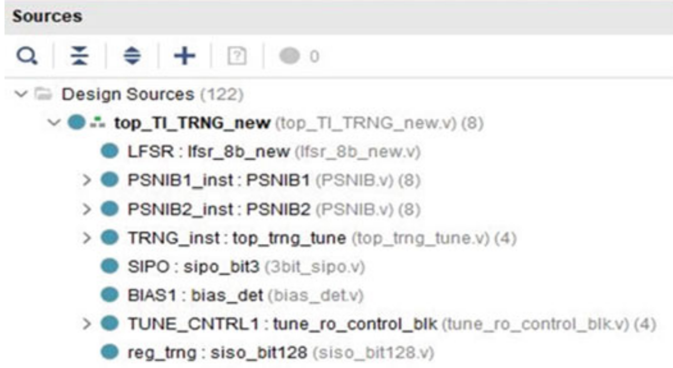


Fig. 2.7 Design hierarchy of the RO-based TRNG implementation

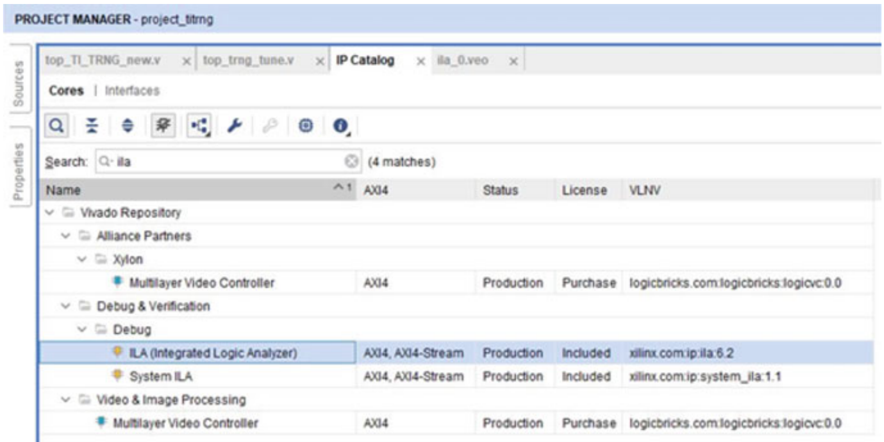


Fig. 2.8 ILA probing from IP catalog

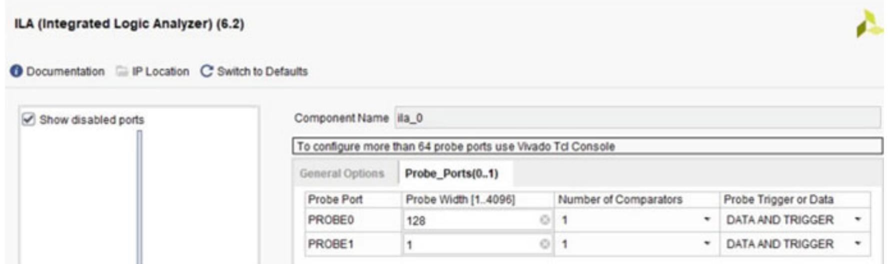


Fig. 2.9 ILA (Integrated Logic Analyzer)

Probe0 as 128 and the width of Probe1 as 1 (as shown in Fig. 2.9). Here, the data depth of Probe0 is 128. We will be observing the output of the 128-bit shift register which stores 128 bits from the TRNG output bit stream.

Then, we need to synthesize, implement, and create a .bit file for the target FPGA. For more details, one may refer to a tutorial on the FPGA implementation of digital systems [29]. Before loading the bitstream, the FPGA board must be connected to a laptop/PC through UART. Then, we specify the desired bistream file and the debug probe file (.ltx) which is used to set up the virtual IO interface for transferring the TRNG output. Now, we need to upload this bit file into the FPGA and finally observe the TRNG output. Once the bitstream is generated successfully, first connect the FPGA board with our PC/laptop, and then we can open hardware manager. Next, open target and click on "auto-connect" option. This will connect our target FPGA board with our PC/laptop. Next, click on the "program device" option. Now, a window (as shown in Fig. 2.10) will appear which will show which bitstream file will be uploaded to our FPGA board. Then select a bitstream programming file and download it to our target FPGA device.

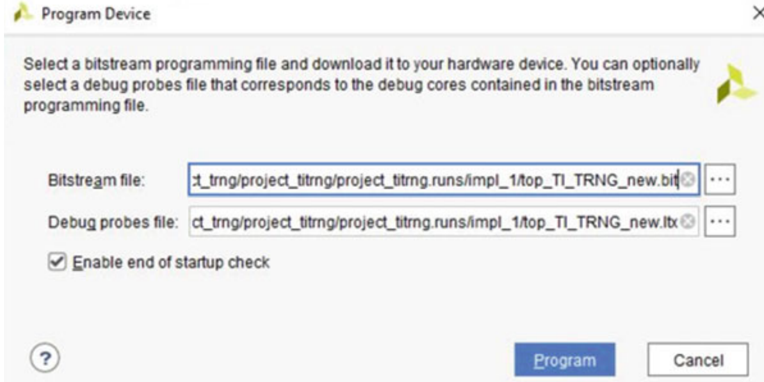


Fig. 2.10 Program device

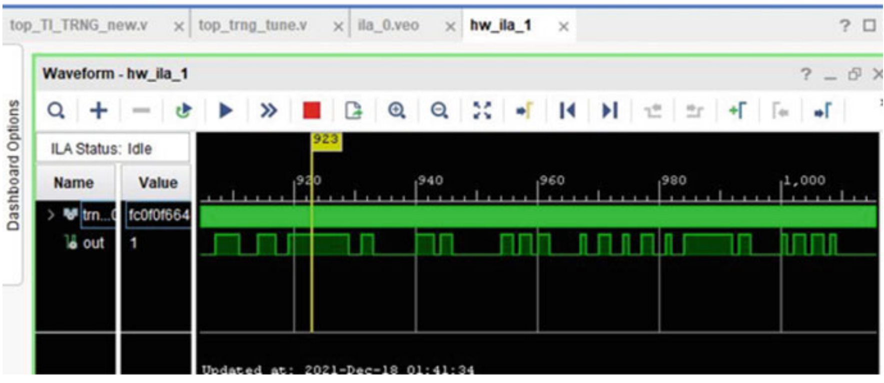


Fig. 2.11 TRNG output bitstream

After downloading the bitstream, in the Xilinx Vivado, we will see a dashboard called hardware vio pops up. The VIO module here can provide the interface between the FPGA board and then collect the TRNG output. Now, we can see TRNG output from ILA probing. First, we need to set the reset pin high and enable low. V10 switch (leftmost switch of the FPGA board) is defined as the reset pin, and V11 switch (second from the leftmost switch of the FPGA board) is defined as enable pin. Now, set V10 switch option to high (reset=high) and make the V11 switch low (enable=low). Note that both reset and enable pin cannot be high together. When we set reset pin to high, the TRNG out will be zero. Now, to enable the TRNG, make reset low and enable high. (Make V11 switch position low and V10 switch position high). The TRNG output bitstream is shown in Fig. 2.11.

2.4 Measures of the Quality of Randomness

We will discover how to assess the quality of randomness in the generated TRNG bitstreams in this section. The entropy test is used as the objective criterion of randomness in accordance with accepted industry practice. The restart test is then performed to demonstrate that the output, prior to post-processing, is unique after many system restarts under the same conditions. Finally, the NIST statistical test suite is used to evaluate the TRNG's bitstream quality.

2.4.1 Entropy Estimation

For a perfect true random number generator, the proportion of "0"s to "1"s is 0.5; hence, the predicted entropy per bit is 1. In this work, the test $T8$ of the procedure B of the AIS-31 [31], which is used to test raw random numbers, is used to determine the entropy rate for the generated numbers. We have downloaded the AIS31 test suite from the following URL: https://web.archive.org/web/20090228190713/www.bsi.bund.de/zertifiz/zert/interpr/ais_cc.htm. The instruction of the tool execution is quite clear and detailed, but the instruction is given in the German language. Therefore, we need to use the "Google translated" help info (obtained by clicking "jilfe anzeigen" on Fig. 2.12) for the English version.

When a sequence of length N is taken into consideration, test $T8$ divides the sequence into $Q + K$ disjoint L-bit words. $L = 8$, $Q = 2560$, and $K = 256{,}000$ are the recommended settings for the test $T8$. The recommended minimum sequence length for this test is 7,200,000 bits. For the test, we created 80 million random bits

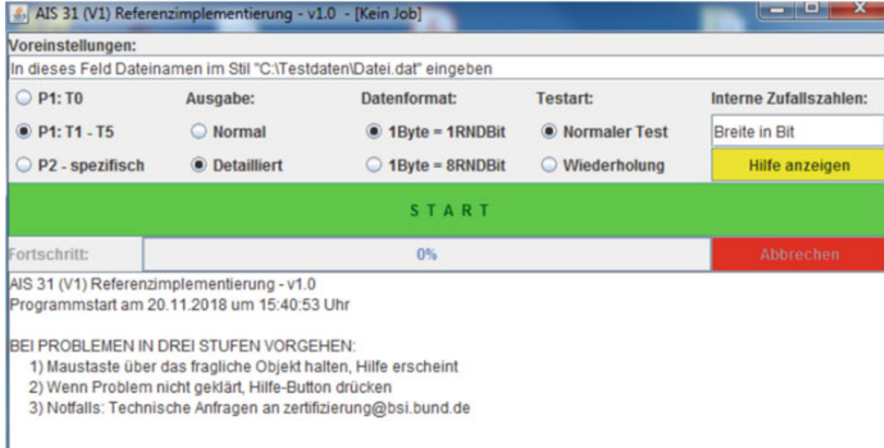


Fig. 2.12 AIS31 test suite

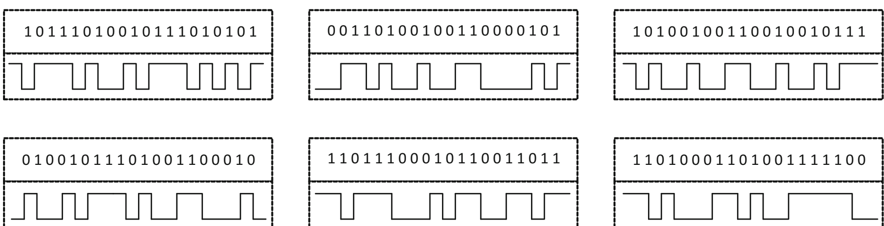


Fig. 2.13 Six output bitstreams were recorded after restarting the TRNG

using our test. The bit sequence passed the test and obtained 7.887 bits of entropy per byte.

2.4.2 Restart Experiment

For our TRNG design, a test was run to confirm the startup sequences for six restarts from identical beginning conditions. Similar restart techniques were employed by works [19, 41] and [12] to determine how much actual randomness was present in a pseudorandom oscillating signal. Following each restart, the first 20 sampled bits were observed and plotted as seen in Fig. 2.13. When the TRNG is frequently started from the exact same beginning conditions, the restart experiment results in graphs that are similar for a pseudorandom signal. However, in our situation, these were observed to differ each time, eliminating any chance of pseudorandomness [5].

2.4.3 Statistical Evaluation of the Output

NIST has developed a set of standardized randomness tests for assessing the quality of randomness in bitstreams [6]. The most complete publicly accessible tool is the NIST statistical test suite. In essence, 15 different types of tests are routinely run to evaluate the performance of TRNGs. The *frequency* test determines the proportion of ones and zeros for the bit sequence, and the bit sequence must be fairly evenly split between 0s and 1s in order to pass the test. Similarly, the percentage of 1s within M-bit blocks is determined by the *block frequency*. While the *longest run* test examines the lengthy strings of 1s within the M-bit block, the *runs* test determines whether the sequence of consecutive 1s and 0s is as predicted in a really random sequence. The *fast Fourier transform (FFT)* test examines the repeating patterns in the tested sequence that would suggest a departure from the assumption of randomness, whereas the *rank* test looks for linear dependence among fixed length sub-strings of the bit sequence. The goal of the *nonoverlapping template* test is to count the instances of predefined target strings. The *overlapping template* test is

comparable, except that when the pattern is discovered, the window only moves a little bit before the next search. The *linear complexity* test seeks to estimate the length of a linear feedback shift register and evaluate its complexity. The *universal statistical* test establishes the number of bits between matched patterns. The *serial* test measures the frequency of all possible overlapping m-bit patterns over the bit sequence. The *approximate entropy* test is similar to the *serial* test in that it examines overlapping blocks of two lengths. The *cumulative sum* test determines whether the 1s and 0s are evenly distributed across the whole sequence or if they are distributed in high quantities at the beginning or end of the sequence. The *random excursion* calculates the total number of times a specific state appears in a cumulative sum random walk, whereas the*random excursion variant test* calculates the number of cycles with exactly K visits.

The parameters for each test in this chapter were determined in accordance with NIST recommendations. Because it represents a 99% confidence interval, the default value for the significance level, $alpha$ was decided to be 0.01. Prior to and following post-processing, 1000 times (sequences) of 10^6 bits were collected in order to analyze the distribution of P-values (randomness measure). When the P-value is greater than 0.01, the distribution of the sequences is roughly uniform. In addition, a test is deemed successful if the permissible proportions fall within the range of 0.98056 and 0.99943 for $alpha$ = 0.01 and sequences = 1000, according to NIST recommendation [6]. The proportion of P-values should be higher than the 00.01 confidence interval. We have downloaded the NIST test suite from the following URL, https://github.com/dj-on-github/sp800_22_tests, and we have carried out 15 types of NIST tests to assess the performance of our TRNGs. Our TRNG implementation (Fig. 2.6) passed each of the 15 types of tests, and we were able to achieve the minimum pass rate (minimum proportion) for the tests of 0.982.

2.5 Conclusion

This chapter describes about the design, implementation, and analysis of RO-based TRNG implemented on the Artix 7 FPGA board using the Xilinx Vivado design suite. This chapter has offered a step-by-step breakdown of the whole implementation in the hopes that researchers who want to reproduce and study these findings may find it useful. We were succeeded in producing true random numbers. It successfully completes all NIST statistical tests and obtains a satisfactory entropy rate. By the end of this chapter, readers will understand how true random number generators work in theory; then how to design, implement, and generate TRNG on FPGA; and how to assess the TRNG output by using statistical tests.

References

1. Allini, E.N., Petura, O., Fischer, V., Bernard, F.: Optimization of the PLL configuration in a PLL-based TRNG design. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1265–1270 (2018)
2. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Field programmable gate array based elliptic curve Menezes-Qu-Vanstone key agreement protocol realization using Physical Unclonable Function and true random number generator primitives. IET Circuits Devices Syst. **16**, 382–398 (2022)
3. Anandakumar, N.N., Peyrin, T., Poschmann, A.: A very compact FPGA implementation of LED and PHOTON. In: International Conference on Cryptology in India, pp. 304–321. Springer, Berlin (2014)
4. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: FPGA-based true random number generation using programmable delays in oscillator-rings. IEEE Trans. Circuits Syst. II Express Briefs **67**(3), 570–574 (2020). https://doi.org/10.1109/TCSII.2019.2919891
5. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: Design, implementation and analysis of efficient hardware-based security primitives. In: 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), pp. 198–199. IEEE, Piscataway (2020)
6. Barker, E., Kelsey, J.: NIST Special Publication 800-90: Recommendation for random number generation using deterministic random bit generators (revised). Technical Report (2007)
7. Beirami, A., Nejati, H.: A framework for investigating the performance of chaotic-map truly random number generators. IEEE Trans. Circuits Syst. II Express Briefs **60**(7), 446–450 (2013)
8. Bellido, M.J., Acosta, A.J., Valencia, M., Barriga, A., Huertas, J.L.: Simple binary random number generator. Electron. Lett. **28**(7), 617–618 (1992)
9. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Burlington (2018)
10. Callegari, S., Rovatti, R., Setti, G.: Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. IEEE Trans. Signal Process. **53**(2), 793–805 (2005)
11. Chen, W., Che, W., Bi, Z., Wang, J., Yan, N., Tan, X., Wang, J., Min, H., Tan, J.: A 1.04 μW truly random number generator for gen2 RFID tag. In: 2009 IEEE Asian Solid-State Circuits Conference, pp. 117–120 (2009)
12. Dichtl, M., Golić, J.D.: High-speed true random number generation with logic gates only. In: Cryptographic Hardware and Embedded Systems - CHES 2007, pp. 45–62. Springer, Berlin (2007)
13. Fischer, V., Drutarovský, M.: True random number generator embedded in reconfigurable hardware. In: Cryptographic Hardware and Embedded Systems - CHES 2002, pp. 415–430. Springer, Berlin (2003)
14. Güneysu, T.: True random number generation in block memories of reconfigurable devices. In: 2010 International Conference on Field-Programmable Technology, pp. 200–207 (2010)
15. Hata, H., Ichikawa, S.: FPGA implementation of metastability-based true random number generator. IEICE Trans. Inform. Syst. **E95.D**(2), 426–436 (2012)
16. Jennewein, T., Achleitner, U., Weihs, G., Weinfurter, H., Zeilinger, A.: A fast and compact quantum random number generator. Rev. Sci. Instrum. **71**(4), 1675–1680 (2000)
17. Jun, B., Kocher, P.: The Intel random number generator. Cryptography Research, San Francisco (1999)
18. Li, C., Wang, Q., Jiang, J., Guan, N.: A metastability-based true random number generator on FPGA. In: 2017 IEEE 12th International Conference on ASIC (ASICON), pp. 738–741 (2017)
19. Liu, D., Liu, Z., Li, L., Zou, X.: A low-cost low-power ring oscillator-based truly random number generator for encryption on smart cards. IEEE Trans. Circuits Syst. II Express Briefs **63**(6), 608–612 (2016)
20. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA-based true random number generation using circuit metastability with adaptive feedback control. In: Cryptographic Hardware and Embedded Systems - CHES 2011, vol. 6917, pp. 17–32. Springer, Berlin (2011)

21. Marsaglia, G.: Diehard: A Battery of Tests of Randomness (1996). http://stat.fsu.edu/geo/diehard.html
22. Nohl, K., Evans, D., Starbug, S., Plötz, H.: Reverse-engineering a cryptographic RFID Tag. In: Proceedings of the 17th Conference on Security Symposium, pp. 185–193. USENIX Association, Berkeley (2008)
23. Nyquist, H.: Thermal agitation of electric charge in conductors. Phys. Rev. **32**, 110–113 (1928)
24. Pareschi, F., Setti, G., Rovatti, R.: A fast chaos-based true random number generator for cryptographic applications. In: 2006 Proceedings of the 32nd European Solid-State Circuits Conference, pp. 130–133 (2006)
25. Petrie, C.S., Connelly, J.A.: A noise-based IC random number generator for applications in cryptography. IEEE Trans. Circuits Syst. I Fundam. Theory Appl. **47**(5), 615–621 (2000)
26. Rahman, M.T., Xiao, K., Forte, D., Zhang, X., Shi, J., Tehranipoor, M.: TI-TRNG: Technology independent true random number generator. In: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2014). https://doi.org/10.1145/2593069.2593236
27. Robson, S., Leung, B., Gong, G.: Truly random number generator based on a ring oscillator utilizing last passage time. IEEE Trans. Circuits Syst. II Express Briefs **61**(12), 937–941 (2014)
28. Rodriguez-Vazquez, A., Espejo-Meana, S., Huertas, J.L., Martin, J.D.: Analog building blocks for noise and truly random number generation in CMOS VLSI. In: ESSCIRC ’90: Sixteenth European Solid-State Circuits Conference, pp. 225–228 (1990)
29. Roy, S.: FPGA IMPLEMENTATION – Step By Step (2019)
30. SA., I.Q.: Quantis AIS 31 certified random number generator (RNG). Accessed October, 2019. https://www.idquantique.com/random-number-generation/products/quantis-ais-31/
31. Schindler, W., Killmann, W.: A proposal for: Functionality classes for random number generators (2011)
32. Stefanov, A., Gisin, N., Guinnard, O., Guinnard, L., Zbinden, H.: Optical quantum random number generator. J. Modern Opt. **47**(4), 595–598 (2000)
33. Sunar, B., Martin, W.J., Stinson, D.R.: A provably secure true random number generator with built-in tolerance to active attacks. IEEE Trans. Comput. **56**(1), 109–119 (2007)
34. Tehranipoor, M., Forte, D., Rose, G.S., Bhunia, S.: Security Opportunities in Nano Devices and Emerging Technologies. CRC Press, Boca Raton (2017)
35. Tehranipoor, M., Pundir, N., Vashistha, N., Farahmandi, F.: Hardware Security Primitives. Springer Nature, Cham (2022)
36. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer Science & Business Media, Cham (2011)
37. von Neumann, J.: Various techniques used in connection with random digits. In: Monte Carlo Method. National Bureau of Standards Applied Mathematics Series, vol. 12, pp. 36–38. U.S. Government Printing Office, Washington (1951)
38. Walker, J.: HotBits: Genuine Random Numbers, Generated by Radioactive Decay (2001). http://www.fourmilab.ch/hotbits
39. Walker, S., Foo, S.: Evaluating metastability in electronic circuits for random number generation. In: Proceedings IEEE Computer Society Workshop on VLSI 2001. Emerging Technologies for VLSI Systems, pp. 99–101 (2001)
40. Wold, K., Petrović, S.: Security properties of oscillator rings in true random number generators. In: 2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp. 145–150 (2012). https://doi.org/10.1109/DDECS.2012.6219041
41. Wold, K., Tan, C.H.: Analysis and enhancement of random number generator in FPGA based on oscillator rings. In: International Conference on Reconfigurable Computing and FPGAs, pp. 385–390 (2008)
42. Xu, H., Perenzoni, D., Tomasi, A., Massari, N.: A 16 $\times$ 16 pixel post-processing free quantum random number generator based on SPADs. IEEE Trans. Circuits Syst. II Express Briefs **65**(5), 627–631 (2018)
43. Yalcin, M.E., Suykens, J.A.K., Vandewalle, J.: True random bit generation from a double-scroll attractor. IEEE Trans. Circuits Syst. I Regular Papers **51**(7), 1395–1404 (2004)
44. Yang, K., Fick, D., Henry, M.B., Lee, Y., Blaauw, D., Sylvester, D.: 16.3 A 23Mb/s 23pJ/b fully synthesized true-random-number generator in 28nm and 65nm CMOS. In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 280–281 (2014)

Chapter 3
Recycled Chip Detection Using RO-Based Odometer

3.1 Introduction

With the prosperity of emerging industrial segments such as the Internet of Things (IoT), 5G, and artificial intelligence, semiconductor devices become ubiquitous and connected to constitute cyberspace covering everybody in modern society [11]. The rate of growth of the semiconductor market is mind-boggling over the past decades and is projected to continue from $573.44 billion in 2022 to $1380.79 billion in 2029 [34]. Despite the promising market forecast, counterfeit devices are compromising the integrity of the semiconductor supply chain [1, 2, 14, 17, 18, 38, 44]. Specifically, there are a variety of counterfeit types such as recycled, overproduced, and remarked components that are injected by adversaries, i.e., the untrusted entities in the convoluted supply chain [12]. Even worsen, the COVID-19 pandemic has drastically and negatively impacted the capacity of foundries and thus lifted the prices of most categories, inspiring counterfeiters to introduce more illegitimate devices [41]. The horizontal business model of most semiconductor companies benefits from waiving the prohibitively high cost of maintaining foundries but leads to trust issues in the supply chain. For example, original component manufacturers (OCMs), as the owner of IPs in their semiconductor devices, have very limited controllability during the lifecycle of the devices [32]. As such, untrusted distributors might intentionally alter the markings printed on the chip surface to falsify the device grade, e.g., from commercial grade to space grade even if the device itself does not have any resistance against the radiations. Overproduced devices are parts that rogue foundries illegitimately produce outside the contract [36]. Such devices typically did not get enough tests to rule out defective parts and might have serious reliability issues. Out of all counterfeit types, recycled chips are the majority occupying more than 80% of all illegitimate devices [9]; they are derived from improperly disposed printed circuit boards (PCBs) or systems and sold as new items in the supply chain to acquire unfair profits. Due to the previous excessive usage, recycled chips might

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_3

exhibit lower performance and shorter lifetime which can result in catastrophic aftermaths once they reside in mission-critical systems.

Given the threats from counterfeit devices, it is imperative to have solutions to differentiate them from the legitimate parts [7]. However, existing electrical testing or high-precision inspection methodologies are either less effective or extremely expensive. The objective of this chapter is for readers to understand counterfeit threats comprehensively at first. Next, by focusing on the major counterfeit type, i.e., recycled devices, readers can gain hands-on experience in detecting them using a lightweight ring oscillator (RO) sensor. Readers can follow a step-by-step tutorial to implement the sensor on an FPGA board and use the on-chip debugging infrastructure to access the readings reflecting the aging extent of underlying transistors and indicating the usage time of the entire silicon.

The rest of this chapter is organized as follows: Section 3.2 details the background of recycled device detection. Section 3.3 focuses more on how to detect recycled FPGA devices because it serves as our prototyping platform. Section 3.4 briefly introduces the basic flow of FPGA development to prepare readers working on the configurable devices. Section 3.5 details the flow of how to utilize the RO sensor to measure the aging extent of the target FPGA to differentiate recycled devices. Section 3.6 concludes this chapter.

3.2 Background

3.2.1 Motivations and General Flow

Recycled chips are devices reclaimed from discarded boards and systems and sold as new ones. If integrated into mission-critical infrastructure, such devices would pose induced reliability and performance issues due to their previous excessive usage [45]. Our ring oscillator (RO) odometer is a golden reference-based mechanism to help end users differentiate recycled devices from the entire batch with the assistance of original component manufacturers (OCMs). The underlying mechanism flow is illustrated in Fig. 3.1 where the new device is embedded with our RO primitive by trusted OCMs. RO is a self-oscillated circuitry typically consisting of odd number of inverters [39]. The frequency of RO oscillation depends on the speed of low-level transistors and further relies on their aging extent. More specifically, the more usage a transistor experiences, the slower the transistor would be, and the RO frequency would thus be smaller. Therefore, RO-based primitive can be seen as an odometer measuring the infield work time intervals. Our RO primitive includes multiple paths of cascaded inverters; each path can be seen as a new RO instance and exhibit different frequencies [37]. OCMs register a new device by exhaustively putting down the RO frequency of each path and storing them in a secure database as a reference. Then the new device would enter the convoluted supply chain and might become recycled items. Suppose that an end

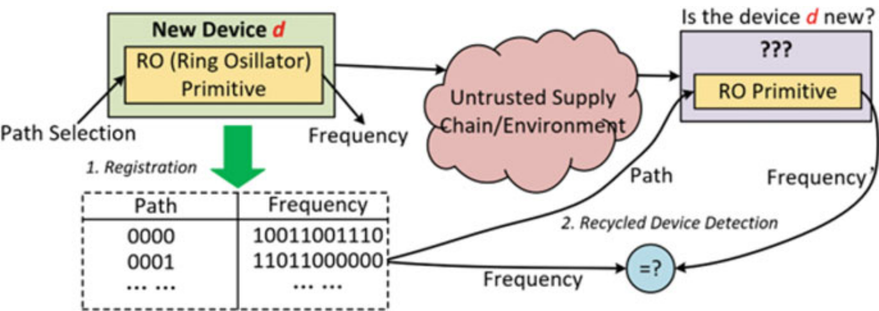


Fig. 3.1 Recycled IC detection using RO primitive

user really cares about the authenticity of their purchased devices because the final systems are reliability-critical. They can reuse the RO-based odometer again (the RO odometer is either embedded into ASICs or implemented as a binary bitstream provided by OCMs as shown in this manual) to regenerate the signatures, i.e., RO frequency. The frequency is going to be sent back to OCMs and matched against its reference counterpart in the secure database. If the two frequencies are close enough, it demonstrates the target device has not been extensively used. Otherwise, this item is suspected to be a recycled item and should not be integrated into mission-critical products [7].

3.2.2 Counterfeit Threats

IC counterfeiting has become a formidable challenge due to limitations in the existing test solutions and lack of available avoidance mechanisms [17, 30]. Over the past decades, a number of reports have claimed the serious counterfeiting issues in the worldwide microelectronic supply chain, e.g., a US Senate Armed Services public hearing was held dedicatedly on this issue and yielded the summary report clearly identifying counterfeiting as a crisis due to the significant impact on system reliability and security. With the ever-increasing complexity of electronic systems ranging from high-end data centers to compact mobile gadgets, they are for most fabricated and assembled in a globalized manner to minimize the time to market and cost. For instance, given the existing technical advancement and labor costs, offshore facilities typically can offer cheap and cheerful service to the fabless design house. Nevertheless, this opens the door to counterfeiting intrusions since illegitimate entities can gain unfair profits or competitive advantages by flooding the market with fake components where the outcome can be catastrophic if counterfeit ICs are integrated into mission-critical infrastructures such as defense and aerospace.

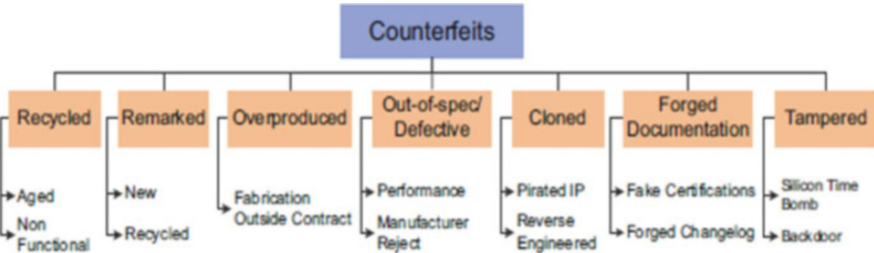


Fig. 3.2 Taxonomy of counterfeit types

There are a variety of counterfeit components as defined in [17] as follows:

- The parts which are unauthorized copies of their legitimate counterparts
- The parts with modified functionality and/or performance compared to the original specification
- The parts produced by unauthorized foundries or contractors
- The parts with defects or excessive previous usage
- The parts with forged markings on their surface or documentation

With the definitions, there are a few common instances in the real world that call for attention and countermeasures as depicted in Fig. 3.2:

- *Recycled*: Recycled devices are the most common counterfeiting components which are reclaimed from obsolete or discarded systems and then falsified as new chips, e.g., cleaning and polishing the surface. Typically, due to the aging phenomena from the excessive prior usage [16], such parts would exhibit much shorter lifetimes and lower speeds. Moreover, the recovery procedure usually needs high-temperature removal, aggressive physical removal, washing, sanding, and repackaging which might fundamentally pose negative impacts on the transistors or even make them completely nonfunctional. Besides compromising the revenue of the OCMs, the resultant degraded reliability renders the systems that incorporate recycled devices vulnerable.
- *Remarked*: Markings on-chip surfaces are the most direct and convenient manner for distributors and end users to identify the manufacturer, trademark, speed grade, and so on. However, the original marking can be easily chemically or physically removed by the adversaries. Next, the attackers can blacktop the surface to conceal the scratches during the marking removal and then reprint the new information. The motivation behind the counterfeiting threat is the huge profit gap between a normal device and a high-end part considering the similar looking except for the marking on the top. For example, a space-grade IC needs advanced expertise, deliberate designs, and huge investments for resilience against the radiations and thus always carries a high price tag. In contrast, its commercial grade can be much cheaper without the dedicated protection and redundant circuitry inside. If remarked commercial grade components are

integrated into the space applications, they would not be able to withstand the disturbances and behave unexpectedly.

- *Overproduced*: Globalization has pushed design houses to outsource their work to businesses throughout the world for fabrication and packaging, mostly to save production costs. When foundries and packaging corporations fabricate and sell components without having a contract with the design house (the owner of the component's intellectual property (IP)), overproduction occurs. Overproduced ICs may pose major reliability hazards because they do not experience the same exhaustive testing as genuine parts and may not fulfill the manufacturer's formal flow requirements.
- *Out-of-Spec/Defective*: If a component gives mistaken results during post-manufacturing tests, it is considered to be faulty. These components ought be either eliminated or demoted. However, there will be an unknown increase in the danger of failure if they are instead sold on the open (gray) markets, knowingly either by a non-trusted business or by a third party that has stolen them.
- *Cloned*: Cloning is frequently utilized by a variety of competitors and counterfeiters (from small businesses to huge corporations) to imitate a design and avoid the high development costs. Reverse engineering and fetching intellectual property (IP) illegally are two methods of cloning (also called IP theft). Cloning can also happen when someone with access to the part design transfers unlicensed knowledge to another individual.
- *Forged Documentation*: A component's accompanying documentation includes details on the specification, testing, certificates of conformance, statement of work, etc. A component can be misrepresented and sold even if it is nonconforming or defective by altering or fabricating this documentation. Since the OCM may not have archived records for earlier designs and older parts, it is frequently challenging to confirm the legitimacy of such documents. Legitimate documentation can also be duplicated and linked to components from a collection that doesn't match it.
- *Tampered*: Modifications to components can have disastrous effects on the systems that use them. As a silicon time bomb, altered chips, for instance, can suddenly "kill" their functioning at a crucial time. Additionally, tampered chips might have backdoors that provide adversary access to crucial system features or reveal private data.

3.3 Recycled FPGA Detection

The literature introduced a number of techniques for counterfeit detection such as aging detection sensors and PUF-based and hardware metering-based solutions [4]. We also give the different counterfeit detection techniques in Table 3.1. In this section, we go through how to use the RO sensor to determine how old the target FPGA is in order to distinguish between recycled devices. Configurable logic blocks

Table 3.1 Counterfeit detection techniques

Counterfeit detection types	Counterfeit detection techniques
Aging detection sensors	Statistical analysis [20]
	Timing warning aging sensor [22]
	Electromigration-based sensor [19]
	Electromigration-based sensor [35]
	Ring oscillator combating die [27]
	Fused CDIR [17]
	Pass logic [33]
	IDDT signature [46]
	Tampering [28]
	Image processing [6]
PUF	Arbiter PUF [3, 5]
	PUF authentication [4]
	Hardware watermarking [29]
	PUF obfuscation [4]
	BIST PUF [21]
Hardware metering	Combinational logic encryption [24]
	Optimization of combinational logic locking [8]
	Split manufacturing [23]
	Security analysis of logic encryption [43]
	Key interdependency [25]
	IC testing [31]
	Delay locking [42]
	Reverse engineering-based anti-counterfeiting [13]

are used by FPGAs to implement logic functions (CLBs). Lookup tables (LUTs) are fundamental components of an FPGA design since they serve as function generators in CLBs. Reconfiguring modern FPGAs enables changing the mapped function of LUTs. As a result, LUTs implement trillions of logic operations in total. Therefore, it's critical to comprehend LUT behavior in order to research aging deterioration for recycled FPGA identification. An SRAM bit that stores mapped values and a set of multiplexers that choose the bit that drives the LUT output are the typical components of a LUT. In this illustration, a 16:1 multiplexer and a 4-input LUT with 16 SRAM cells are used. The 16:1 multiplexer was constructed from a tree of 2:1 multiplexers. By putting the proper values in the SRAM cells and four-level hierarchical selectors (I0; I1; I2; I3), any logic function with four inputs can be implemented (Fig. 3.3).

The use of ring oscillators (ROs) is a popular technique for measuring delay variations in ICs [15]. In this endeavor, ROs will be developed to evaluate FPGA performance. A closed-loop chain made up of an odd number of inverting delay stages connected in series is known as a RO. An example is illustrated in Fig. 3.6. The oscillation period is equal to the sum of the propagation delays of all the loop's

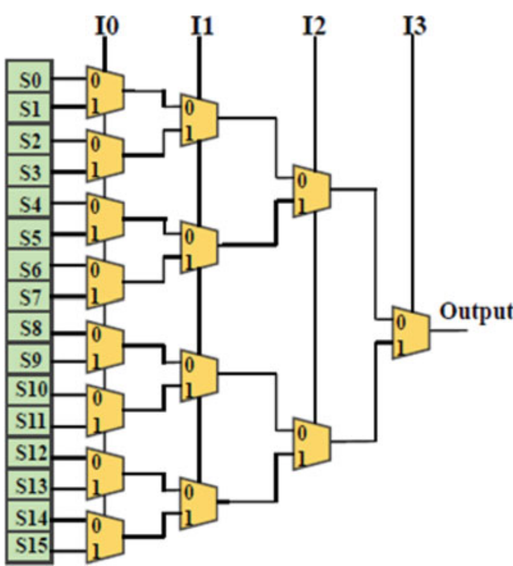


Fig. 3.3 Recycled IC detection sensor diagram

constituent parts. Using LUTs with an inverting stage implemented, ROs can be mapped on FPGAs. SRAM cells, selector transistors, and connection delay are all added together to create each stage's propagation delay. Over its lifetime, an operating FPGA becomes slower. The aging mechanisms include bias temperature instability (BTI) [26, 40], hot carrier injection (HCI) [26, 40], time-dependent dielectric breakdown (TDDB) [26] and electromigration [10]. Due to their enormous influence on the transistor switching speed, BTI and HCI are the main topics of this lab. This effect can be utilized to detect recycled FPGAs since it is quantifiable.

BTI and HCI alter the circuits' threshold voltage, which over time reduces the performance of FPGAs [2]. Selector circuits of LUTs get slower as the propagation delay of the BTI- and HCI-induced transistors rises. PMOS transistor speed is slowed by NBTI threshold deterioration, and SRAM cell static noise margin is dramatically decreased. Wire defects are brought on by the slow flow of ions, and TDDB from protracted low electric field exposure also reduces performance. The temperature has a significant impact on all modes of deterioration, with high temperatures deteriorating aging.

3.4 FPGA Development Procedure

A field-programmable gate array (FPGA) is an integrated circuit that is flexible and can be configured by the user or designer after the manufacturing phase. The typical development flow of an FPGA device involves design entry, synthesis, implementation, and bitstream generation as shown in Fig. 3.4.

Fig. 3.4 FPGA development procedure

Design entry can be done in various ways. The most intuitive method is drawing the schematics by connecting some predefined functional modules together. It is better and recommended to write your behavioral implementation in the form of hardware description language (HDL) like Verilog and VHDL. During the synthesis stage, the HDL code composed at the design entry stage will be converted into a circuit in the form of netlist by the electronic design automation (EDA) tools. Our HDL code is going to be parsed to check syntax and then optimized to reduce redundant logic according to the specified settings. The generated netlist will contain the needed logic elements and the connectivity among them as described by the HDL code. The implementation phase will then technology map the logic elements in the netlist to the primitives available in the selected FPGA model so that the design could be implemented on your physical chip. Also, this step will place and route the primitives on the FPGA layout virtually per the constraints from designers and physical aspects to make the final design meet the power, performance, and area requirements. Finally, the placed and routed netlist will be translated to the binary configuration data, the so-called bitstream with the vendor-specific tool, and then downloaded to the target device to fulfill the functionality.

3.5 Recycled Chip Detection Experiments

In this section, the effectiveness of the RO-based odometer is demonstrated through hands-on experiments on the FPGA platform. The odometer primitive can measure the usage time of the target silicon based on the switching speed of the underlying transistors, further being used as a solution for recycled semiconductor microelectronic detection. Specifically, the experimental FPGA platform is introduced at first. Then, the particular structure of the RO-based odometer is presented. Finally, the experimental flow of design mapping and recycled chip detection is illustrated step by step.

3.5.1 Experimental FPGA Platform

The experimental FPGA platform is a low-cost MiniZed board from Avnet (see Fig. 3.5) featuring a Zynq single-core system-on-chip (SoC) device XC7Z007S and multiple storage devices including Micron 512-MB DDR3L, Micron 128-Mb Quad SPI NOR flash, and Micron 8-GB eMMC. The device can be programmed by the host using a USB-to-JTAG interface. Here, the Zynq SoC is treated as a normal

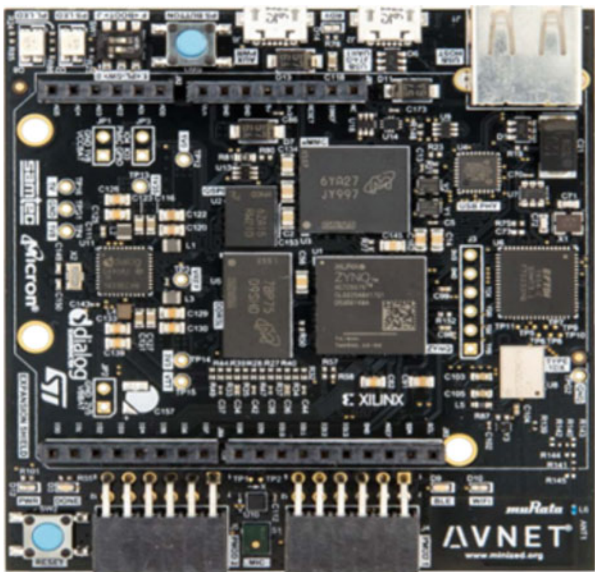


Fig. 3.5 MiniZed board

FPGA device where only its reconfigurable fabric is utilized for our experiments while the embedded ARM core is reserved for future advanced applications.

3.5.2 Experimental Flow

Why Can RO-Based Odometer Detect Recycled Devices? As mentioned above, recycled devices refer to the ICs reclaimed from the discarded PCBs and/or systems where the previous stress has drastically degraded the reliability and performance. However, conventional electrical testing or inspection techniques require intensive expertise and costly facility, making themselves infeasible options for common customers or OCMs. On the other hand, as an avoidance solution, the RO-based odometer measuring the aging/usage time of the target silicon essentially serves as a good foothold to thwart counterfeit threats; supposedly, new silicon that suffers from serious performance degradation as detected by the odometer would be rejected to be integrated into mission-critical infrastructure because of the authenticity concerns. Be more specific on commercial FPGA cases where dedicated analog RO-based odometers are mostly not available; before entering the convoluted supply chain, the new FPGA device will be configured with a RO primitive. There are different paths available in the RO primitive that can be selected to choose the desired RO path. Each RO will oscillate freely for a specified time period to measure the switching frequency. The frequency statistics would be collected and stored in the secure database as the reference characteristics of new devices (registration

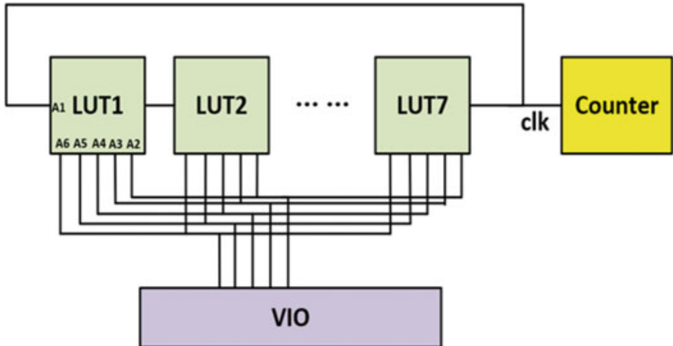


Fig. 3.6 Recycled IC detection sensor (RO-based odometer) diagram

phase). Given the fact that the transistor in an FPGA would gradually become slower due to the aging effects, one can differentiate the recycled devices by measuring the RO (mapped to the same intra-FPGA location as the one when measuring the reference frequency) frequency again since the new device would produce a very close frequency data to the enrolled one, whereas aged ones typically produce perceptibly smaller values.

What Is the Structure of RO-Based Odometer on FPGA? The detailed FPGA-based RO odometer implementation is presented in Fig. 3.6. This main component is the chain of the seven consecutive LUTs which is used to measure the path delay. These seven LUTs will be configured in a specific way to form a seven-stage RO. Since the aging procedure will degrade the device performance, the aged path will exhibit a larger delay which can be detected by this RO. In addition, we attach a virtual IO module (VIO as shown in Fig. 3.6) to control the other five inputs of every LUT in the RO chain. VIO module provides a very convenient interface for online debugging the FPGA (Fig. 3.7). We can deliver our inputs and collect outputs to/from the target implementation easily with the module. In this way, the user can select the arbitrary path in the LUT to enable a comprehensive evaluation (see Fig. 3.3; S0 and S15 should experience different paths to reach the destination). The counter module comprises two counters. One is a hardcoded one to issue the stop signal after every 1 ms to the other which receives the ring oscillator variations. In this manner, we can measure the frequency of the ring oscillator path by counting how many times it oscillates in 1 ms. Note that the wrapper looks very complicated, but their actual functionality is just simply providing a clock signal to the RO implementation for recycled chip detection from the processor side since there is no available hardware clock source (external oscillator) connected to the FPGA fabric. We mainly focus on the inst:`top_RO` and its submodules as illustrated in Fig. 3.8. Inside the RO instance, in order to guarantee the structure or logic would not be altered during the synthesis procedure, the implementation is written in a primitive-based manner where each LUT is precisely constructed by specifying its connectivity and content as shown in Fig. 3.9.

```
Design Sources (1)
  design_1_wrapper (design_1_wrapper.v) (1)
    design_1_i : design_1 (design_1.bd) (1)
      design_1 (design_1.v) (2)
        processing_system7_0 : design_1_prc
        top_RO_0 : design_1_top_RO_0_0 (M
          design_1_top_RO_0_0 (design_1_tc
            inst : top_RO (top_RO.v) (3)
              RO : Ring_Oscillator (Ring
              cnt : counter (counter.v)
              vio : vio_0 (vio_0.xci)
```

Fig. 3.7 Design hierarchy of the RO implementation for recycled chip detection

```
23 module top_RO( clk, done);
24
25 input clk;
26 output done;
27
28 wire reset;
29
30 (* dont_touch = "yes" *) wire in1;
31 (* dont_touch = "yes" *) wire in2;
32 (* dont_touch = "yes" *) wire in3;
33 (* dont_touch = "yes" *) wire in4;
34 (* dont_touch = "yes" *) wire in5;
35 (* dont_touch = "yes" *) wire [4:0] path;
36 (* dont_touch = "yes" *) wire RO_out;
37 (* dont_touch = "yes" *) wire [31:0] value;
38
39 Ring_Oscillator RO(.in1(path[0]), .in2(path[1]), .in3(path[2]), .in4(path[3]), .in5(path[4]), .out(RO_out),.reset(reset));
40 counter cnt(.clk(clk), .reset(reset), .done(done), .RO_out(RO_out), .value(value));
```

Fig. 3.8 Code snippet of the **top_RO** implementation

```
40 // LUT1
41 LUT6 #(
42 .INIT(64'h5555555555555555) // Specify LUT Contents
43 ) LUT1 (
44 .O(wire1), // LUT general output
45 .I0(wire_start), // LUT input
46 .I1(in1), // LUT input
47 .I2(in2), // LUT input
48 .I3(in3), // LUT input
49 .I4(in4), // LUT input
50 .I5(in5) // LUT input
51 );
```

Fig. 3.9 Code snippet of a single LUT instance

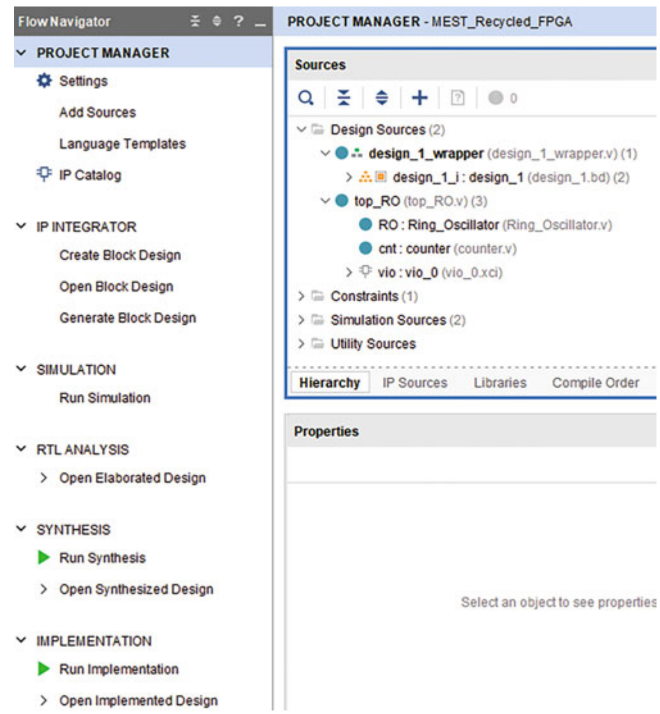


Fig. 3.10 Compiling the design in Xilinx Vivado: recycled IC detection

How to Use the RO-Based Odometer on the MiniZed Platform Step by Step? The detailed experimental flow using the RO-based odometer on the MiniZed platform is illustrated in a step-by-step manner. The design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch3_recycled_ic_odometer/. We will first introduce how to compile the RTL design into FPGA bitstreams and download it into the target FPGA (MiniZed board) through Xilinx Vivado (see Fig. 3.7). Then, the VIO interface is accessed during run-time to control the primitive and retrieve the readings. Finally, we will move around the sensor to test the aging extent at different corners of the target FPGA device:

Step 1 We first need to synthesize, implement, and generate the consequent bitstream for the target FPGA (Fig. 3.10). Since the combinational loops in ring oscillators might be considered illegitimate designs by Vivado, we need to add the constraint to degrade the consequent errors to be warnings in Vivado by using the instruction **set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets *{loop net name}*]** in the XDC constraint file (see Fig. 3.11). Before loading the bitstream, the FPGA board must be connected by following the steps in Fig. 3.12. Then, we specify the desired bitstream file and the debug probe file (See Fig. 3.13) which is used to set up the virtual IO interface for selecting the desired LUT path and accessing the counter value.

```
4  set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets design_1_i/top_RO_0/inst/RO/wire1]
5
6  create_pblock pblock_1
7  resize_pblock [get_pblocks pblock_1] -add {SLICE_X14Y51:SLICE_X43Y98}
8  resize_pblock [get_pblocks pblock_1] -add {DSP48_X1Y22:DSP48_X1Y37}
9  resize_pblock [get_pblocks pblock_1] -add {RAMB18_X1Y22:RAMB18_X2Y37}
10 resize_pblock [get_pblocks pblock_1] -add {RAMB36_X1Y11:RAMB36_X2Y18}
11 create_pblock pblock_2
12 add_cells_to_pblock [get_pblocks pblock_2] [get_cells -quiet [list design_1_i]]
13 resize_pblock [get_pblocks pblock_2] -add {SLICE_X0Y0:SLICE_X17Y49}
14 resize_pblock [get_pblocks pblock_2] -add {DSP48_X0Y0:DSP48_X0Y19}
15 resize_pblock [get_pblocks pblock_2] -add {RAMB18_X0Y0:RAMB18_X0Y19}
16 resize_pblock [get_pblocks pblock_2] -add {RAMB36_X0Y0:RAMB36_X0Y9}
```

Fig. 3.11 Code snippet of the XDC constraint file

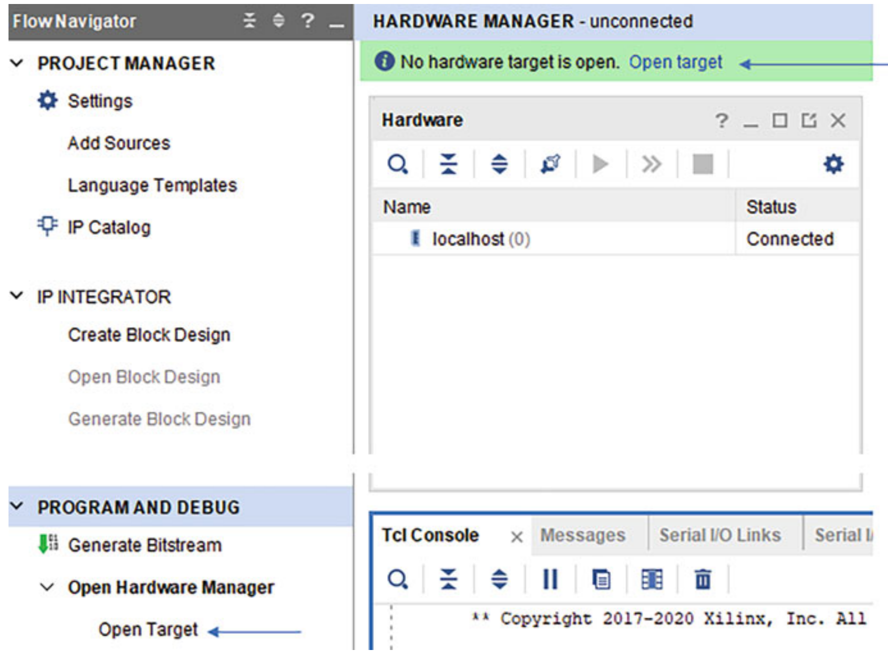


Fig. 3.12 Connect the MiniZed board

Step 2 After downloading the bitstream, in the Xilinx Vivado, one will see a dashboard called `hw_vios` pops up (otherwise, you might want to double click the `hw_vio_1` in the Hardware window). At the first time, you might have to click the + button to add the path and value signals to the dashboard. With this interface, you can set the path parameter to specify which LUT path should be measured in the current run while the value will reflect its frequency. As we set the time window to be 1 ms, the frequency will value/1000 MHz. For example, the value in Fig. 3.14 is 235239 which means the first LUT path exhibits a 235.239 MHz frequency. Every time you apply a new path value in this window, the reset value should be set to be 0

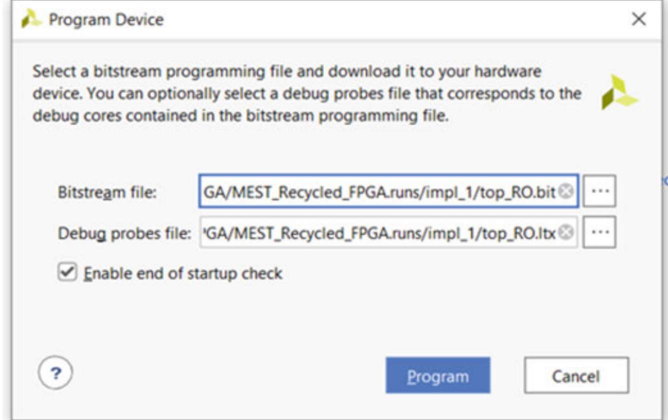


Fig. 3.13 Download bitstream

hw_vio_1

Name	Value	Activ...	Directi...	VIO
design_1_i/top_RO_0/inst/path[4:0]	[U] 11		Output	hw_vio_1
design_1_i/top_RO_0/inst/reset	[B] 1		Output	hw_vio_1
design_1_i/top_RO_0/inst/value[31:0]	[U] 235239		Input	hw_vio_1

Fig. 3.14 Inputs and outputs inside the hardware VIO dashboard

and then 1 to enforce the CDIR design to be restarted to get the response. Note that the statistics during the experiments might be a little different because of the impact of ambient temperature and power supply.

Step 3 The experimental FPGA device has been aged so different LUT paths would have a distinguished frequency. To measure the aging extent of multiple portions, we need to apply pblock constraints to the ring oscillator primitive to move the sensor. You can open the implemented design and then draw pblocks using the button as highlighted in Fig. 3.15. There are two pblock regions we created in this project. The pblock1 is at the bottom while pblock2 locates at the top right corner. To move the odometer module, we apply different pblock constraints (pblock1 and pblock2 in Fig. 3.15). You can click the P+ button to create new pblocks and right-click the desired module (RO here), floorplanning –> assign to pblock. Regenerate the bitstreams with the RO instance in different locations. The location of pblock1 is typically the default mapping location of the resources so it is more aged than pblock2. For example, for the same LUT path (set the path parameter to be 5’h00), we can observe that the value parameter is around 224.9 MHz and 246.3 MHz in pblock1 and pblock2, respectively. In other words, the pblock1 is more degraded

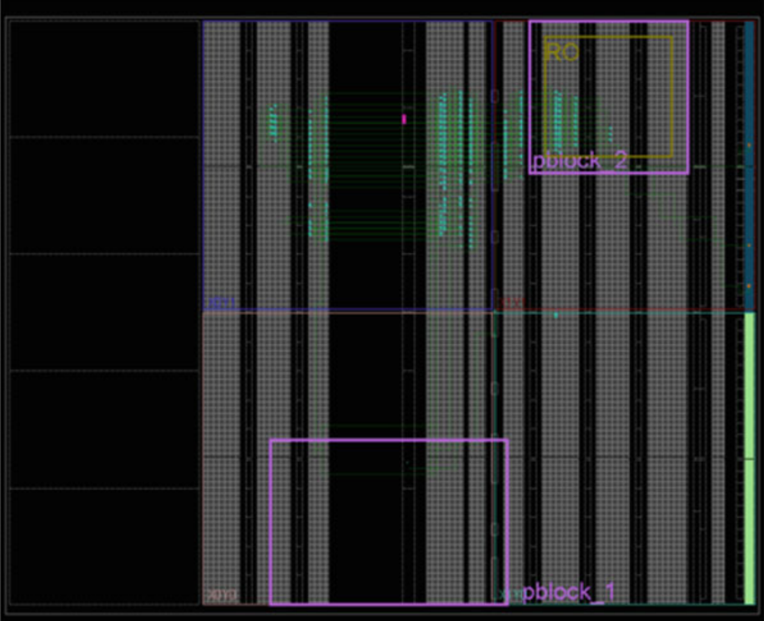


Fig. 3.15 Pblock constraints in the FPGA

due to the aging extent of the transistors. Therefore, we can conclude this device is a recycled (aged) device by using this methodology.

3.6 Conclusion

In this chapter, we discussed recycled IC detection using an RO-based odometer. We experimentally demonstrated how to use the RO sensor to measure the aging of a target FPGA to distinguish recycled devices. By the end of this chapter, readers will learn about the features of recycled FPGA devices and a typical ring oscillator-based recycled FPGA detection methodology and FPGA development procedure. Also, they will learn about how to use virtual IO interfaces to control/observe the infield application and detect counterfeit chips.

References

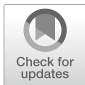
1. Alam, M., Chowdhury, S., Tehranipoor, M.M., Guin, U.: Robust, low-cost, and accurate detection of recycled ICs using digital signatures. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 209–214. IEEE, Piscataway (2018)

2. Alam, M.M., Tehranipoor, M., Forte, D.: Recycled FPGA detection using exhaustive LUT path delay characterization and voltage scaling. IEEE Trans. Very Large Scale Integr. Syst. **27**(12), 2897–2910 (2019)
3. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: Design, implementation and analysis of efficient hardware-based security primitives. In: 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), pp. 198–199. IEEE, Piscataway (2020)
4. Anandakumar, N.N., Hashmi, M.S., Tehranipoor, M.: FPGA-based physical unclonable functions: a comprehensive overview of theory and architectures. Integration **81**, 175–194 (2021)
5. Anandakumar, N.N., Hashmi, M.S., Chaudhary, M.A.: Implementation of efficient XOR arbiter PUF on FPGA with enhanced uniqueness and security. IEEE Access **10**, 129832–129842 (2022)
6. Asadizanjani, N., Gattigowda, S., Tehranipoor, M., Forte, D., Dunn, N.: A database for counterfeit electronics and automatic defect detection based on image processing and machine learning. In: ISTFA 2016, pp. 580–587. ASM International (2016)
7. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Los Altos (2018)
8. Chen, X., Liu, Q., Wang, Y., Xu, Q., Yang, H.: Low-overhead implementation of logic encryption using gate replacement techniques. In: 2017 18th International Symposium on Quality Electronic Design (ISQED), pp. 257–263. IEEE, Piscataway (2017)
9. Counterfeit Report. https://inside.battelle.org/blog-details/are-counterfeit-circuits-in-your-electronic-devices. Accessed 29 Nov 2022
10. Dey, S., Dash, S., Nandi, S., Trivedi, G.: PGIREM: reliability-constrained IR drop minimization and electromigration assessment of VLSI power grid networks using cooperative coevolution. In: 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 40–45. IEEE, Piscataway (2018)
11. Dey, S., Nandi, S., Trivedi, G.: Machine learning for VLSI CAD: a case study in on-chip power grid design. In: 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 378–383. IEEE, Piscataway (2021)
12. Dey, S., Park, J., Pundir, N., Saha, D., Shuvo, A.M., Mehta, D., Asadi, N., Rahman, F., Farahmandi, F., Tehranipoor, M.: Secure physical design. Cryptology ePrint Archive (2022)
13. Dofe, J., Yan, C., Kontak, S., Salman, E., Yu, Q.: Transistor-level camouflaged logic locking method for monolithic 3d IC security. In: 2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST), pp. 1–6. IEEE, Piscataway (2016)
14. Dogan, H., Forte, D., Tehranipoor, M.M.: Aging analysis for recycled FPGA detection. In: 2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 171–176. IEEE, Piscataway (2014)
15. Guin, U., Zhang, X., Forte, D., Tehranipoor, M.: Low-cost on-chip structures for combating die and IC recycling. In: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, Piscataway (2014)
16. Guin, U., Forte, D., Tehranipoor, M.: Design of accurate low-cost on-chip structures for protecting integrated circuits against recycling. IEEE Trans. Very Large Scale Integr. Syst. **24**(4), 1233–1246 (2015)
17. Guin, U., Huang, K., DiMase, D., Carulli, J.M., Tehranipoor, M., Makris, Y.: Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proc. IEEE **102**(8), 1207–1228 (2014)
18. Guo, Z., Rahman, M.T., Tehranipoor, M.M., Forte, D.: A zero-cost approach to detect recycled SoC chips using embedded SRAM. In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 191–196. IEEE, Piscataway (2016)
19. He, K., Huang, X., Tan, S.X.-D.: Em-based on-chip aging sensor for detection and prevention of counterfeit and recycled ICs. In: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 146–151. IEEE, Piscataway (2015)
20. Huang, K., Liu, Y., Korolija, N., Carulli, J.M., Makris, Y.: Recycled IC detection based on statistical methods. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **34**(6), 947–960 (2015)

21. Hussain, S.U., Yellapantula, S., Majzoobi, M., Koushanfar, F.: BIST-PUF: online, hardware-based evaluation of physically unclonable circuit identifiers. In: 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 162–169. IEEE, Piscataway (2014)
22. Jang, B., Lee, J.K., Choi, M., Kim, K.K.: On-chip aging prediction circuit in nanometer digital circuits. In: 2014 International SoC Design Conference (ISOCC), pp. 68–69. IEEE, Piscataway (2014)
23. Jarvis, R.W., Mcintyre, M.G.: Split manufacturing method for advanced semiconductor circuits. US Patent 7,195,931, 27 Mar 2007
24. Karmakar, R., Chattopadhyay, S., Kapur, R.: Enhancing security of logic encryption using embedded key generation unit. In: 2017 International Test Conference in Asia (ITC-Asia), pp. 131–136 IEEE (2017)
25. Karmakar, R., Prasad, N., Chattopadhyay, S., Kapur, R., Sengupta, I.: A new logic encryption strategy ensuring key interdependency. In: 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID), pp. 429–434. IEEE, Piscataway (2017)
26. Keane, J., Wang, X., Persaud, D., Kim, C.H.: An all-in-one silicon odometer for separately monitoring HCI, BTI, and TDDB. IEEE J. Solid-State Circuits **45**(4), 817–829 (2010)
27. Lin, C.W., Ghosh, S.: Novel self-calibrating recycling sensor using Schmitt-Trigger and voltage boosting for fine-grained detection. In: Sixteenth International Symposium on Quality Electronic Design, pp. 465–469. IEEE, Piscataway (2015)
28. Liu, M., Kim, C.H.: A powerless and non-volatile counterfeit IC detection sensor in a standard logic process based on an exposed floating-gate array. In: 2017 Symposium on VLSI Technology, pp. T102–T103. IEEE, Piscataway (2017)
29. Marchand, C., Bossuet, L., Jung, E.: IP watermark verification based on power consumption analysis. In: 2014 27th IEEE International System-on-Chip Conference (SOCC), pp. 330–335. IEEE, Piscataway (2014)
30. Mark (Mohammad) Tehranipoor, Guin, U., Forte, D.: Counterfeit Integrated Circuits: Detection and Avoidance. Springer, Berlin (2015)
31. Plaza, S.M., Markov, I.L.: Solving the third-shift problem in IC piracy with test-aware logic locking. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **34**(6), 961–971 (2015)
32. Rahman, M.T., Rahman, M.S., Wang, H., Tajik, S., Khalil, W., Farahmandi, F., Forte, D., Asadizanjani, N., Tehranipoor, M.: Defense-in-depth: a recipe for logic locking to prevail. Integration **72**, 39–57 (2020)
33. Savanur, P.R., Alladi, P., Tragoudas, S.: A BIST approach for counterfeit circuit detection based on NBTI degradation. In: 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 123–126. IEEE, Piscataway (2015)
34. Semiconductor Market Report. https://www.fortunebusinessinsights.com/semiconductor-market-102365. Accessed 29 Nov 2022
35. Shinde, S., Jothibasu, S., Ghasr, M.T., Zoughi, R.: Wideband microwave reflectometry for rapid detection of dissimilar and aged ICs. IEEE Trans. Instrum. Meas. **66**(8), 2156–2165 (2017)
36. Tehranipoor, M.: Emerging Topics in Hardware Security. Springer, Berlin (2021)
37. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
38. Tehranipoor, M., Forte, D., Rose, G.S., Bhunia, S.: Security Opportunities in Nano Devices and Emerging Technologies (2017)
39. Tehranipoor, M., Pundir, N., Vashistha, N., Farahmandi, F.: Hardware Security Primitives. Springer, Berlin (2022)
40. Wang, Y., Cotofana, S., Fang, L.: A unified aging model of NBTI and HCI degradation towards lifetime reliability management for nanoscale MOSFET circuits. In: 2011 IEEE/ACM International Symposium on Nanoscale Architectures, pp. 175–180. IEEE, Piscataway (2011)
41. Wu, X., Zhang, C., Du, W.: An analysis on the crisis of “chips shortage” in automobile industry—Based on the double influence of COVID-19 and trade Friction. In: Journal of Physics: Conference Series, vol. 1971, p. 012100. IOP Publishing (2021)

42. Xie, Y., Srivastava, A.: Delay locking: security enhancement of logic locking against IC counterfeiting and overproduction. In: Proceedings of the 54th Annual Design Automation Conference 2017, pp. 1–6 (2017)
43. Yu, C., Zhang, X., Liu, D., Ciesielski, M., Holcomb, D.: Incremental sat-based reverse engineering of camouflaged logic circuits. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **36**(10), 1647–1659 (2017)
44. Zhang, X., Salmani, H.: Integrated circuit authentication: hardware Trojans and counterfeit detection (2014)
45. Zhang, T., Rahman, F., Tehranipoor, M., Farahmandi, F.: FPGA-chain: enabling holistic protection of FPGA supply chain with blockchain technology. IEEE Design & Test **40**(2), 127–136 (2022)
46. Zheng, Y., Liu, H., Dorsey, J., Mitra, N.J.: Smartcanvas: context-inferred interpretation of sketches for preparatory design studies. In: Computer Graphics Forum, vol. 35, pp. 37–48. Wiley Online Library (2016)

Chapter 4
Recycled FPGA Detection



4.1 Introduction

FPGAs are widely used today for their relatively low engineering costs, instant availability, high performance, and low power consumption. Recent reports indicate that programmable logic is among the top five counterfeited electronic components with a percentage of 8.3% of reported counterfeit incidents [1, 20]. With the increased volume of usage, FPGAs will likely become an even better target for counterfeiting, and thus, their reliability becomes significant for those within both government and industry [17]. The recycled electronic components are recovered from a system and then modified to be misrepresented as a new component of an original component manufacturer [10]. The recycled parts may have been exposed to harmful conditions, such as high humidity and temperatures, but they have reliability issues. Due to the high volume of used components each year, the same component must be recycled multiple times. Today's complex electronic component supply chain makes preventing the infiltration of recycled FPGAs very challenging [18].

Some works have been aimed at recycled IC detection using electrical tests and on-chip sensors [11, 12, 21]. Mostly, intrinsic delay [11] and path delay variations have been used [2, 14, 16] as sensors. But all sensor-based recycled IC detection methods have area and power overhead, and they need to work for the existing ICs already in circulation. In 2012, Zhang et al. proposed a path delay-based method to detect recycled IC that does not have any area overhead [21]. In 2015, Huang et al. proposed a statistical method to detect recycled ICs using a one-class classifier and degradation analysis [12]. In 2016, Bergman et al. used a power side channel to detect counterfeit [5]. These methods need golden or reference data and mainly focus on ASICs, not FPGAs. In 2014, Dogan et al. proposed a detection approach using the aging degradation in FPGA after accelerated aging [8]. As a result, this method is less effective for detecting only a few months old FPGAs since it considers only a portion of the circuit for degradation. The requirement

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_4

of golden data and accelerated aging makes this method less practical. In 2016, Couch et al. used ring oscillator (RO) frequencies of FPGAs and machine learning-based classifiers to identify manufacturing lots considering lot-to-lot frequency variation [7]. In 2019, M. M. Alam et al. first proposed using exhaustive look-up table (LUT) path delay characterization to identify recycled FPGAs [1]. In this chapter, we discussed FPGA aging to understand the nature of recycled components. In particular, this chapter can help a reader better understand and gain hands-on experience in detecting recycled FPGA with or without access to golden or reference data using the exhaustive look-up-table (LUT) path delay characterization.

The rest of the chapter is organized as follows: Sect. 4.2 briefly discusses basic information on the look-up table structure, RO path formation, and aging mechanism. Section 4.3 briefly provides the classification using supervised and unsupervised methods. The experimental setup for detecting recycled FPGAs is provided in Sect. 4.4. Capturing RO frequencies and recycled FPGA detection is presented in 4.5. Finally, conclusions are presented in Sect. 4.6.

4.2 Background

4.2.1 Look-Up Table Structure

LUTs are the basic building block of FPGA applications. Generally, an n input LUT contains $2n$ number of SRAM cells to hold the mapped values and a set of multiplexers that selects the cells to drive out the cell value to output. For a four-input LUT, 16 SRAM cells are necessary. Partially used and fully used LUT architecture example for a four-input LUT is given in Fig. 4.1. Used paths are colored in red and unused paths are colored in black.

For partially used LUT, in Fig. 4.1a, for a four-input LUT and 3-bit adder implementation, 1 input is unused, and half of the paths are unused. It has been found that for combinational circuits containing less than 2000 LUTs, approximately 50% of the LUTs use four inputs or fewer and 82% of the LUTs use five inputs or fewer. The used portions experience more aging due to switching activity, while the unused portions experience less aging irrespective of switching activity. For fully used LUT, in Fig. 4.1b, for a four-input LUT and carryout of 4-bit adder implementation, all four inputs and all the LUT paths have been utilized. Now, aging that each path will experience depends on the switching activity. ROs can be implemented using such LUTs where they are mapped as an inverter. The delay of such a stage is the combination of delay of SRAM cells, selector transistors, and interconnect delay of the LUT.

The necessity of exhaustive path configurations: As the ROs are implemented using LUTs as an inverter and the inverter has only one input and one output, ROs implemented using one set of input pins do not cover all the paths of the LUTs. An application may have a different number of paths and their aging behavior differs. When a specific path is suffering from aging during normal FPGA usage, a method

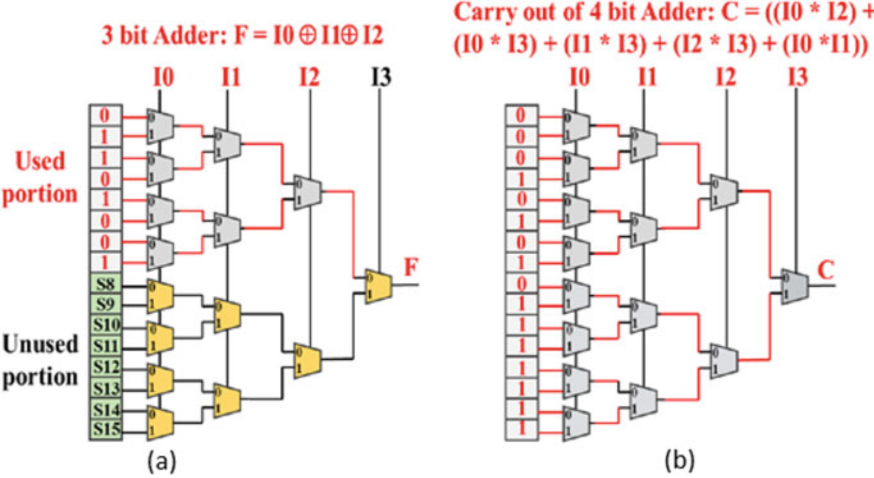


Fig. 4.1 LUT architecture: (**a**) partially used LUT, (**b**) fully used LUT

that only characterizes the delay of one path is not comprehensive enough to give the complete delay characterization. Exhaustive path delay characterization is necessary so that path delay will represent actual aging characterization. To do that, XOR- and XNOR-based ROs can be implemented in place of inverter-based ROs.

4.2.2 RO Path Formation Using XNOR and XOR Logic

An example of RO path formation using XNOR and XOR logic is presented in Figs. 4.2, and 4.3 depicts the truth table of the implementation for four-input LUTs. In this example, 16 SRAM cells for each LUT is shown in boxes to the left of each LUT. If one of the LUT inputs is considered RO and all other inputs remain unchanged, the standard logic XOR or XNOR acts as an inverter. The design tools cannot optimize these standard logics. Here, I0 is considered as RO input and F is the RO output. F of one LUT enters as I0 in the next LUT to construct RO structure. For four-input LUTs, there can be total of eight paths through which ROs can be implemented. The input patterns for all eight paths are presented in Fig. 4.3.

If we consider the RO frequencies of path p1 to p8 as a frequency array, the frequencies will vary very little in a new FPGA due to process variation. But in a used FPGA, aging degradation will affect each path differently and create additional frequency variation.

4.2.3 Aging Mechanism

Throughout the lifetime of an operational FPGA, degradation happens due to bias temperature instability (BTI), hot carrier injection (HCI), time-dependent dielectric

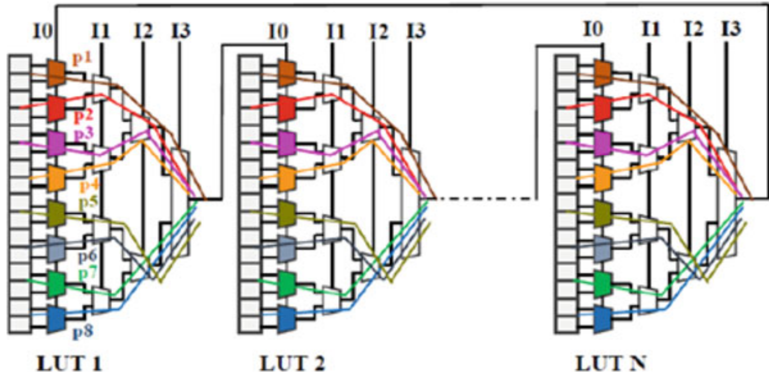


Fig. 4.2 RO in four-input LUTs with eight possible paths p1 to p8

LUT Input				Inverter Output (F)		Path
I3	I2	I1	I0	XNOR	XOR	
0	0	0	0	1		p1
0	0	0	1	0		
0	0	1	0		1	p2
0	0	1	1		0	
0	1	0	0	1		p3
0	1	0	1	0		
0	1	1	0		1	p4
0	1	1	1		0	
1	0	0	0		1	p5
1	0	0	1		0	
1	0	1	0	1		p6
1	0	1	1	0		
1	1	0	0		1	p7
1	1	0	1		0	
1	1	1	0	1		p8
1	1	1	1	0		

Fig. 4.3 Formation of paths using XNOR and XOR logic functions

breakdown (TDDB), and electromigration [1, 3, 13]. Specially BTI and HCI impact significantly on the switching speed of the transistors which is measurable and is reflected in the measured RO frequencies.

- *Bias Temperature Instability*: PMOS and NMOS threshold voltages are increased by negative BTI (NBTI) and positive BTI (PBTI), respectively. NBTI traps the interface of the channel and the gate oxide of PMOS transistors during prolonged gate-to-source negative bias stress. There is an increase in threshold voltage, which leads to an increase in switching delay. When stress is removed, partial recovery occurs, but high temperatures and voltages aggravate the effect.

Although NBTI dominates PBTI beyond 65-nm technology nodes, the introduction of high-k gate and metal gate transistors elevates PBTI's influence [4, 6].

- *Hot carrier injection*: A high electric field in the drain region causes electrons or holes on the substrate to gain high energy and get trapped in the gate oxide layer [19]. A charge defect in the dielectric layer builds up an electric field over time, increasing the threshold voltage. Switching activity is slowed down and delays are introduced. As effective channel lengths decrease, HCI degradation becomes more prominent at lower technology nodes.

4.3 Classification Using Supervised and Unsupervised Methods

To exploit the degradation of look-up tables (LUTs), we implement a sophisticated ring oscillator (RO). Using supervised and unsupervised machine learning algorithms, we discuss two recycled FPGA detection methods. Algorithms for supervised and unsupervised machine learning are briefly discussed in this section.

4.3.1 Supervised Classification Method

Machine learning algorithms are briefly discussed in this section. A reference or golden FPGA is assumed in the supervised method. This method of classification is illustrated in Fig. 4.4. ROs based on XNOR and XOR are placed in golden components, a frequency array is obtained for each RO, and a support vector machine (SVM) classifier is trained using the data. Class labels and features are included in the training set. LUT paths are the features in the training set. SVM creates a decision model based on the training data that predicts the test data's label. The classification will be improved by adding other features, such as kurtosis and skewness of the frequency distribution.

4.3.2 Unsupervised Classification Method

It is possible to use unsupervised classification methods when golden or reference FPGAs are limited or when they are unavailable. Figure 4.5 shows the steps in the unsupervised classification process. A frequency array is created by placing XNOR- and XOR-based ROs. For this method, K-means clustering is used [1, 15]. Using K-means clustering, samples are divided into k clusters by minimizing the average squared distance between cluster members. The average silhouette value (SV) of all cluster sets is used to determine the cluster number. Using silhouette values, you can

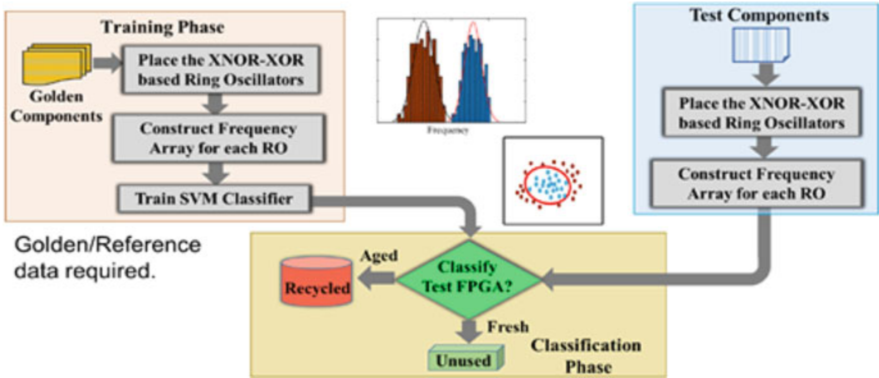


Fig. 4.4 Supervised classification flow

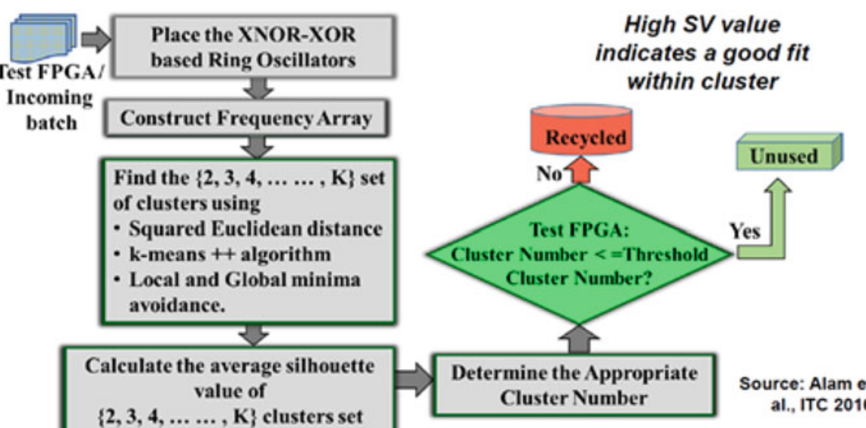


Fig. 4.5 Unsupervised classification flow

tell how well frequencies fit within their own clusters and how they differ from their neighbors. When the SV value is high, it indicates that the cluster is well-fitting. The appropriate cluster number is obtained and then compared to a threshold cluster number before a decision is made. It is possible to distinguish between new and recycled FPGAs with high accuracy and without gold samples by using a threshold of two or three.

4.4 The Setup for the Experiment

In this section, we will use Artix 7 FPGA board ensuring it is correctly set up, a laptop/pc, and a USB interface for the connection. For the software, we will use

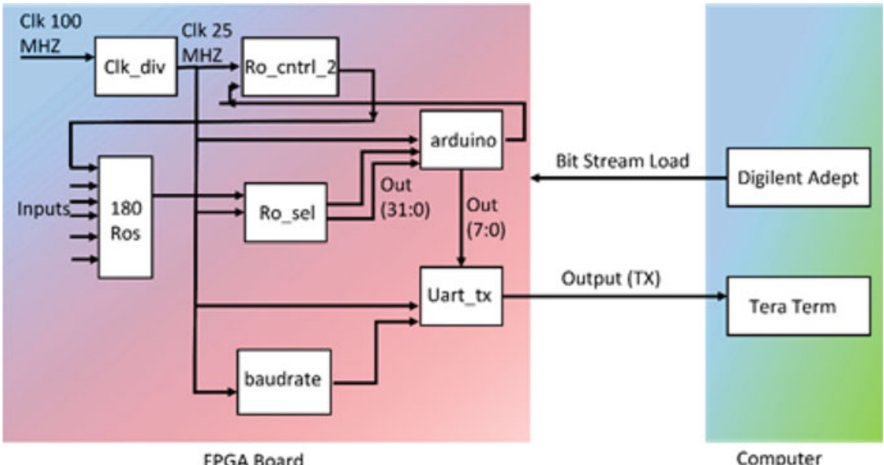


Fig. 4.6 Overview of the experimental setup with internal components

Xilinx ISE 14.7 where we will add the VHDL/Verilog programs of all the modules of our design and generate the bitstream. After that, we will employ Digilent Adept to load the bitstream into the specific FPGA board. Finally, Tera Term will be utilized to see the output via UART. The design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch4_recycled_fpga/. An overview of the experimental setup with internal components is shown in Fig. 4.6. The module ROs is the instantiated 180 hard macro used as the delay element to capture the frequency of the ROs utilizing the LUTs of the FPGA. The rest of the circuitry is to assist in measuring the frequency, e.g., divide the clk and select and control the ROs. A brief description of submodules is given below:

Brief description of submodules:

- `clkdiv_4to1`: This submodule is to divide the 100-MHZ FPGA global clock to 25 MHZ.
- `rosel_v2`: The purpose of this module is to select Ros one by one.
- `rocntrl_2`: After selecting the ROs, this module will enable the pin of the selected RO, count for 100000 clock cycles, and then capture the output value.
- `uart_tx`: This one is to establish a connection between the computer and the board.
- `baudrate`: Baud rate is the rate at which information is transferred between laptop and board. For our experiment, we will take it as 115200 baud which means that the serial port is capable of transferring a maximum of 115200 bits per second. This is specified in this module by a simple counter.
- `arduino`: In this module, a 32-bit output will be divided into 8 bits because the USB serial port supports 8 bits. It also sends and receives signals of data.

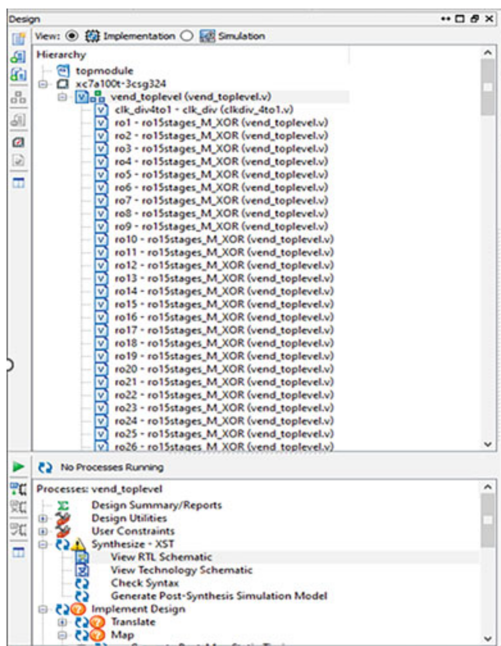


Fig. 4.7 Design hierarchy of the recycled FPGA detection

4.4.1 Bitstream Generation

- At first, we have to install Xilinx ISE 14.7 software. The WEBPACK version is free for our target Artix-7 FPGA.
- For this tutorial, we will provide the codes along with project files to build a project using the Xilinx ISE software. When we open the project file, the screen will be shown in Fig. 4.7.
- If someone wants to create the project from scratch step by step, please watch the video provided in https://www.youtube.com/watch?v=DIOIl3P65hg where step-by-step procedure is shown in Xilinx ISE to create the bitstream.
- In our implementation, there are two parts: (1) creating hard macro and (2) instantiating the hard macro in top module with other submodules. To create the hard macro, the following steps need to be considered:

  1. To load the project directly: Needed file name, hardmacro (type: Xilinx ISE Project).
  2. To create the hard macro from scratch: Needed file name, ro (type: VHDL), ro.UCF (type: user constraint file).
  3. After creating the hard macro, an NMC file will be created. We need to provide the NMC file path to the top module.
  4. If someone wants to skip the hard macro creation part and use the pre-created hard macro NMC file, the file will be provided named `ro_15stages_M_XOR.nmc`. Note that this hard macro will only work

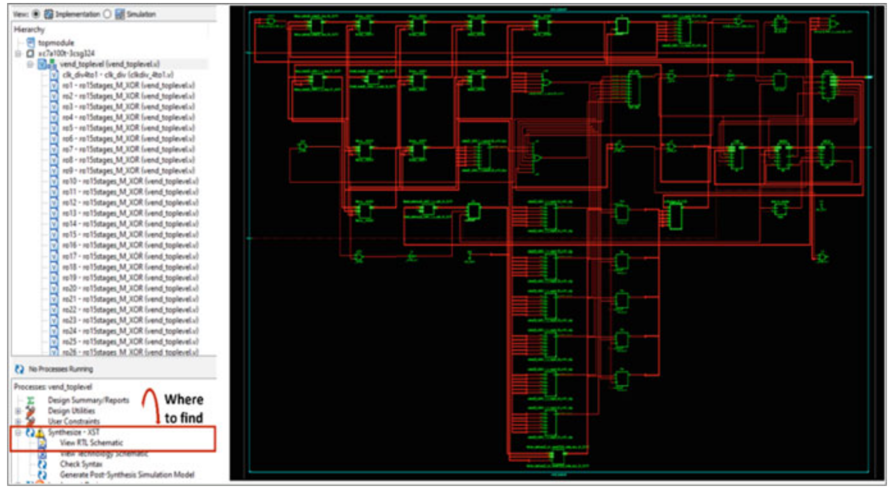


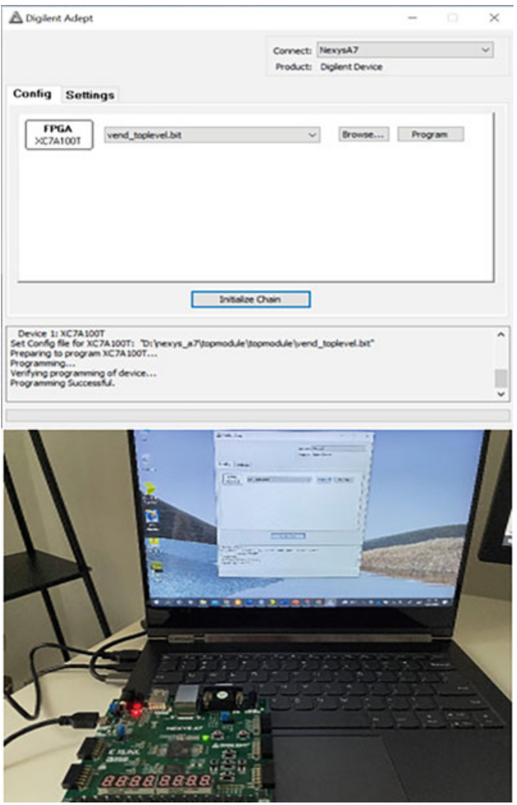
Fig. 4.8 RTL schematic of the design

for Artix 7 FPGA board; in case of using other boards, hard macro creation step is a must as the placing and routing will vary board to board.

- After creating the hard macro, it will be instantiated in the top module with other submodules.
- To load the project directly: Needed file name: topmodule (Type: Xilinx ISE Project).
- To create from scratch: Needed file name: `Top module- Vend_toplevel` (Type: Verilog); Submodule: `clkdiv_4to1` (Type: Verilog), `rocntrl_2` (Type: Verilog), `rosel_v2` (Type: Verilog), counter (Type: Verilog), `uart_tx` (Type: Verilog), `baudrate` (Type: VHDL), `arduino` (Type: Verilog); User constraint file: `vend_toplevel` (Type: UCF).
- There are three steps to be performed by ISE to turn our provided VHDL/Verilog code into the bitstream in the form of .bit file.

  1. Synthesis: The VHDL/Verilog codes are synthesized into a gate-level representation. In this step, an RTL schematic will be created as shown in Fig. 4.8. We encourage everyone to look into this carefully; this will give a clear overview of how every module is connected to each other as well as their purpose.
  2. Implementation: In this step, the synthesized logic will be placed and routed according to our user-defined constraint file to fit onto the device.
  3. Bitstream Generation: After the successful synthesis and implementation, it's time to generate the bitstream. A .bit file will be generated which we will load into the Artix-7 FPGA board.

Fig. 4.9 Overview of Digilent Adept software



4.4.2 Bitstream Loading

For this step, we will install software named Digilent Adept (see Fig. 4.9). After installing the software, please connect the FPGA board to the laptop with a USB. The software should be able to detect the FPGA board, and the user just needs to load the bitstream file into the FPGA.

4.4.3 Capturing Output

To see the output, we can use Tera Term, MATLAB, or Putty. In our case, we have used Tera Term. Before starting, we need to adjust the setting of COM port and baud rate as shown in Fig. 4.10. To set the baud rate, go to setup and then serial port. Set the speed to 115200. Note that it should be 115200; otherwise, we will not be able to capture the output successfully.

The output will come serially in 8-bit by 8-bit as shown in Fig. 4.11.

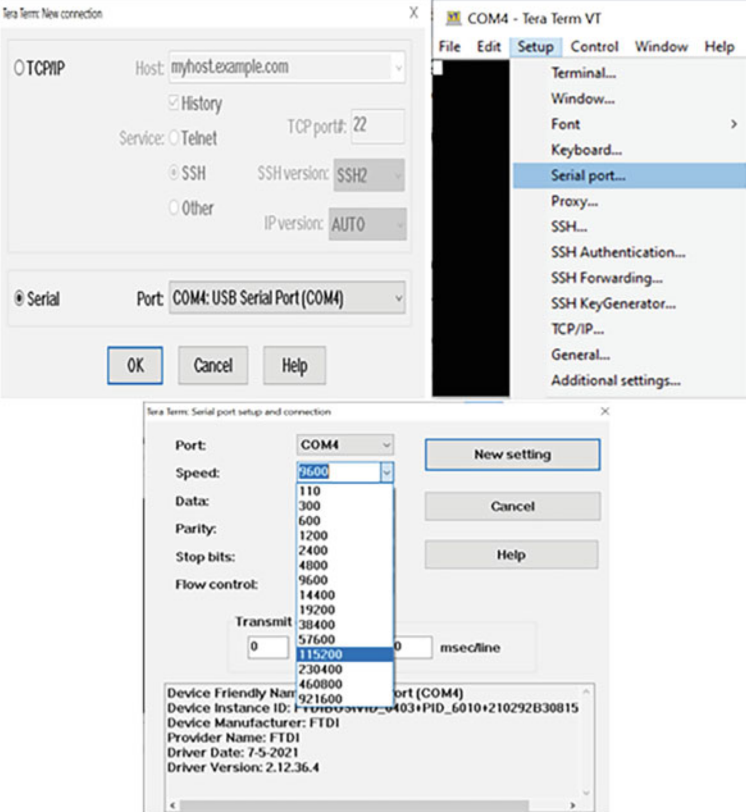


Fig. 4.10 Overview of Tera Term software

Fig. 4.11 Output of the design

4.5 Capturing RO Frequencies and Recycled FPGA Detection

Once the RO frequencies are extracted, they are first converted from binary to decimal values and saved as .csv files. For each FPGA, RO frequencies are obtained after using the FPGA for 0 hours (i.e., unused), and 8, 12, and 16 h are saved in four subfolders named as `'t_0h'`, `'t_8hours'`, `'t_12hours'`, and `'t_16hours'`, respectively. Each CSV file contains a 2D matrix, where each row represents the RO number for a particular FPGA and each column represents the

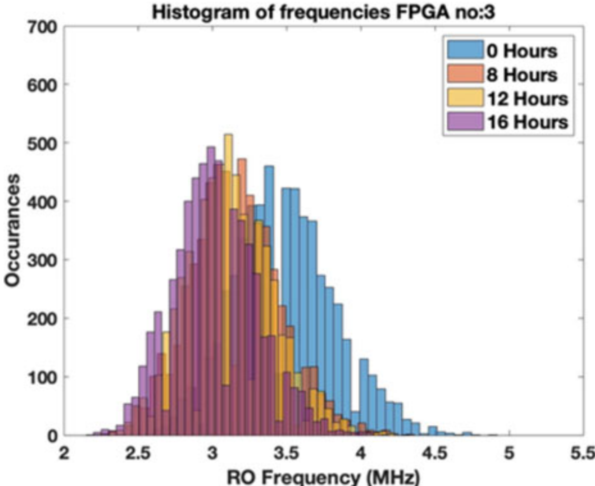


Fig. 4.12 Histogram of frequencies of used and unused RO for a specific FPGA

path. For the Spartan6LX9 data, there are 180 ROs and 32 paths. So, the size of each 2D matrix is 180X32.

4.5.1 Visualization of RO Frequencies

For visualization of the RO frequencies and how it changes with time, we have plotted histograms of the RO frequency for each FPGA and shown how it changes with aging. Figure 4.12 shows the histogram plot for one such FPGA. All the analysis and result plots shown in this document are done for FPGA ID3 of the SPartan6LX9 dataset. The number of bins chosen is 40. The MATLAB code for plotting histogram is `histogram(data,‘Name’,Value)`.

Also, for visualizing how the different RO values are scattered, the first two principal components of each of the 32-dimensional RO frequency vector is plotted using the TSNE algorithm and scatterplot. The MATLAB code for scatterplot is as follows:

```
rng default % for reproducibility
Y1 = tsne(t0,'Algorithm','barneshut','NumPCAComponents',30);
L1     = cell(180, 1);
L1(:)  = {'0 Hour'};

Y2 = tsne(t8,'Algorithm','barneshut','NumPCAComponents',30);
L2     = cell(180, 1);
L2(:)  = {'8 Hour'};
```

```
Y3 = tsne(t12,'Algorithm','barneshut','NumPCAComponents',30);
L3    = cell(180, 1);
L3(:) = {'12 Hour'};

Y4 = tsne(t16,'Algorithm','barneshut','NumPCAComponents',30);
L4    = cell(180, 1);
L4(:) = {'16 Hour'};

Y=[Y1;Y2;Y3;Y4];
L=[L1;L2;L3;L4];
size(Y)
size(L)
figure,
gscatter(Y(:,1),Y(:,2),L)
ylabel('Principal Component 1','FontWeight',
'bold', 'FontSize',15);
% Create xlabel
xlabel('Principal Component 2','FontWeight',
'bold','FontSize',15);
% Create title
title(strcat('S.Plot Reducing Dimension:',
num2str(fpga_no)),...'FontWeight','bold');
box(axes1,'on');
hold(axes1,'off');
% Set the remaining axes properties
set(axes1,'FontSize',15,'FontWeight','bold');
```

The output for the same FPGA is plotted in Fig. 4.13.

4.5.2 Analysis Using Machine Learning

As we can see from the visualization plots, the frequency distribution of the RO frequencies changes in a particular pattern; we can use these frequencies directly or statistical measures computed from these frequencies to train a machine-learning model. Machine learning algorithms can be categorized into two types. They are supervised and unsupervised methods. Each of these methods and how they are used for detecting used ROs are explained in the next subsection.

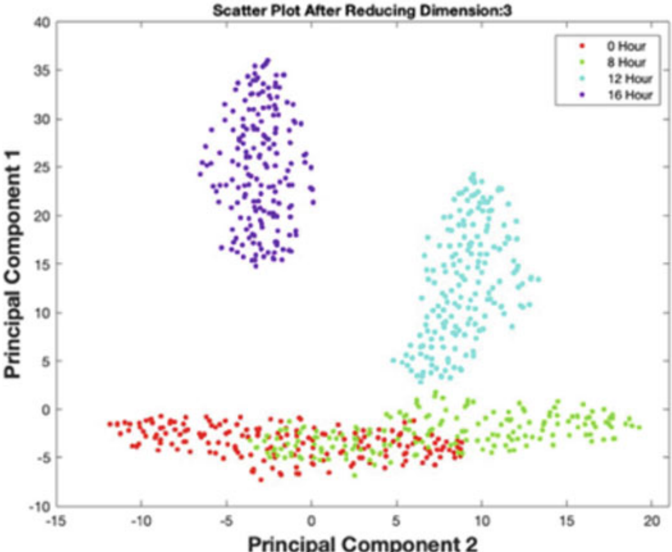


Fig. 4.13 Scatterplot for the used and unused RO for a specific FPGA

4.5.2.1 Supervised Learning Method

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. This type of algorithm takes labeled data as input and is used either for classification or for regression. There are an innumerable number of supervised learning algorithms. For our problem, we have used a support vector machine (SVM). Using this algorithm, we have trained an SVM model to classify used and unused RO. The feature vector used for training the algorithm is taken from each RO over all paths.

The given dataset is divided into two parts in the ratio 8:2, where 80% of the data is used for training and the rest is used for testing. The entire train dataset is again divided into two classes – used and unused. The data for the unused class is taken from the 0-hour files, and the data for the user class is taken from the 8-hour, 12-hour, and 16-hour files. The mean(M), skewness(S), and kurtosis(K) of the frequencies for all paths of a particular RO are computed for each data point (RO). These statistical measures are used in all combinations as features to train individual SVM models. Each of these models is then tested with a similar feature extracted from the test set. Figure 4.14a shows the receiver operator curve (ROC) for an SVM model trained on the mean of the frequencies of the train set ROs of each class and tested on the mean of the frequencies of each test set RO [1, 9]. Similarly, Fig. 4.14b and c shows the ROC curves for the models trained on the skewness and kurtosis of the RO frequencies of the train data and tested on the respective measures of the test set RO frequencies.

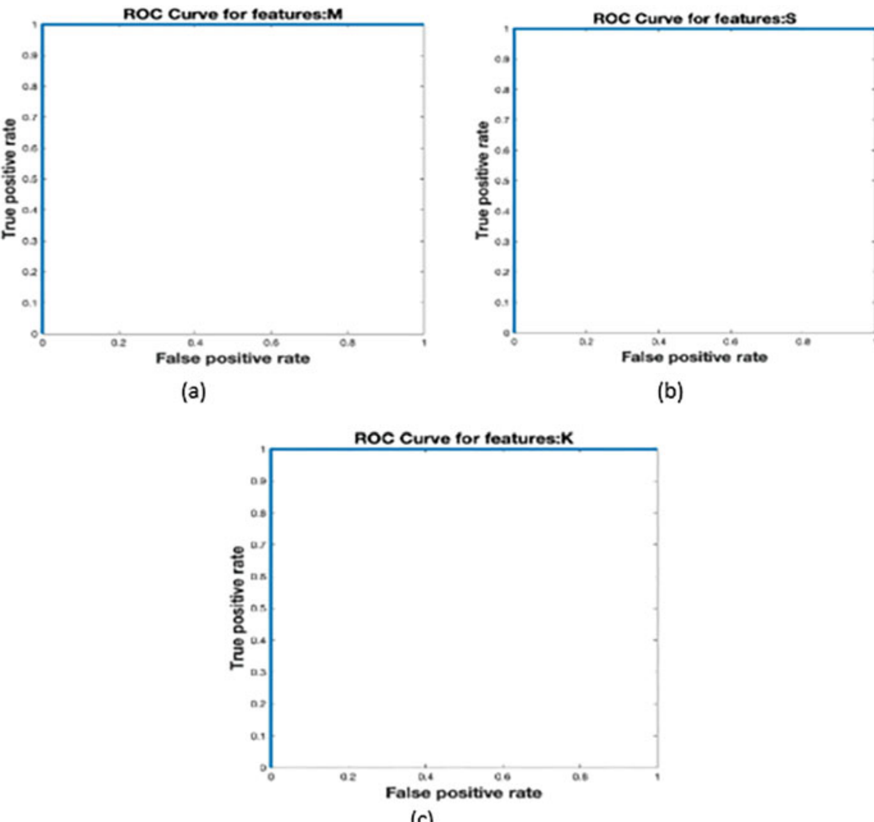


Fig. 4.14 ROC curve obtained from SVM models trained on different types of features (**a**) Mean. (**b**) Skewness. (**c**) Kurtosis

`MATLAB Code for SVM:`

An SVM model for training dataset: Each row of `RO_train` contains features from each RO, and each row of train class contains the corresponding class of the RO. A sample code snippet is provided below:

```
disp('Training Started..');
SVM_train = fitcsvm(RO_train,trainclass,'Standardize',
true, ...'KernelFunction','rbf','KernelScale','auto');
disp('Training Finised!!');
```

An SVM model for test dataset: Each row of `RO_test` contains features from each RO:

```
disp('Prediction Started..');
        ntest=(length(theclass)-idx_limit);
        all_index=(1:length(RO_data));
```

```
index_test=setdiff(all_index,index);
RO_test=RO_data(index_test,:);
[label,score] = predict(SVM_train,RO_test);
disp('Prediction Done!!');
trueclass1(1:ntest)= theclass(index_test);
```

ROC curve: Each row of `RO_test` contains features from each RO, and their classes are saved in the variable true class. This true class is used only to evaluate the accuracy of the model and not for training. A sample code snippet is provided below:

```
%AUC plot
figure;
[Xsvm1,Ysvm1,Tsvm,AUCsvm] =
perfcurve(trueclass1',score(:,2),1);
%as used in original code, modified for Kmeans
disp(AUCsvm);
plot(Xsvm1,Ysvm1,LineWidth=3)
% Create title
title(strcat('ROC Curve for features: ',head(k)),
'FontWeight',...'bold','FontSize',16);
axis square; %axis tight;
xlabel('False positive rate','FontWeight','bold','FontSize',16);
ylabel('True positive rate','FontWeight','bold','FontSize',16);
filename=strcat(save_path,'ROC_Features_',head(k),'.fig');
filename2=strcat(save_path,'ROC_Features_',head(k),'.png');
savefig(filename)
saveas(gcf,filename2)
hold off;
```

From Fig. 4.14, we see all the combination of statistical measures is giving 100% accuracy. That is, the SVM model can accurately classify the test set used and unused data. We have also tried different combinations of mean, kurtosis, and skewness. All these combinations also showed similar results.

4.5.2.2 Unsupervised Learning Method

Machine learning algorithms learn the pattern from unlabeled data and group them into categories known as unsupervised machine learning. Unlike supervised machine learning, they cannot be classified into labeled classes, but they group similar data into groups that can be identified later as a used or unused class. There are innumerable unsupervised algorithms. We have chosen the k-means algorithm for our problem, and the raw frequencies of each FPGA are used as features. Here, the hypothesis is that the frequencies of an unused FPGA under test will have less optimum clusters compared to used FPGAs. For the k-means algorithm, one needs to provide the number of clusters (k) during clustering. For these, before doing clustering, another pre-processing step is done to evaluate the optimum number

of clusters present in the used/unused FPGA under test. In this step, the average values of the silhouette indices are evaluated for a series of k values (3–16 for our case), wherein every iteration the data is divided into k clusters. Then the k value for which the average silhouette value is maximum is chosen as the optimum number of clusters for the given data.

`MATLAB Code for K-Means Clustering`:

The observed K value obtained from silhouette value analysis is then used for the k-mean clustering algorithm:

```
[cidx1,~] = kmeans(K,i,'MaxIter',100,'Start', C, ...
'EmptyAction','drop');
```

Once the optimum number of clusters is obtained for each of the CSV files, a threshold can be set to classify them into used and unused data. Figure 4.15

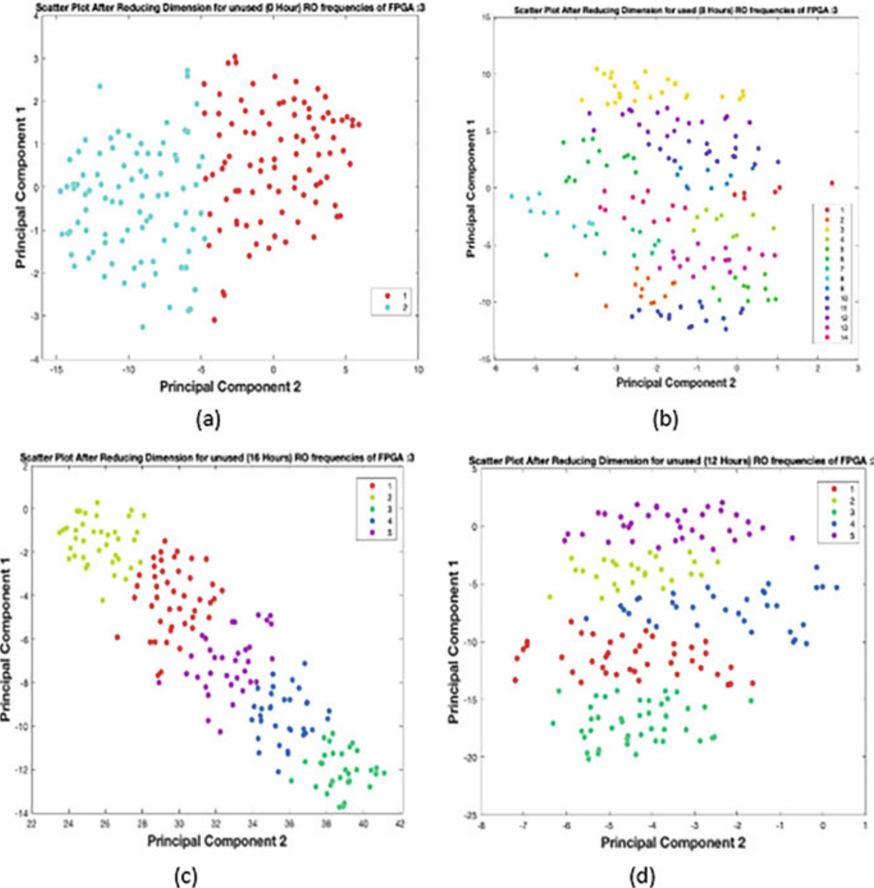


Fig. 4.15 Scatterplots of the clusters formed by frequencies of an FPGA used for a specific time (0, 8, 12, and 16 h, respectively)

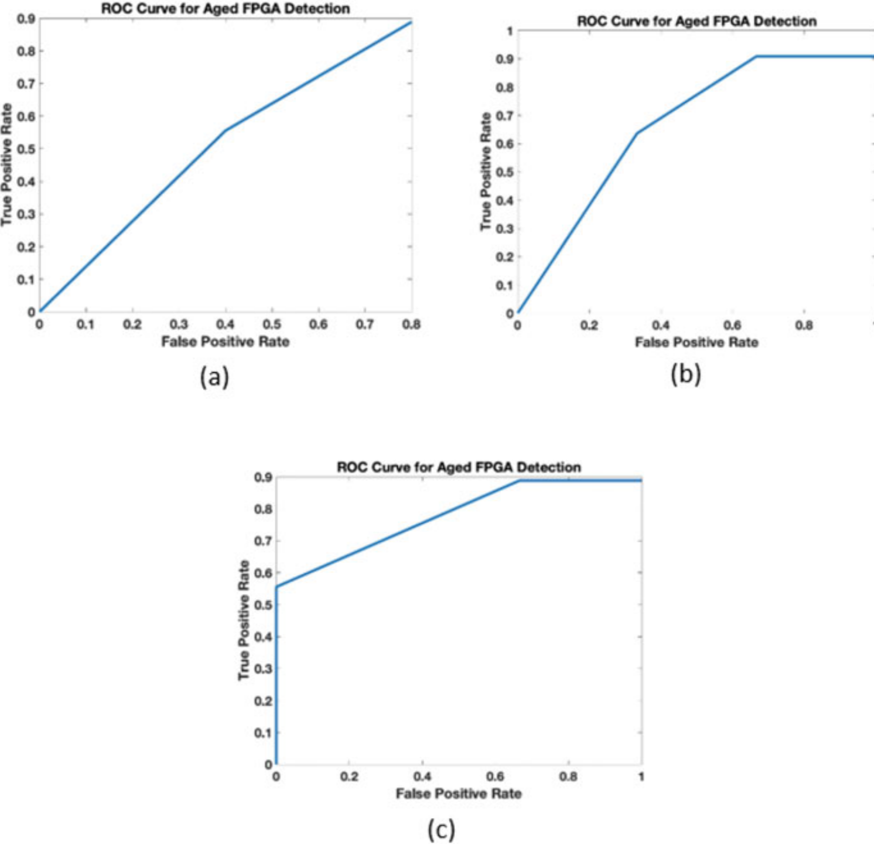


Fig. 4.16 ROC curve obtained from K-means model trained on unused and used ROs. (**a**) 0 Hour and 8 Hour. (**b**) 0 Hour and 12 Hour. (**c**) 0 Hour and 16 Hour

shows the data distribution along with the labels obtained by optimally clustering the frequencies for a specific FPGA used for 0, 8, 12, and 16 h respectively.

It is observed that for most of the scenarios, the number of predicted clusters for unused FPGA is less than that of the used one. Based on the predicted clusters for each FPGA file, the ROC curve is drawn for unused and each of the used (8, 12 and 16 h) FPGAs. Figure 4.16 shows how the ROC curve improves as the FPGA is aged more. The more the age of the FPGA, the better the accuracy. This is because the more an FPGA is used, the frequency distributions of the used ROs shift away from their unused frequency distribution, as shown in Fig. 4.16. This makes it easier for the model to distinguish between used and unused ROs.

4.6 Conclusion

As discussed in this chapter, there are supervised methods (with golden data) and unsupervised methods (without golden data) for detecting recycled FPGAs. In this chapter, readers will learn how to distinguish new from used/recycled FPGAs by detecting partially used, fully used, and spare LUTs in FPGAs exhibiting different aging behaviors.

References

1. Alam, M.M., Tehranipoor, M., Forte, D.: Recycled FPGA detection using exhaustive LUT path delay characterization and voltage scaling. IEEE Trans. Very Large Scale Integr. Syst. **27**(12), 2897–2910 (2019). https://doi.org/10.1109/TVLSI.2019.2933278
2. Amouri, A., Tahoori, M.: A low-cost sensor for aging and late transitions detection in modern FPGAs. In: 2011 21st International Conference on Field Programmable Logic and Applications, pp. 329–335 (2011). https://doi.org/10.1109/FPL.2011.66
3. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Design and analysis of FPGA based PUFs with enhanced performance for hardware-oriented security. ACM J. Emerg. Technol. Comput. Syst. **18**(4), 1–26 (2022)
4. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: Design, implementation and analysis of efficient hardware-based security primitives. In: 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), pp. 198–199. IEEE, Piscataway (2020)
5. Bergman, T.D., Manager, C.P., Liszewski, K.T.: Battelle barricade: a nondestructive electronic component authentication and counterfeit detection technology. In: 2016 IEEE Symposium on Technologies for Homeland Security (HST), pp. 1–6 (2016). https://doi.org/10.1109/THS.2016.7568901
6. Bhunia, S., Tehranipoor, M.: Hardware Security: A hands-on Learning Approach. Morgan Kaufmann, Los Altos (2018)
7. Couch, J., Arkoian, J.: An investigation into a circuit based supply chain analyzer for FPGAs. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–9 (2016). https://doi.org/10.1109/FPL.2016.7577335
8. Dogan, H., Forte, D., Tehranipoor, M.M.: Aging analysis for recycled FPGA detection. In: 2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 171–176 (2014). https://doi.org/10.1109/DFT.2014.6962099
9. Fawcett, T.: ROC graphs: notes and practical considerations for data mining researchers (2003)
10. Guin, U., DiMase, D., Tehranipoor, M.: Counterfeit integrated circuits: detection, avoidance, and the challenges ahead. J. Electron. Test. **30**(1), 9–23 (2014)
11. Guin, U., Zhang, X., Forte, D., Tehranipoor, M.: Low-cost on-chip structures for combating die and IC recycling. In: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2014). https://doi.org/10.1145/2593069.2593157
12. Huang, K., Liu, Y., Korolija, N., Carulli, J.M., Makris, Y.: Recycled IC detection based on statistical methods. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **34**(6), 947–960 (2015). https://doi.org/10.1109/TCAD.2015.2409267
13. Kiamehr, S., Amouri, A., Tahoori, M.B.: Investigation of NBTI and PBTI induced aging in different LUT implementations. In: 2011 International Conference on Field-Programmable Technology, pp. 1–8 (2011). https://doi.org/10.1109/FPT.2011.6132704
14. Leong, C., Semião, J., Teixeira, I.C., Santos, M.B., Teixeira, J.P., Valdés, M., Freijedo, J., Rodríguez-Andina, J., Vargas, F.: aging monitoring with local sensors in FPGA-based designs. In: 2013 23rd International Conference on Field programmable Logic and Applications

15. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inform. Theory **28**(2), 129–137 (1982). https://doi.org/10.1109/TIT.1982.1056489
16. Sadi, M., Winemberg, L., Tehranipoor, M.: A robust digital sensor IP and sensor insertion flow for in-situ path timing slack monitoring in SoCs. In: 2015 IEEE 33rd VLSI Test Symposium (VTS), pp. 1–6 (2015). https://doi.org/10.1109/VTS.2015.7116292
17. Tehranipoor, M.: Emerging Topics in Hardware Security. Springer, Berlin (2021)
18. Tehranipoor, M., Pundir, N., Vashistha, N., Farahmandi, F.: Hardware security primitives. Springer, Berlin (2022)
19. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
20. Tehranipoor, M.M., Guin, U., Forte, D.: Counterfeit Integrated Circuits: Detection and Avoidance. Springer, Berlin (2015)
21. Zhang, X., Xiao, K., Tehranipoor, M.: Path-delay fingerprinting for identification of recovered ICs. In: 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 13–18 (2012). https://doi.org/10.1109/DFT.2012.6378192

Chapter 5
Hardware Trojan Insertion

5.1 Introduction

Integrated circuit (IC) designs are becoming increasingly complex to incorporate more advanced capabilities and speed, inspiring and supporting cloud infrastructure, machine learning applications, and ubiquitous handheld mobile devices. Besides the high complexity, the aggressive time-to-market pressure makes completing the entire system-on-chip (SoC) design in-house an infeasible and time-consuming option for most companies. A common practice in the industry is that the design house will search for available commercial third-party intellectual property (3PIP) cores from others to constitute the SoC implementation with their own IPs for faster development and short-time-to-market advantages [1, 23]. In addition to the chip design phase, the prohibitively high cost of maneuvering advanced nodes (e.g., 5-nm technology) has been motivating the horizontal business model of the semiconductor industry in the past 20 years or so. The fabless organizations need to hand their physical designs, e.g., GDSII file, to offshore contract foundries and facilities for silicon fabrication, packaging, and final testing [2, 18].

Although the IP integration and outsourcing fabrication model significantly saves the monetary and time cost for product provision and iterations, it inevitably introduces hardware attack surfaces which have drawn more and more attention from the community since they are extremely hazardous and rarely patchable, compared to their software counterparts. Out of them, malicious addition and modification on the original IC design, the so-called *hardware Trojan*, are two of the most well-recognized attack vectors. For instance, the attackers might want to create backdoors in the chip designs to steal confidential information from the mission-critical applications built on the target device or hamper the reputations of the original component manufacturers. Besides, hardware Trojan can be inserted at arbitrary stages throughout the device lifetime, e.g., it can be implanted at pre-silicon stages such as register-transfer level (RTL) and gate-level 3PIPs by the untrusted IP design teams [9] or physical layout by the adversarial foundries

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_5

or even at post-silicon stages through advanced physical chip editing techniques like focused ion beam (FIB) [25]. Therefore, it is imperative for researchers and government employees to be mindful of the common modality and insertion manners of hardware Trojan.

The learning objective of this chapter is for trainees to gain experience in the hardware Trojan insertion techniques. Readers will learn how a typical hardware Trojan-infected AES (advanced encryption standard) cryptographic design is constructed at RTL to leak the symmetric key in a rare scenario. Also, this chapter will present the flow mapping of the RTL design on the silicon, i.e., a field-programmable gate array (FPGA) platform using the electronic design automation (EDA) toolchain. In-system debug logic is embedded as well so that readers can *trigger* the malicious functionality during run-time easily. Moreover, the bitstream tampering experiment demonstrates that the adversary can leverage the malleability of a binary FPGA bitstream to enable the hidden malicious circuitry with merely subtle bit flips, further showing the stealthiness of a Trojan.

The rest of the chapter is organized as follows: Section 5.2 provides the background information on hardware Trojans including the present chip design flow and threat model as well as the detailed Trojan structure and taxonomy. Section 5.3 states the provided Trojan-infected AES design and how to implement the RTL design on the FPGA platform and trigger the Trojan at run-time through the on-chip debug infrastructure. Next, Sect. 5.4 shows how to tamper the binary bitstream to enable the hidden Trojan stealthily. Finally, Sect. 5.5 concludes this chapter.

5.2 Hardware Trojan Attacks

In this section, the modern microelectronic device design flow is introduced as to how untrusted entities can implant Trojans. Besides, typical hardware Trojan basics are presented such as its typical structure and taxonomy.

5.2.1 Modern Chip Design Flow and Threat Model

The complexity of SoCs mandates the collaboration between SoC integrators, design-for-test (DFT) teams, and 3PIP vendors to meet the strict time-to-market constraints and procedures and the advantages of competition with peer companies. On the other hand, the prohibitively high cost of maintaining a foundry pushes the shift of semiconductor businesses from traditional integrated device manufacturers (IDM) who design and fabricate chips by themselves to the horizontal model, i.e., fabless companies are only responsible for chip designs, while foundries such as TSMC are focusing on fabrication for minimizing the budget. However, this shift, unfortunately, introduces trustworthiness issues and opens the door for hardware Trojan attacks [13].

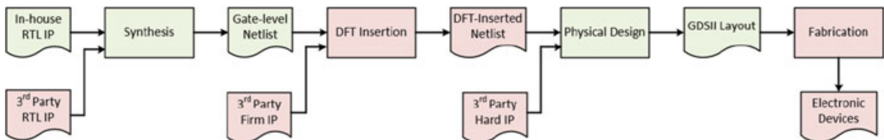


Fig. 5.1 Modern SoC design flow and threat model

The typical modern chip design flow is illustrated in Fig. 5.1 where SoC integrators start with defining the target design specification, e.g., the functionality, performance, and power requirements. To implement the design specification, the functionality is divided as a couple of functional blocks called intellectual property (IP) cores. SoC integrators can design their own in-house IP cores, whereas a more common way to speed up the entire development cycle is to purchase third-party IP (3PIP) cores from 3PIP vendors. When all of IP cores, either designed in-house or from 3PIP vendors, are ready, SoC integrator will use them to constitute the complete RTL description of SoCs. Behavioral simulation is operated by the verification engineers to locate and fix the functionality bugs. After that, electronic design automation (EDA) tools, e.g., Cadence RTL Compiler and Synopsys Design Compiler, will convert the RTL design to the gate-level netlist that comprises low-level information consisting of gates and wires. At this stage, different from the third-party IP cores at the RTL stage, the so-called firm IP at the gate level can be integrated into the netlist as well to accomplish design goals at a more determined performance and functionality because EDA tools usually optimize the RTL design to distinct levels according to customized power, performance, and area constraints during synthesis. Then, design-for-test infrastructure is inserted in the netlist by the (third-party) DFT team for enabling structural testing at the post-silicon stage for observability, controllability, and testing coverage. Next, the DFT-inserted netlist is transformed into a physical layout GDSII design, i.e., transistor-level design. The physical design will incorporate hard 3PIP with the most fixed parameters and go to the outsourced offshore silicon foundry such as Samsung and TSMC for wafer fabrication and die package.

As one can see in Fig. 5.1, there are several untrusted entities/stages in the light red boxes. The adversarial roles they might play in the supply chain for hardware Trojan attacks are discussed as follows:

- **3PIP Vendors**: As mentioned, SoC integrator has to rely on 3PIP vendors to provide a variety of RTL (soft), firm, and hard IP cores to meet the time-to-market requirements. Nevertheless, the reality is that such 3PIP vendors locate across the entire world such that SoC integrators are not able to inspect their integrity. Moreover, as 3PIP cores are typically presented as black boxes, e.g., by following the IEEE 1735 standard [12], SoC integrator cannot look into the details for hardware Trojan detection. Even if some of 3PIP cores are given in plaintext, the complexity of the IP design and the stealthiness of malicious hardware Trojan prevent SoC integrators from successful security closure by using conventional testing and verification techniques.

- **Outsourced DFT Teams**: DFT infrastructure has gained tremendous importance over the past decade for empowering better controllability and observability for post-silicon debugging. This task is usually outsourced to external specialized DFT teams for improved efficiency and less cost. In this way, the untrusted DFT teams have the access to all details of gate-level designs and can insert malicious functionality into the original implementation. As the scan flip-flop insertion during DFT inevitably changes the design topology and the gate-level netlist contains more than millions of gates in most cases, it is extremely hard to detect very few gates of malicious functionality. Even worsen, as the DFT infrastructure can stream in and stream out information from the internals of the silicon, some Trojan can be directly implanted in the debug logics to access security-sensitive on-chip assets such as private keys and user credentials [4, 16].
- **Offshore Foundries**: Establishing and maintaining a silicon foundry come at billions of dollars cost that very few companies can afford. Besides, the advanced technology node such as 3 nm or 5 nm can merely be maneuvered by a few offshore foundries such as TSMC. The fabless SoC integrator will hand all of the design details in the GDSII physical layout to the foundry which might be untrusted and intend to insert malicious hardware Trojans. For example, the empty corners in the chip area can be used to place Trojan gates, while a rouge foundry can wire sensitive signals to the public interface which can be accessed externally. Detecting such Trojans could be extremely difficult since the post-silicon device is a black box from the security inspection perspective and hardware Trojans can be dormant most of the time so the Trojan-infected devices do not manifest any abnormal behaviors.

5.2.2 Hardware Trojan Insertion

Hardware Trojan is a malicious addition or modification of an integrated circuit (IC) [26]. The malicious functionality includes but is not limited to changing the original functionality, compromising the confidentiality of security assets, and causing performance degradation or even denial of service. It is challenging to detect and remove hardware Trojans from the infected design because golden designs are not available for 3PIP cores, netlist, and layout. Besides, most of the hardware Trojans are designed to function in a stealthy way, i.e., manifesting malicious functionality in very rare conditions.

As illustrated in Fig. 5.2, the hardware Trojan structure includes two parts, i.e., trigger and payload [11, 22]. Trojan trigger simultaneously monitors possible stimulus in the circuitry or physical environment. The most common way is that the Trojan trigger monitors multiple internal signals simultaneously and outputs an asserted trigger signal to start malicious activities of the payload if a specific predefined signal pattern is found. In other words, without seeing the pattern on the trigger inputs, the hardware Trojan will remain inactive and hard to be detected. Although a variety of Trojans were proposed [4], they can be classified into

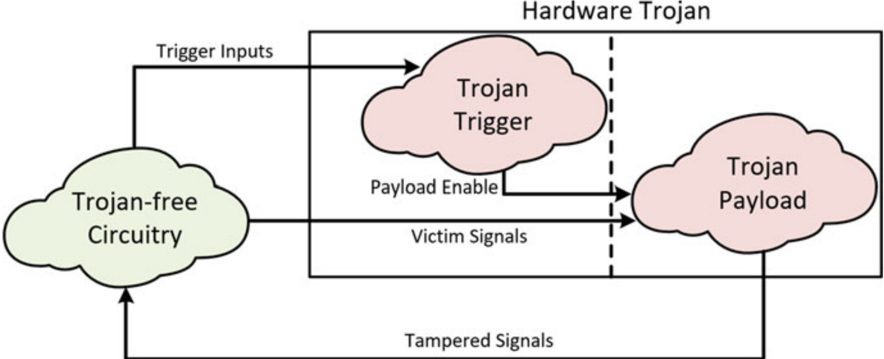


Fig. 5.2 High-level Trojan diagram

combinational Trojans and sequential Trojans according to the trigger mechanism. Note that digital hardware Trojan is focused on in this chapter. The analog hardware Trojans [28] is out of the scope. Figure 5.3a and b shows the simplified models of both combinational and sequential Trojans. Suppose that the trigger inputs are n-bit and the trigger circuitry is an n-bit AND gate. Therefore, the rare trigger condition is defined that only if all of the trigger inputs are 1, the payload enabled signal will be asserted, i.e., the probability of random Trojan triggering is $\frac{1}{2^n}$. To make Trojans even more stealthy, sequential Trojan can use the output of the AND gate as the start flag of a finite state machine (FSM). For example, a straightforward counter can be implemented as a time bomb by activating the Trojan payload at a future moment [24]. As for Trojan payloads, it can be very versatile. For instance, to degrade the device performance and launch denial-of-service attacks, [5] proposes to place a design-independent ring oscillator (RO) on the same silicon as an always-on Trojan. Since RO consists of odd number of inverters, its logical status is in-stable and self-oscillating. Such behaviors will continuously draw currents from the power supply and accelerate the chip aging. The authors of [15] present an off-chip Trojan that can leak the cryptographic keys through a power side channel by using an external capacitor.

Below hardware Trojan taxonomy (see Fig. 5.4) is based on five aspects, i.e., (i) insertion phase, (ii) abstraction level, (iii) activation mechanism, (iv) payload, and (v) location:

- **Insertion Phase**: *Specification* level defines crucial factors of the device including the functionality, performance, and area. A hardware Trojan inserted at specification can, for example, alter the timing and functionality of the final design. *Design* level can be exploited by 3PIP vendors to implant hardware Trojans by injecting malicious circuitry in the IP cores. Rogue foundries can insert hardware Trojans during the *fabrication* phase since they have the access to all of the physical design details in the layout and thus have the ability to alter the final layout by modifying the mask set before wafer fabrication. In the

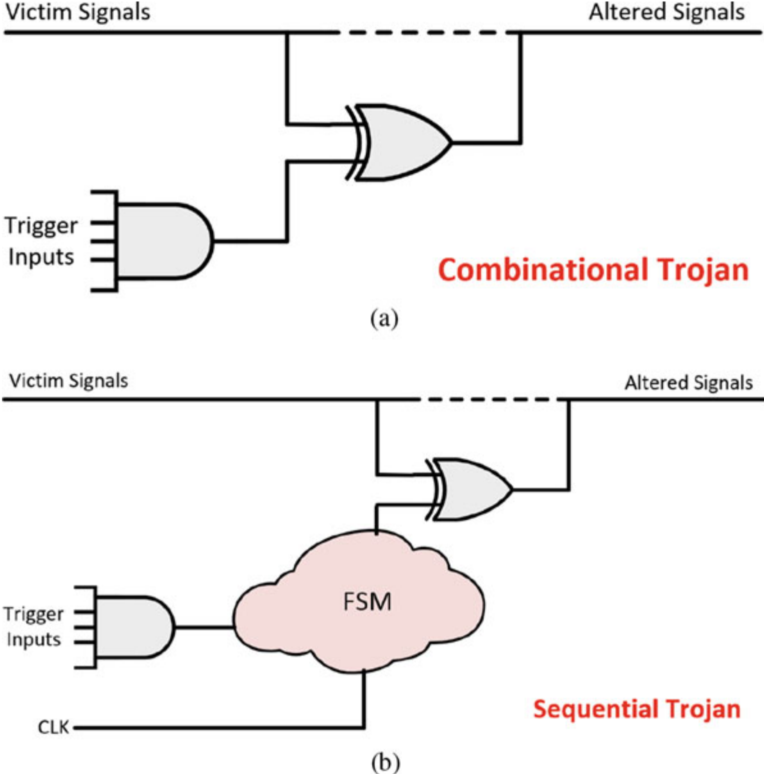


Fig. 5.3 Hardware Trojan models (**a**) Combinational Trojan (**b**) Sequential Trojan

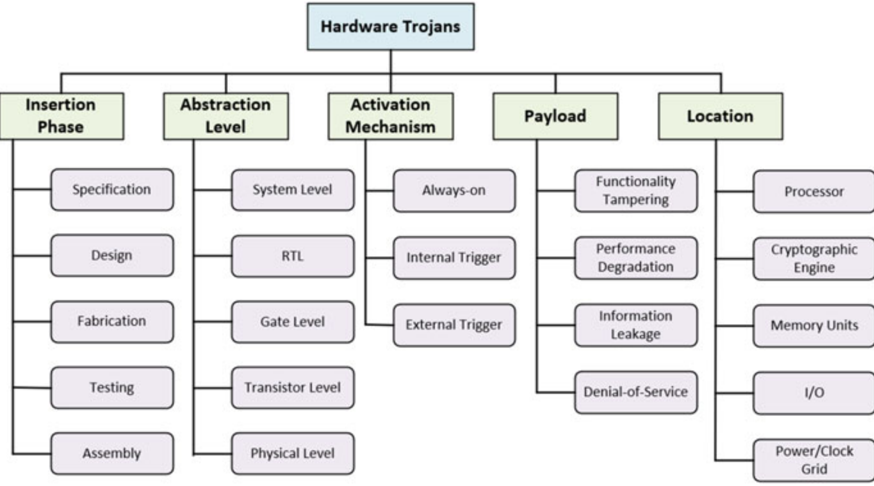


Fig. 5.4 Hardware Trojan taxonomy

testing phase, the adversaries might intentionally generate a set of low-coverage test vectors with low Trojan coverage to help the infected devices escape from detection. People purchase the components and integrate them with the fabricated microelectronic devices on the same printed circuit board (PCB) at the *assembly* phase. System-level Trojans such as BigHack [17] can be inserted at this time to compromise confidentiality, integrity, and availability.

- **Abstraction Level**: Hardware Trojans at *system level* can behave as an additional module altering the functionality of other system modules, interconnects, and communication traffic. *RTL*, *gate-level*, and *transistor-level* hardware Trojans can all be incorporated in the corresponding design files, i.e., RTL code, gate-level netlist, and physical layouts, respectively. As for the *physical level* hardware Trojans, it is implemented by modifying the design parameters. For example, the critical path wire length can be varied to increase the risks of timing failure.
- **Activation Mechanism**: Some Trojans are made to be *always on* such as the RO Trojan in [5]. In contrast, most Trojans tend to be trigger-based in case the malicious behaviors can be discovered easily. *Internally triggered* hardware Trojans rely on internal events like an embedded counter activating the Trojan payload after 2 days since the circuit starts to work. As for *externally triggered* Trojans, they usually depend on user input patterns. For instance, a Trojan targeting a cryptographic module monitors the plaintext input and leaks the key if and only if the plaintext is set to be a specific value.
- **Payload**: Trojan payload for *functionality tampering* can alter the benign behaviors of the original circuit, e.g., an activated Trojan enforces a password-checking circuit to accept an arbitrary string. *Performance degradation* can be brought by a power-hungry Trojan design. It consumes a significant amount of power and thus induce more IR drop to slower the entire device. *Information leakage* can be launched by Trojans by sending sensitive security assets and credentials without the approval of supervisors. In addition, *denial-of-service* Trojan might physically disable or even destroy the microelectronic chips.
- **Location**: Trojans at *random logic* can hinder effective test stimuli generation for detection since understanding such random logic is difficult. Other Trojan locations such as *processing unit*, *cryptographic engine*, *memory units*, and *I/O* can make significant differences even with minor Trojan impacts because they either process or store sensitive information. As for Trojans in *power supply* and *clock grid*, they are more potent to result in timing failures like setup/hold-time violations by causing physical glitches.

5.3 Trojan-Infected Implementation on FPGA

In this section, the FPGA development flow will be introduced because our experiments are carried out on an FPGA. Next, the experimental setup is discussed including the target FPGA platform. Finally, the structure of the Trojan-infected implementation and detailed steps compiling the RTL designs and demonstrating the Trojan's effectiveness are presented.

Fig. 5.5 FPGA development procedure

5.3.1 FPGA Development Flow

Field-programmable gate array (FPGA) has become the most important and popular option for agile hardware prototyping. It is flexible and can be reconfigured by the users in the post-manufacturing phase. The typical development flow of FPGA device involves design entry, synthesis, implementation, and bitstream generation as shown in Fig. 5.5. Design entries can accept a variety of design files. The most intuitive method is drawing the schematics by connecting some predefined functional modules together, whereas a more common and recommended way in industry is to write the behavioral implementation in the form of hardware description language (HDL) like VHDL and Verilog at RTL which is the same as the standard application-specific integrated circuits (ASICs) development flow. During the synthesis stage, the HDL code composed at the design entry stage will be converted into a circuit in the form of netlist by vendor-specific EDA tools like Xilinx Vivado [6] and Intel Quartus [20]. The RTL code is going to be parsed automatically in the EDA environment to check syntax and then optimized to reduce redundant logic per the specified settings. The outcome functionally equivalent netlist contains the mapped logic elements and the connectivity among them as described in the RTL code. The implementation phase will then technology map the logic elements in the netlist to the primitives available in the target FPGA model so that the design could be built on the physical silicon. Also, this step will place and route the primitives on the FPGA layout virtually per the constraints from designers and physical aspects to pursue the closure of the power, area, and performance on the final design. Finally, the placed and routed netlist will be translated to the binary configuration data, the so-called bitstream, and then be downloaded to the target device via an interface like JTAG.

5.3.2 Experimental Setup

In our hands-on experiments, the Nexys A7 board featuring Xilinx Artix 7 FPGA (part number XC7A100T-1CSG324C) is used as shown in Fig. 5.6. With its large, high-capacity FPGA, generous external memories, and collection of USB, Ethernet, and other ports, the Nexys A7 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, a temperature sensor, a MEM digital microphone, a speaker amplifier, and several I/O devices allow the Nexys A7 to be used for a wide range of designs without needing any other components. The Nexys A7 is

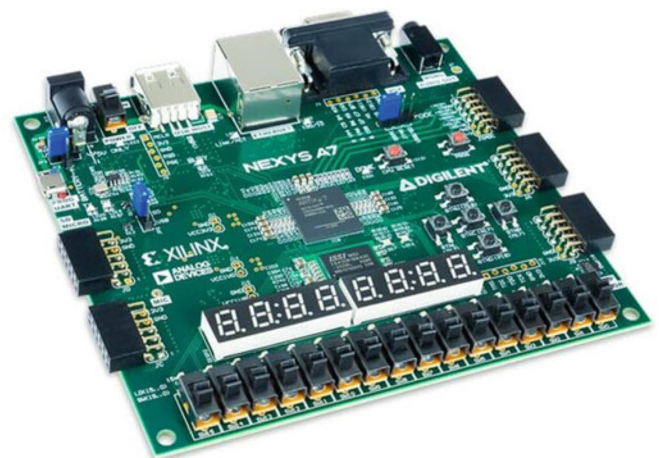


Fig. 5.6 Digilent Nexys A7 board

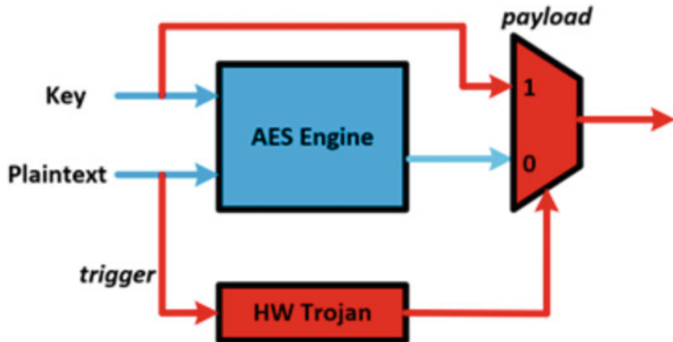


Fig. 5.7 Trojan-infected AES implementation overview

programmed and debugged via the USB connection to the host running Xilinx Vivado 2019.1 and Win10 operating system. The design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch5_hw_trojan_insert/.

5.3.3 Trojan-Infected Design

In this chapter, a hardware Trojan-infected implementation is designed as depicted in Fig. 5.7. The benign implementation is an AES-128 cryptographic engine (blue module) that encrypts the incoming plaintext with the fixed secret key. The AES algorithm is mathematically strong, which means that it is computationally infeasible for an adversary to derive the unknown secret with the controllability of plaintext and observability of ciphertext even if the AES algorithmic or

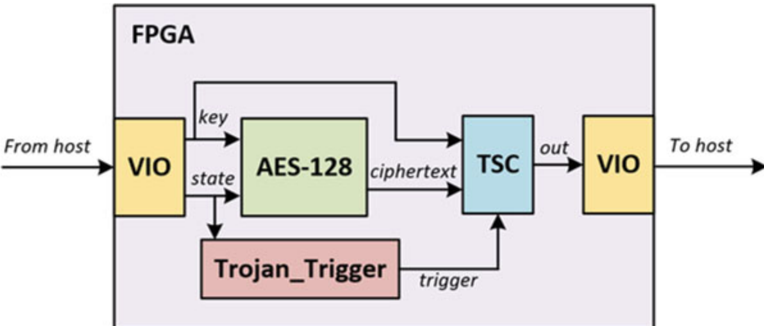


Fig. 5.8 Detailed FPGA implementation diagram

hardware accelerator design details are public. To guarantee the confidentiality of the secret key, it needs to be stored in a tamper-proof memory with strict access control and securely transferred to the hardware AES engine, i.e., the adversary cannot intercept it during the transmission. However, this Trojan-infected implementation can lead to key leakage by bypassing the AES core in a specific rare condition. The trigger part of the inserted hardware Trojan continuously monitors every input plaintext to see if the predefined pattern occurs (128'h0011_2233_4455_6677_8899_AABB_CCDD_EEFF) here. If it is the case, the Trojan will be activated to enable the "1" channel of the output multiplexer to leak the secret key. Otherwise, the "0" channel is enabled to output ciphertext when the Trojan is dormant. This is a typical information leakage Trojan model in cryptographic implementations.

Figure 5.8 presents the detailed implementation. To simplify the interface between the host and FPGA for sending and receiving data, Xilinx Virtual IO (VIO) is instantiated as an IP core by configuring virtual input/output probes. In this way, one can easily set the key and state (plaintext) for the AES-128 engine and observe the output of the Trojan-infected AES. The ciphertext and key signals are multiplexed by the **TSC** module. If the *trigger* signal is asserted, i.e., Trojan is activated by the predefined pattern on the state (plaintext) input port, key input will bypass the AES-128 core to the output which can be captured by VIO. Figure 5.9 shows the design hierarchy of this Trojan-infected implementation in Xilinx Vivado where the naming of functional blocks *aes_128*, *Trojan_Trigger*, *TSC*, and *vio_0* is self-explanatory by corresponding to the blocks in Fig. 5.8. The XDC file (top.xdc) is indispensable for a successful compilation by providing constraints on package pin assignment, targeted clock frequency, implementation settings, etc.

5.3.4 Compiling Target Design and Trigger Trojan

We present detailed step-by-step instructions on compiling and activating the Trojan-infected implementation in Xilinx Vivado:

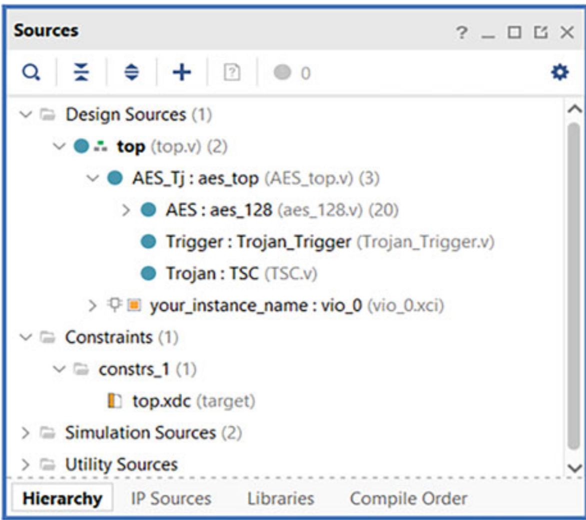


Fig. 5.9 Trojan-infected code hierarchy

Step 1: Compile the RTL Design in Vivado Through Synthesis, Implementation, and Bitstream Generation Xilinx Vivado can compile the loaded RTL designs to the bitstream in an automatic manner. By clicking the *Run Synthesis* as labeled in Fig. 5.10, Xilinx Vivado will initiate a run to transform the RTL design to a synthesized design where a functionally equivalent circuit is generated consisting of generic components. Then, *Run Implementation* will map the generic components to available resources in the specified FPGA model. For example, an individual AND gate cannot be found in most modern FPGA devices. In the implementation phase, Vivado will map the AND gate to an **LUT2** primitive, i.e., a look-up table instance storing the Boolean equation **O = A1 & A2** to implement the AND function. Besides, the low-level routing will be mapped to the programmable interconnect point (PIP) configuration of the FPGA model. *Bitstream Generation* interprets the implemented design to the proprietary binary configuration file which can be downloaded to the FPGA silicon.

Step 2: Downloading the Bitstream to FPGA Click the *Open Target* under *Open Hardware Manager* as shown in Fig. 5.11. Then, one needs to *Open Target* to auto-connect the hardware server, i.e., our host, to the FPGA device instance through the JTAG interface. Next, program the device by right-clicking the target device under the local host and selecting *Program Device* as shown in Fig. 5.12. Then, the *Program Device* pop-up window will set the default bitstream as the newly generated bitstream (*top.bit*). Besides, the design probe file is set to be *top.ltx* which contains the debug VIO core configuration like the virtual input/output probe name and width.

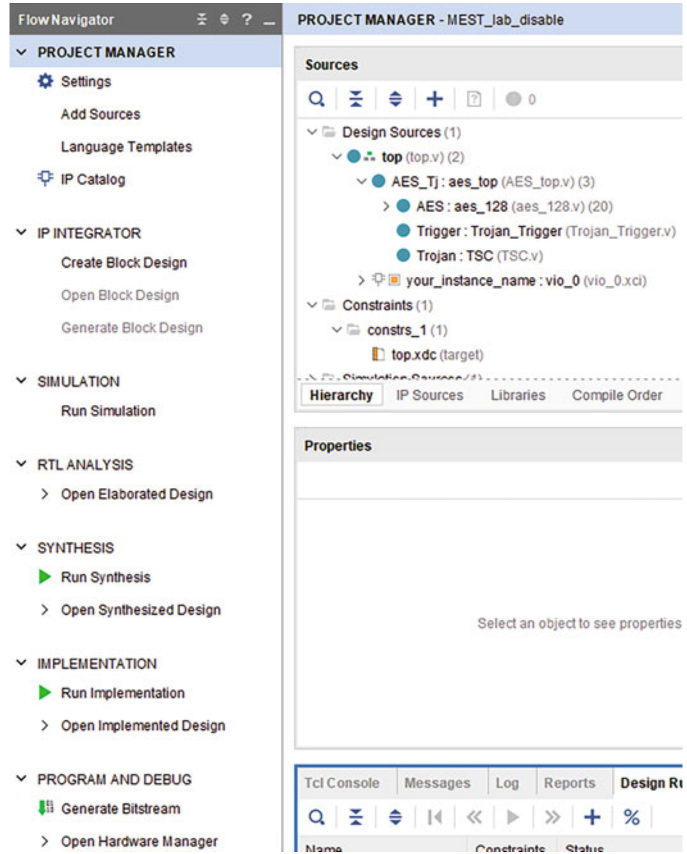


Fig. 5.10 Compiling design in Xilinx Vivado

Step 3: Activating the Trojan on FPGA After downloading the bitstream to FPGA, the VIO debug core dashboard will appear in Vivado as illustrated in Fig. 5.13. Note that the reset input pin of the implementation is bonded to the DIP switch SW5 which needs to be down to dessert the signal. In Fig. 5.13, key is 128'hAAAA_AAAA_AAAA_AAAA_BBBB_BBBB_BBBB_BBBB, while the state (plaintext) is 128'h0000_0000_0000_0000_0000_0000_0000_FFFF. As the plaintext input is not the triggering pattern, the Trojan still remains dormant, and the AES-128 core will perform ten-round *SubByte*, *MixColumn*, *ShiftRow*, and *AddRoundKey* operations on the plaintext to produce the ciphertext output. In contrast, if the 128-bit state is set to 128'h0011_2233_4455_6677_8899_AABB_CCDD _EEFF as depicted in Fig. 5.14, Trojan is activated such that the encryption key is leaked through the output port. In this way, the effectiveness of the inserted hardware Trojan is demonstrated on the FPGA device at run-time.

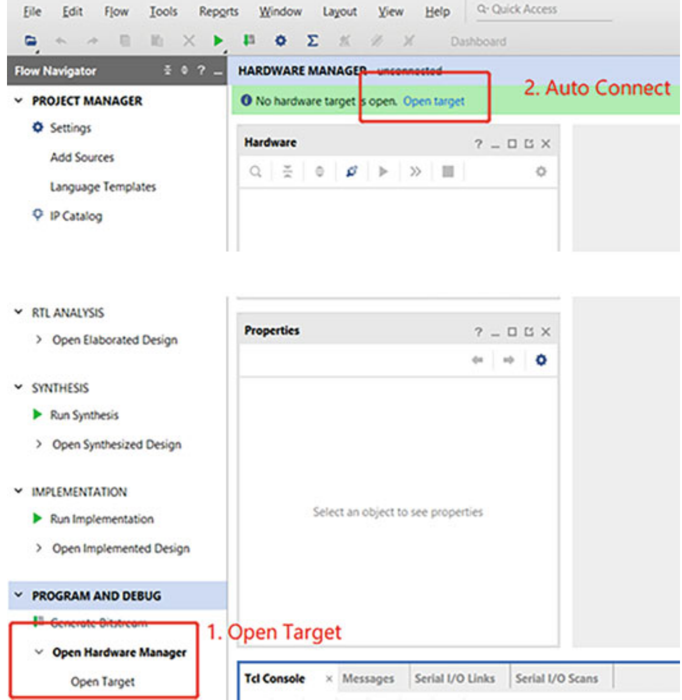


Fig. 5.11 Open and connect the FPGA target

5.4 Bitstream Tampering for Trojan Triggering

In this section, FPGA bitstream tampering attacks to enable Trojan triggers will be presented. First, FPGA bitstream preliminaries and tampering attacks are introduced. Next, experimental files and steps are detailed.

5.4.1 FPGA Bitstream Format Preliminaries

FPGA bitstream is crucial because it determines the FPGA behaviors at the post-configuration phase. Figure 5.15 depicts the high-level format of Xilinx bitstream. It mostly starts with human-readable content such as design name and timestamp. However, the content will be discarded by the hardware FPGA configuration engine during the bitstream loading stage. The beginning of the configuration data stream is the sync word for the alignment of the subsequent data. The following header commands will read/write important registers. For example, one can write the WBSTAR register for multi-boot configuration [10]. Also, writing the IDCODE register to inform the engine the target device model of the incoming bitstream;

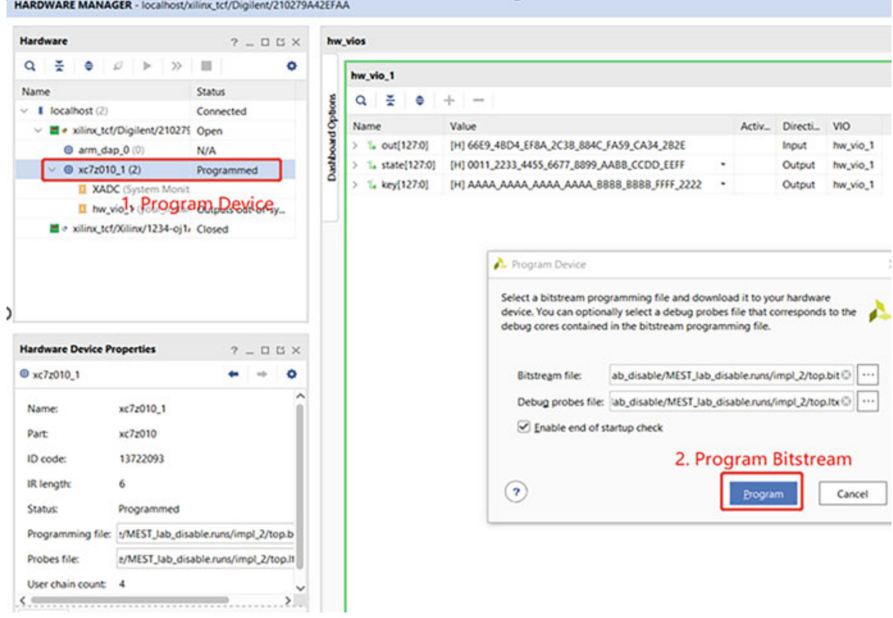


Fig. 5.12 Download bitstream to FPGA

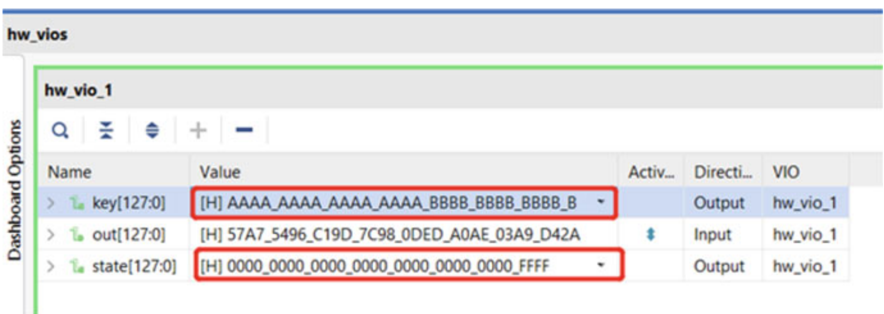


Fig. 5.13 Ciphertext on the outport when the Trojan is dormant

Name	Value	Activity	Direction	VIO
key[127:0]	[H] AAAA_AAAA_AAAA_AAAA_BBBB_BBBB_BBBB_BBBB		Output	hw_vio_1
out[127:0]	[H] AAAA_AAAA_AAAA_AAAA_BBBB_BBBB_BBBB_BBBB		Input	hw_vio_1
state[127:0]	[H] 0011_2233_4455_6677_8899_AABB_CCDD_EEFF		Output	hw_vio_1

Fig. 5.14 Key leaked to the outport when the Trojan is activated

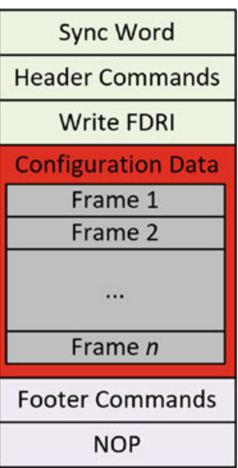


Fig. 5.15 High-level FPGA bitstream format

the device will reject the bitstream if the IDCODE does not match. The most crucial operations are **Write FDRI** which write the configuration frames to the on-chip SRAM determining the FPGA functionality. The address of the on-chip configuration memory the frames flow to is determined by the frame address register (FAR) which is auto-incremented by default. The footer commands include cyclic redundancy check (CRC) checksum and end with a couple of NOP (no operation) commands.

Although bitstream format can be understood at a high level from official documents [27], FPGA vendors are reluctant to reveal the bitstream format documenting the mapping between configuration data and FPGA primitive functionality. FPGA bitstream reverse engineering techniques [3] have been developed to retrieve such information to assist applications like hardware Trojan detection [14, 29] or insertion [7]. For instance, [29] proposes a high-accuracy FPGA reverse engineering for recovering the netlist from the binary bitstream and applies an unsupervised machine learning solution to detect suspicious Trojan signals. Conversely, the retrieved bitstream format can be used for bitstream tampering as well. Swierczynski et al. [21] first reverse engineer the configuration bits of look-up tables (LUTs) and then target AES accelerator on FPGA. As the AES hardware typically relies on the LUTs to store the S-box content, tampering with LUT content can inject faults to steal keys through differential fault analysis (DFA). Even if security mechanisms like bitstream encryption are enabled, such bitstream tampering is still feasible since AES-CBC allows causing bit flips on the target block by corrupting the previous one. Moreover, [8] hacks the Xilinx 7-series bitstream engine and discloses the underlying vulnerabilities which can be exploited to decrypt the ciphertext bitstream by using the FPGA as an oracle [19].

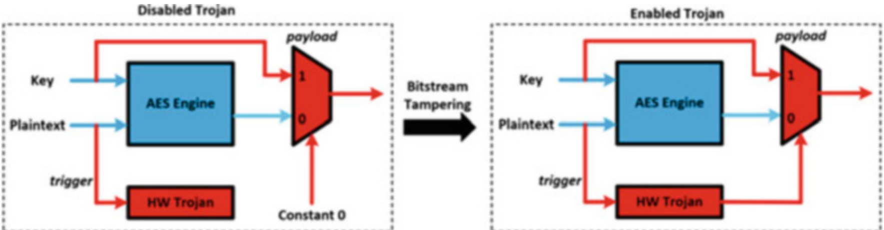


Fig. 5.16 Enabling hardware Trojan through bitstream tampering

5.4.2 Bitstream Tampering Enabling Trojan Trigger

In this experiment, Trojan trigger is disabled by disconnecting the trigger signal from the multiplexer in the original implementation as depicted in the left part of Fig. 5.16. As the multiplexer SEL port is fixed at the constant 0, the Trojan can never be activated even if the trigger part observes the predefined pattern on the plaintext input. However, by tampering with the design at the bitstream level, the low-level routing can be altered to revert the hardware Trojan functionality, i.e., the trigger signal will be connected to the multiplexer.

There are supposed to be five files in the ./nexys/ folder. The details and observations on FPGA are introduced as follows:

- *AES_trigger_disabled.bit*: This bitstream encodes the Trojan-infected AES implementation with the disabled Trojan trigger. To generate the bitstream, one can modify the driver component configuration of the trigger signal in the original infected design. A feasible way is to uncomment the command *set_property INIT 4'h0 [get_cells AES_Tj/Trigger/Tj_Trig_reg_i_3]* in the constraint file *top.xdc* (Sources panel → constraints → constrs_1 → top.xdc) and rerun the flow through synthesis to bitstream generation. As a matter of fact, the command erases the configuration bits of the driver LUT of the trigger signal to be all 0s. In other words, the output of LUT is fixed to 0 to all combinations of inputs. This results in minimal changes in the implemented design where only the content of a LUT is cleared. If one changes the functionality at RTL, the entire design floor plan is likely to deviate which would make the bitstream tampering less intuitive and clear. As shown in Fig. 5.17, downloading this bitstream along with the same probe file *top.ltx* because the VIO configuration is identical. The VIO dashboard in Fig. 5.18 illustrates that even if the predefined pattern 128'h0011_2233_4455_6677_8899_AABB_CCDD_EEFF occurs on the plaintext input, the Trojan is still inactive as the output is not the key value but the AES-128 ciphertext instead, which is different from the original Trojan-infected behaviors.
- *AES_trigger_enabled_ref.bit*: This bitstream contains the original Trojan-infected AES implementation with an enabled trigger for reference. The functionality has been detailed in Sect. 5.3.3.

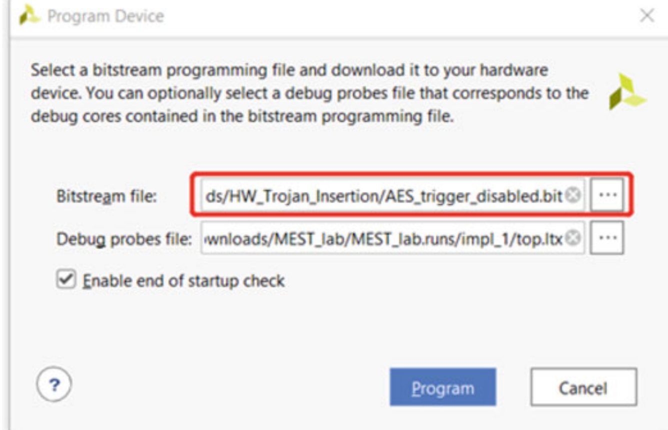


Fig. 5.17 Download AES trigger-disabled bit (disabled Trojan)

Name	Value	Activity	Direction	VIO
key[127:0]	[H] AAAA_AAAA_AAAA_AAAA_BBBB_BBBB_BBBB_BBBB		Output	hw_vio_1
out[127:0]	[H] 6C63_19E2_8368_1D2C_AFB1_4D60_BA2F_84D9		Input	hw_vio_1
state[127:0]	[H] 0011_2233_4455_6677_8899_AABB_CCDD_EEFF		Output	hw_vio_1

Fig. 5.18 Key cannot be leaked even if the predefined pattern occurs

```
C:\Users\zht24\Downloads\HW_Trojan_Insertion>python tampering.py
Bistream tampering started!
Bistream tampering finished! The Trojan trigger has been enabled in the AES_trigger_enabled.bit file.

C:\Users\zht24\Downloads\HW_Trojan_Insertion>
```

Fig. 5.19 Running Python script to tamper with the bitstream to enable Trojan

- *Tampering.bit*: This binary file has the same length of other bitstreams which documents the critical bits in AES_trigger_disable.bit that needs to be flipped during the tampering procedure.
- *Tampering.py*: The python script can tamper the AES_trigger_disabled.bit to activate the disabled Trojan trigger. Run the python script tampering.py to generate the bitstream AES_trigger_enabled.bit by XORing AES_trigger_disabled.bit and tampering.bit. To run the python script, just cd to the directory (e.g., ./nexys/), and type "python tampering.py." The successful bitstream tampering printout is shown in Fig. 5.19. By examining the content of *tampering.bit* using HexEditor (see Fig. 5.20), only few configuration bits need to be flipped to enable the Trojan at the bitstream level. The reason is that only the driver LUT content is erased to fix the Trojan multiplexer to channel 0 so the bitstream tampering procedure essentially converts the LUT back to the functional status.

```
002a55c0  00 cc cc 00 00 00 00 00 00 00 00 00 00 00 00 00
002a55d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002a55e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002a55f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002a5600  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002a5610  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Fig. 5.20 A portion of tampering bit file

hw_vio_1

Name	Value	Activity	Direction	VIO
key[127:0]	[H] AAAA_AAAA_AAAA_AAAA_BBBB_BBBB_BBBB_BBBB		Output	hw_vio_1
out[127:0]	[H] AAAA_AAAA_AAAA_AAAA_BBBB_BBBB_BBBB_BBBB		Input	hw_vio_1
state[127:0]	[H] 0011_2233_4455_6677_8899_AABB_CCDD_EEFF		Output	hw_vio_1

Fig. 5.21 Trojan can be activated in the AES trigger-enabled bit file

- *AES_trigger_enabled.bit (to be generated)*: One needs to run the Python script tampering.py to generate the bitstream. The content in this binary file should be equivalent to the AES_trigger_enabled_ref.bit. The bitstream contains the enabled Trojan trigger as a consequence of the bitstream tampering. By programming the FPGA device with this bitstream, it is observed in Fig. 5.21 that Trojan can be activated when the specific pattern appears on the state (plaintext) input to leak the key input through the output port.

5.5 Conclusion

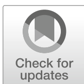
Hardware Trojan threats are a long-standing concern in the hardware security community. It is imperative to learn about its attributes and features. In this chapter, the horizontal chip design cycle is introduced at first to give insights into why Trojan insertion is feasible in the real world. Next, Trojan background including its structure and taxonomy is detailed. In order to give an intuitive understanding of Trojan insertion, a Trojan-infected AES engine is provided to be implemented on an FPGA platform which can be activated/deactivated at run-time to leak the sensitive security key or not. Also, FPGA bitstream tampering, as an advanced attack technique, is experimentally demonstrated on enabling the hidden Trojan trigger at the binary level.

References

1. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms (2022). arXiv preprint arXiv:2204.07909
2. Anandakumar, N.N., Rahman, M.S., Rahman, M.M.M., Kibria, R., Das, U., Farahmandi, F., Rahman, F., Tehranipoor, M.M.: Rethinking watermark: providing proof of IP ownership in modern SoCs (2022). Cryptology ePrint Archive
3. Benz, F., Seffrin, A., Huss, S.A. Bil: a tool-chain for bitstream reverse-engineering. In: 22nd International Conference on Field Programmable Logic and Applications (FPL), pp. 735–738. IEEE, Piscataway (2012)
4. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Los Altos (2018)
5. Chakraborty, R.S., Saha, I., Palchaudhuri, A., Naik, G.K.: Hardware trojan insertion by direct modification of FPGA configuration bitstream. IEEE Design Test **30**(2), 45–54 (2013)
6. Churiwala, S., Hyderabad, I. (2017). Designing with xilinx® FPGAS. In: Circuits &Systems. Springer, Berlin (2017)
7. Ender, M., Swierczynski, P., Wallat, S., Wilhelm, M., Knopp, P.M., Paar, C.: Insights into the mind of a trojan designer: the challenge to integrate a trojan into the bitstream. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference, pp. 112–119 (2019)
8. Ender, M., Moradi, A., Paar, C.: The unpatchable silicon: a full break of the bitstream encryption of xilinx 7-series {FPGAs}. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 1803–1819 (2020)
9. Farahmandi, F., Huang, Y., Mishra, P.: Trojan localization using symbolic algebra. In 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 591–597. IEEE, Piscataway (2017)
10. Gören, S., Ozkurt, O., Yildiz, A., Ugurdag, H.F.: FPGA bitstream protection with PUFs, obfuscation, and multi-boot. In: 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–2. IEEE, Piscataway (2011)
11. Giri, N., Anandakumar, N.N.: Design and analysis of hardware Trojan threats in reconfigurable hardware. In: 2020 Inter. Conf. on Emerging Trends in Infor. Tech. and Engineering (IC-ETITE), pp. 1–5. IEEE, Piscataway (2020)
12. Guin, U., Asadizanjani, N., Tehranipoor, M.: Standards for hardware security. GetMobile: Mobile Comput. Commun. **23**(1), 5–9 (2019)
13. Kelly, S., Zhang, X., Tehranipoor, M., Ferraiuolo, A.: Detecting hardware Trojans using on-chip sensors in an ASIC design. J. Electron. Testing **31**(1), 11–26 (2015)
14. Li, M., Davoodi, A., Tehranipoor, M.: A sensor-assisted self-authentication framework for hardware Trojan detection. In: 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1331–1336. IEEE, Piscataway (2012)
15. Lin, L., Burleson, W., Paar, C.: Moles: malicious off-chip leakage enabled by side-channels. In: 2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers, pp. 117–122. IEEE, Piscataway (2009)
16. Manivannan, S., Anandakumar, N.N., Nirmala Devi, M.: Key retrieval from AES architecture through hardware Trojan horse. In: International Symposium on Security in Computing and Communication, pp. 483–494 (2018)
17. Mehta, D., Lu, H., Paradis, O.P., MS, M.A., Rahman, M.T., Iskander, Y., Chawla, P., Woodard, D.L., Tehranipoor, M., Asadizanjani, N.: The big hack explained: detection and prevention of PCB supply chain implants. ACM J. Emerg. Technol. Comput. Syst. **16**(4), 1–25 (2020)
18. Rahman, M.T., Rahman, M.S., Wang, H., Tajik, S., Khalil, W., Farahmandi, F., Forte, D., Asadizanjani, N., Tehranipoor, M.: Defense-in-depth: a recipe for logic locking to prevail. Integration **72**, 39–57 (2020)

19. Salmani, H., Tehranipoor, M., Karri, R.: On design vulnerability analysis and trust benchmarks development. In: 2013 IEEE 31st International Conference on Computer Design (ICCD), pp. 471–474. IEEE, Piscataway (2013)
20. Samokhvalov, Y., Toliupa, S., Buchyk, S., Shtanenko, S.: Design of robotic systems in the basis of Sapr Intel Quartus Prime. In: 2021 IEEE 3rd International Conference on Advanced Trends in Information Theory (ATIT), pp. 179–183. IEEE, Piscataway (2021)
21. Swierczynski, P., Becker, G.T., Moradi, A., Paar, C.: Bitstream fault injections (BiFI)–automated fault attacks against SRAM-based FPGAs. IEEE Trans. Comput. **67**(3), 348–360 (2017)
22. Tehranipoor, M., Koushanfar, F.: A survey of hardware trojan taxonomy and detection. IEEE Design Test Comput. **27**(1), 10–25 (2010)
23. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
24. Wang, X., Narasimhan, S., Krishna, A., Mal-Sarkar, T., Bhunia, S.: Sequential hardware trojan: Side-channel aware design and placement. In: 2011 IEEE 29th International Conference on Computer Design (ICCD), pp. 297–300. IEEE, Piscataway (2011)
25. Wang, H., Shi, Q., Nahiyan, A., Forte, D., Tehranipoor, M.M.: A physical design flow against front-side probing attacks by internal shielding. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **39**(10), 2152–2165 (2019)
26. Xiao, K., Forte, D., Jin, Y., Karri, R., Bhunia, S., Tehranipoor, M.: Hardware Trojans: lessons learned after one decade of research. ACM Trans. Design Autom. Electron. Syst. **22**(1), 1–23 (2016)
27. Xilinx, I: 7 series FPGAs configuration user guide (UG470) (2018). https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf
28. Yang, K., Hicks, M., Dong, Q., Austin, T., Sylvester, D.: A2: analog malicious hardware. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 18–37. IEEE, Piscataway (2016)
29. Zhang, T., Wang, J., Guo, S., Chen, Z.: A comprehensive FPGA reverse engineering tool-chain: from bitstream to RTL code. IEEE Access **7**, 38379–38389 (2019)

Chapter 6
Hardware Trojan Detection



6.1 Introduction

Over the past decades, the advance of semiconductor technology is mind-boggling by following Moore's law that the density of transistors doubles every 18 months. On the other hand, the more and more complicated chip designs and the prohibitively high cost of maintaining a foundry inspired the industry to transform from the conventional integrated device manufacturing (IDM) mode, where a company takes care of everything of their products including design, and validation, and fabrication, to the horizontal business model requiring collaborations from entities across the globe [3, 26]. A such shift can significantly boost the capabilities of most fabless companies by eliminating the monetary and technical cost of maintaining a prohibitively expensive foundry. However, the convoluted supply chain also creates *trust* issues between involved entities including intellectual property (IP) vendors, system-on-chip (SoC) integrators, design-for-test (DFT) engineers, and fabrication facilities [4, 17]. Hardware Trojan, a concept of malicious addition or modification to the original chip designs, emerges as a prominent attack vector because most entities can have the motivations and capabilities to tamper with the pre-silicon designs [25]. On the other hand, the modality of a particular Trojan can be very versatile in terms of the detailed structure, locations, and insertion phases; how to thwart the malicious circuitry effectively is a long-standing research problem in both industry and academia [30].

The mainstream countermeasures against hardware Trojan threats so far can be divided into two categories, i.e., Trojan detection and Trojan prevention. Trojan detection is the most common methodology for security and trust verification, aiming to detect and diagnose malicious functionality from the pre-silicon or post-silicon designs without the support of any dedicated hardware circuitry [14]. In contrast, Trojan prevention techniques rely on the design-for-trust infrastructure built inside the chip covering functional tests, run-time monitoring, and logic obfuscation. In this chapter, we mostly focus on *Trojan detection* techniques since they are

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_6

prevalent, straightforward, and relatively low-cost. As mentioned, Trojan detection can be performed at both pre-silicon and post-silicon stages, indicating a broad spectrum of possible solutions. For example, one can use statistical approaches to identify the signals with low transition probabilities in a Trojan-infected netlist by utilizing the fact that hardware Trojan trigger is typically designed to be stealthy, i.e., the trigger signal can only be activated in a very rare condition to avoid easy detection. However, one-size-fits-all techniques are still not available to detect all kinds of Trojan logic universally. Therefore, it is imperative to explore the design space of detection solutions to cover different inserted Trojan implementations.

The objective of this chapter is for readers to learn about state-of-the-art Trojan detection techniques and gain experience in hardware Trojan detection. This chapter introduces the concept of hardware Trojan detection techniques and representatives at different phases. Moreover, this chapter presents a hands-on experiment for showcasing hardware Trojan detection in a cryptographic engine, i.e., advanced encryption standard (AES), using commercial formal verification tools at the register-transfer level (RTL). By creating *security properties* to regulate the behaviors of the AES implementation, the counterexample violating the security property will explicitly point out the trigger pattern activating the inserted Trojan.

The rest of the chapter is organized as follows: Section 6.2 provides the background information on hardware Trojan detection at both pre-silicon and post-silicon stages including code coverage analysis, formal verification, and side-channel-based detection. Section 6.3 provides hands-on experience in hardware Trojan detection, and an experiment is presented to demonstrate how to use a commercial EDA formal verification tool to effectively identify the Trojan trigger sequence of an infected AES implementation in RTL. Finally, Sect. 6.4 concludes this chapter.

6.2 Hardware Trojan Detection

The fundamentals of hardware Trojan will be introduced in this part first. Next, both pre-silicon and post-silicon hardware Trojan detection solutions will be discussed.

6.2.1 Overview of Hardware Trojan

Figure 6.1 illustrates the possible stages of hardware Trojan insertion in the development cycle of hardware designs. Specifically, hardware Trojans can be undocumented modifications in the specification phase at the very beginning. The adversary can be one of the members of the committee to intentionally create inconsistencies in descriptions, add undesired functionality, or reduce the margin of performance, power, and area. As such, all the subsequent effort in the development procedure will be unknowingly employed to hide the hardware Trojan as the design

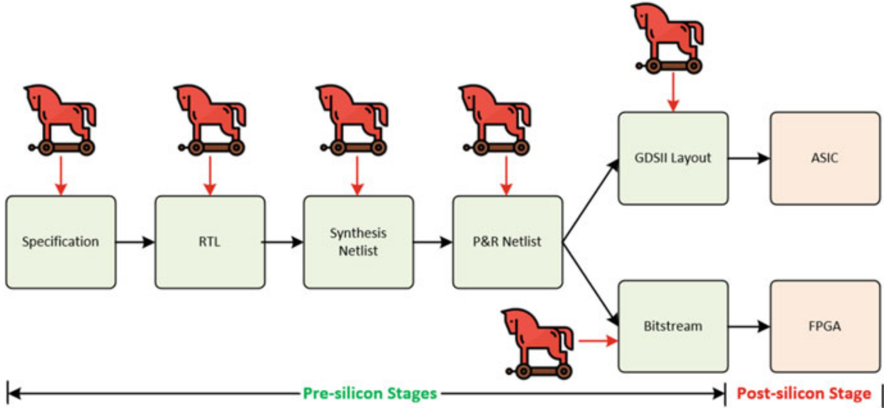


Fig. 6.1 Hardware Trojan insertion is possible at any stage of the ASIC/FPGA development cycle

should follow the defined specification. Next, the RTL phase aims to map the abstract specification to a specific hardware description language (HDL) like VHDL or Verilog. The adversary in the RTL design team can stealthily implant hardware Trojans as well by inserting only a few lines of code as a new branch or a concurrent statement. Note that due to the requirement of short time to market, more and more complicated designs like an SoC prefer purchasing the encapsulated IP solution from third-party teams. Such IP cores are provisioned with low controllability and observability, making them a good place for Trojan insertion. After that, the RTL designs will be compiled into a synthesized netlist consisting of general logic primitives and then placed and routed to be the **P&R** netlist. The RTL-to-netlist conversion is mostly taken by the commercial electronic design automation (EDA) tool where the adversaries can put malicious gates and routing [2]. From the placed and routed netlist, the development procedure will diverge to different steps depending on whether the final silicon platform is a field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC). As for ASIC, the netlist will be packaged to be GDSII which is the package of all design details at the physical layout including not only the logic functionality but also the physical implementations like metal layers and power distribution network. However, the (rogue) foundry can therefore be capable of manipulating the implementation for Trojan implantation with maximal flexibility. When it comes to the FPGA platform which is a configurable device without any particular functionality until being programmed with the binary configuration data, the so-called bitstream, bitstream-level tampering is a feasible method for Trojan insertion [6] and hard to be detected because an FPGA bitstream is generally a series of binary 0s and 1s while its format is architecture-specific and proprietary [33]. After ASIC tapeout or bitstream downloading, hardware Trojans inserted in any previous stages will be ultimately mapped into the post-silicon phase, which can be activated at a future moment for illegitimate applications like information leakage and denial of service (DoS).

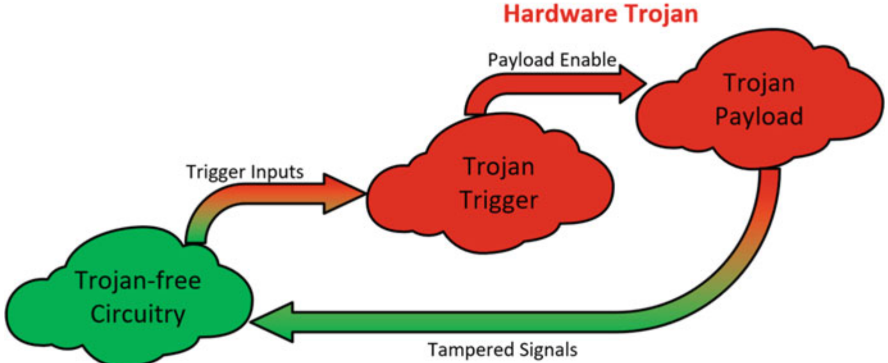


Fig. 6.2 High-level view of a trigger-based hardware Trojan

There are generally two types of hardware Trojans regarding the activation mechanisms, i.e., *always-on* and *trigger-based*. The always-on Trojans are straightforward variants working all the time during the entire lifetime of the infected devices. Such Trojans can lead to severe performance degradation and/or lifetime reduction on the targets. The trigger-based Trojan as depicted in Fig. 6.2 is much more common because it continuously monitors the trigger input signals in the Trojan-free circuitry. Only if the trigger signals manifest a predefined pattern, the subsequent malicious payload can become active. Therefore, the trigger-based Trojans are much stealthier than their always-on counterparts. The signals affected by the activated hardware Trojans (tampered signals) will behave as regulated by the malicious functionality to inject intentional faults back into the Trojan-free part and/or leak the sensitive information directly.

6.2.2 *Pre-silicon Hardware Trojan Detection*

As depicted in Fig. 6.3, existing Trojan detection approaches can be performed at both pre-silicon and post-silicon stages. Pre-silicon hardware Trojan detection aims to identify the malicious functionality from the third-party IP cores, which can be categorized into formal verification, code coverage analysis, logic testing, structural analysis, and functional analysis [5, 15].

6.2.2.1 Code Coverage Analysis

Code coverage refers to a metric assessing how effectively a test bench can exercise the target design, i.e., the percentage of code lines that have been executed. Generally, EDA tools like Synopsys VCS can perform code coverage analysis highlighting which nets in the design under test are toggled or never toggled. Given

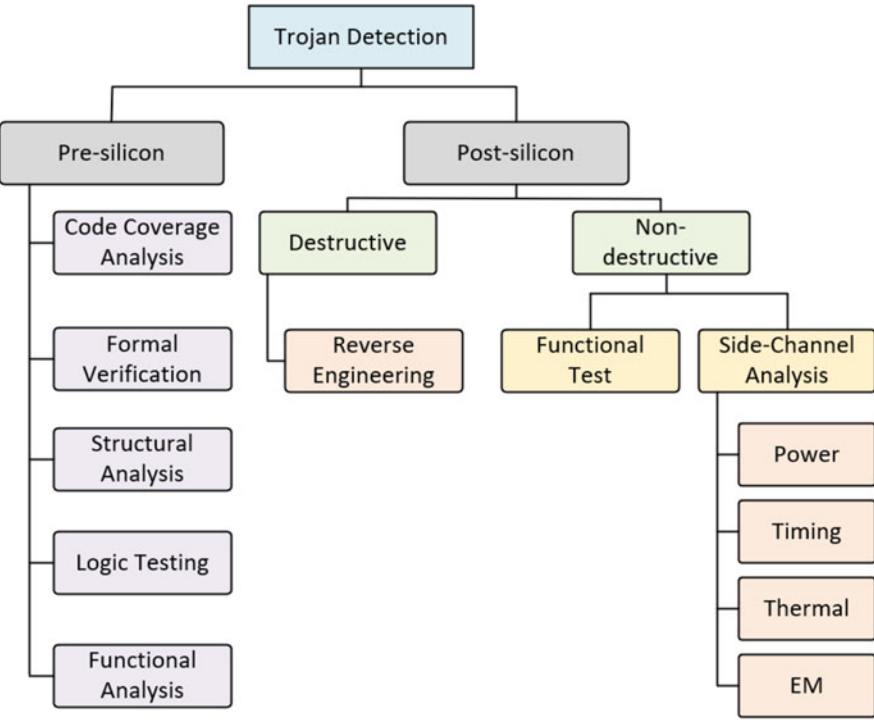


Fig. 6.3 Taxonomy of existing hardware Trojan detection approaches [5]

the stealthy nature of trigger-based hardware Trojans, the adversaries tend to select rare conditions as the trigger point to circumvent conventional testing solutions. Therefore, the nets with low transition probability are their preferable choices which are also likely to be identified as suspicious signals using the code coverage analysis. However, since code coverage analysis is originally proposed for test bench evaluation, it might not be effective in detecting Trojan scenarios; the adversary can easily come up with a circuit covered by the provided test bench, whereas the malicious circuitry is not activated as depicted in Fig. 6.4a. If the test set comprises the control states (ctl signals **Ctl(0)** and **Ctl(1)**) of **00**, **01**, and **10**, the three lines of HDL code in Fig. 6.4a will be covered, whereas the output is always *Good* instead of the malicious *Attack* input. To overcome the limitations of conventional code coverage analysis, unused circuit identification (UCI) technique [10] has been proposed to locate the suspicious lines of RTL code that does not affect the outputs during simulation, which are considered as a part of malicious circuitry. More specifically, UCI first creates a data-flow graph of the target circuit implementation, e.g., the data flow of the multiplexer circuit as presented in Fig. 6.4b. The data-flow graph contains all direct (e.g., the connectivity between *X* and *Out*) and indirect (e.g., the connectivity between *Good* and *Out*) dependencies. Next, based on the test bench, UCI will reduce the size of the data-flow dependency graph to find the

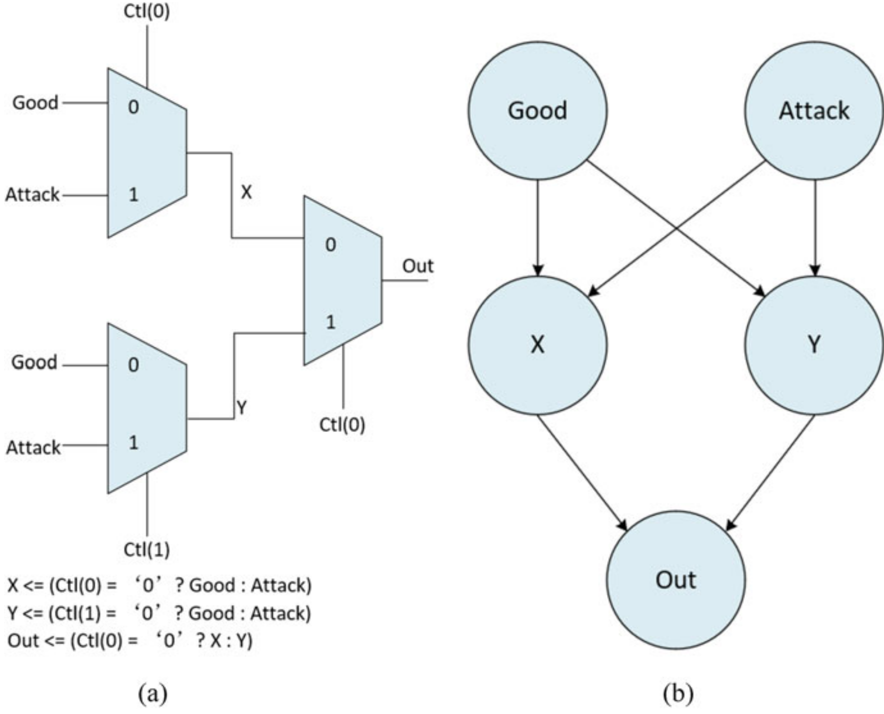


Fig. 6.4 Unused circuit identification (UCI) approach for hardware Trojan detection and removal (**a**) Code coverage analysis failing to detect the malicious functionality in the multiplexer circuit (**b**) UCI technique: generating data-flow graph for the multiplexer circuit

pair where the intermediate logic does not affect it. To provide readers with a more intuitive understanding, the UCI analysis on the multiplexer circuitry in Fig. 6.4a is detailed as follows [10]:

- UCI establishes the data-flow graph of the multiplexer circuit as illustrated in Fig. 6.4b where the set of dependency pairs is {(*Good*, *X*), (*Attack*, *X*), (*Good*, *Y*), (*Attack*, *Y*), (*Good*, *Out*), (*Attack*, *Out*), (*X*, *Out*), and (*Y*, *Out*)}
- By applying **00** on the signals **Ctl(0)** and **Ctl(1)**, *Out* is *Good*, *X* is *Good*, and *Y* is *Good*. This can help remove their exclusive items, i.e., (*Attack*, *Out*), (*Attack*, *X*), and (*Attack*, *Y*), from the set of dependencies.
- By applying **01** on the signals **Ctl(0)** and **Ctl(1)**, *Out* is *Good*, *X* is *Good*, *Y* is *Attack*. This removes, i.e., (*Y*, *Out*) and (*Good*, *Y*) from the set of dependencies.
- By applying **10** on the signals **Ctl(0)** and **Ctl(1)**, *Out* is *Good*, *X* is *Attack*, and *Y* is *Good*. This removes (*X*, *Out*) and (*Good*, *X*) from the set of dependencies.
- UCI finally reduces the set to be one element (*Good*, *Out*) because it is not affected by the intermediate logic and input patterns.

Therefore, the malicious logic can be removed by directly routing the *Good* signal to the *Out* signal using the UCI solution. Note that although UCI has been demonstrated to be outperforming conventional code coverage analysis, it is not bulletproof since advanced Trojan designs [22] can still evade its detection.

6.2.2.2 Formal Verification

Generally, formal verification is a methodology to mathematically check the behaviors of a system. It has been a prevalent solution for design verification in software, aerospace, and military-industrial systems. To apply the formal verification solution for security inspection on third-party IP cores against hardware Trojan threats, proof-carrying code (PCC) is proposed in [9, 13] as illustrated in Fig. 6.5. IP consumers first create the design specifications for the desirable IP design covering the functionality and performance requests. Additionally, they send the security properties that the IP design must obey to the IP vendors along with the design specification. Next, IP vendors design the HDL implementation by following the specification and also provide the security proof corresponding to the received security properties. IP consumers will then verify the trustworthiness of the IP designs by analyzing the security proof. Proof-carrying code can attempt to detect malicious modifications from inserted hardware Trojans because the security proof might be violated. However, the predefined security properties might not be able to guarantee a Trojan-free IP design because one rouge IP designer can intentionally design and implant the malicious functionality in a way satisfying all known properties and place additional vulnerabilities. Besides, the hardware implementation has to be converted to its Coq format which lacks automation and cannot ensure the equivalence between the design representations.

Rajendran et al. [18, 19] use the bounded model checking (BMC) methods to formally detect the hardware Trojans in the third-party IPs. Particularly, [18]

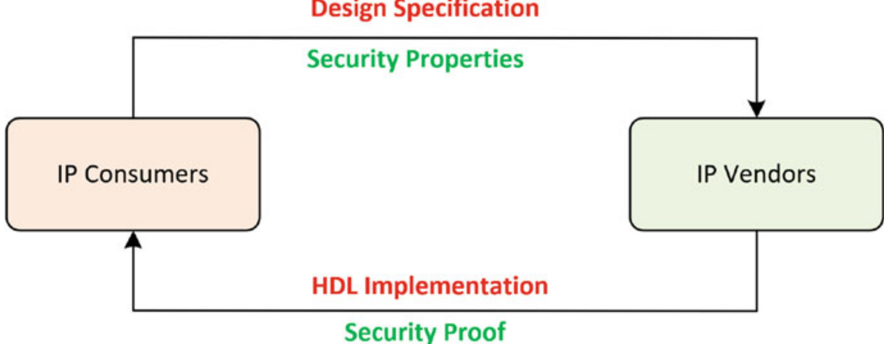


Fig. 6.5 The high-level diagram of proof-carrying solutions for hardware Trojan detection in third-party IPs

targets the Trojan corrupting the security-sensitive assets, while [19] focuses on the information leakage induced by the hardware Trojans. By using the designed security property and state-of-the-art commercial formal verification EDA tools, the proof can be performed automatically to find a *counterexample* if the security property is violated. The Trojan-triggering pattern can be extracted from the counterexample as well. The BMC methods are very effective even when it comes to advanced Trojans such as [32], but scalability is still a great concern since BMC is essentially an NP-hard problem.

6.2.2.3 Structural Analysis

As mentioned, trigger-based hardware Trojans always want to be only activated in rare conditions evading detection to the most. Therefore, the structural analysis aims to locate the suspicious signals in the hardware designs with low transition probability. At RTL, [20] presents a metric named *statement hardness* evaluating how difficult a statement can be executed. The involved signals in the lines of code with high statement hardness are suspicious since they can serve as a good place for placing Trojan logic. Similarly, *hard-to-detect* areas can be identified from the gate-level netlist [21], i.e., the nets with low transition probability and low controllability using the conventional fault models like stuck-at or bridging faults. Note that the identified suspicious signals are not necessarily Trojan logic, calling for manual post-processing and analysis for further confirmation.

6.2.2.4 Logic Testing

Logic testing aims to activate the Trojan inside the pre-silicon implementation using specific patterns in the simulator and detect discrepancies in the outputs from the golden responses. The main challenges come from the fact that the adversaries would intentionally design the trigger condition of Trojans to be very rare to avoid being detected with normal testing methodology. Besides, the large implementations typically contain millions of gates and thus present difficulty in detecting Trojans effectively.

6.2.2.5 Functional Analysis

Instead of using specific patterns to activate Trojans in logic testing, functional analysis chooses *random* patterns to simulate the IPs and identifies the suspicious regions with Trojan features. Waksman et al. [29] presents the functional analysis for nearly unused circuit identification (FANCI) labeling the nets with weak input-to-output dependency as suspicious candidates of hardware Trojans. Since the Trojans can be triggered in rare conditions, the Trojan logic, therefore, remains mostly

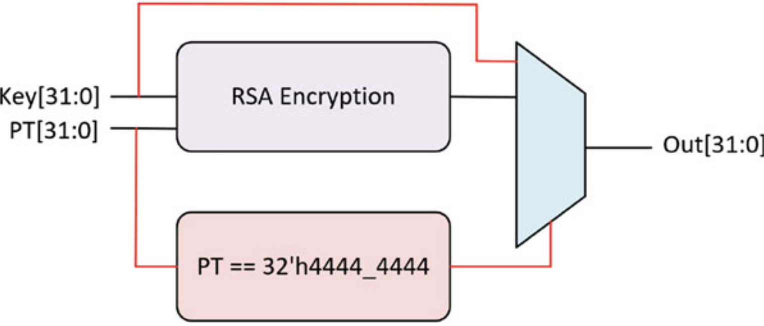


Fig. 6.6 RSA T100 Trojan from TrustHub [27]

unused or *dormant* during normal operations. By applying random patterns, FANCI can calculate the number of transitions for each signal and produce a list of Trojan candidates if the transitions of a function are lower than a predefined threshold. For example, in Fig. 6.6, the active Trojan will direct the security asset, i.e., the private RSA key, to the multiplexer output only when the plaintext is **32'h44444444**. Therefore, the probability of the Trojan trigger signal transitioning from 0 to 1 is as low as 2^{-32} which can be lower than the predefined threshold and categorized as a suspicious net. The major limitation of functional analysis like FANCI is the large number of false positives, making accurate hardware Trojan detection a difficult objective. Also, functional analysis cannot deal with the always-on-type hardware Trojans because the rare condition triggering assumption does not hold.

6.2.3 Post-silicon Hardware Trojan Detection

After the chip tape is out, there are solutions for hardware Trojan detection at post-silicon phases.

6.2.4 Destructive Method

Destructive procedures use hardware reverse-engineering techniques to depackage the target ICs and recover the physical layout according to images retrieved using advanced equipment like scanning electron microscope (SEM). An example solution **TrojanScanner** [28] is illustrated in Fig. 6.7 where the target device samples need to be decapsulated with acid to expose the die at first. Next, SEM is used to scan the entire die to capture feature details for comparison and checking with the golden chip layout. The SEM images are then pre-processed to be 2D shape descriptors where gates can be recognized using K-means and multi-class support

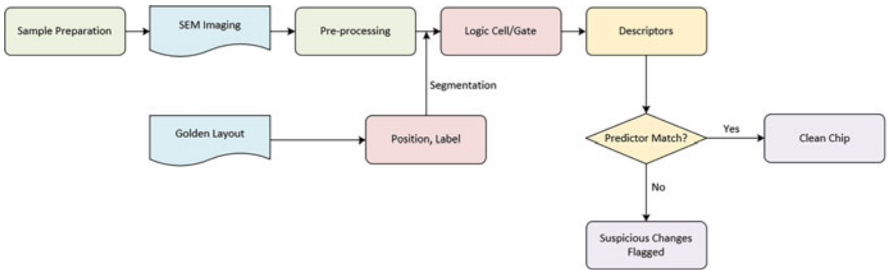


Fig. 6.7 Destructive hardware Trojan detection: Trojan scanner [28]

vector machine (SVM) algorithms. Finally, the information identified from the SEM images will be contrasted against the chip layout to detect any potential malicious modifications of hardware Trojans automatically. The reverse-engineering-based hardware Trojan detection can have a very high assurance and accuracy at the cost of expensive equipment and destructive sample preparation.

6.2.5 Nondestructive Method

As for nondestructive detection solutions, functional tests are very similar to logic testing as discussed in Sect. 6.2. The main difference is functional tests rely on the tester equipment and are limited by the number and capability of probes, while logic testing uses software simulators.

Side-channel-based hardware Trojan detection utilizes the unintentional physical emissions of the running devices including timing delay [12], power consumption [1], and electromagnetic (EM) radiations [23]. The well-known side-channel attacks [34] leverage such observable properties to successfully deduce the underlying private assets like secret keys, demonstrating the usefulness of the physical channels in revealing the design behaviors. Therefore, side channels can be utilized by trusted entities for security or integrity verification of target devices as well. Although the particular physical properties are different per method, the assumption is shared that Trojan-free golden chips are available as the reference. For example, [1] first collects the power profiles of a batch of ICs from all samples as the signatures and then uses destructive methods to confirm that no Trojans are inside these ICs such that the profiled ICs serve as the Trojan-free circuits. The power profiles can thus become the template reflecting the benign behaviors of the original chip designs. The rest of the chips can be tested against the template power fingerprint in a prompt manner using statistical techniques for fast Trojan detection.

To detect hardware Trojan's circuitry during run-time, lightweight security monitors can be placed on-chip. As depicted in Fig. 6.8, [31] presents a network of ring oscillator (RO) instances that are uniformly distributed in the design under monitoring where hardware Trojans might have been implanted. The foundation

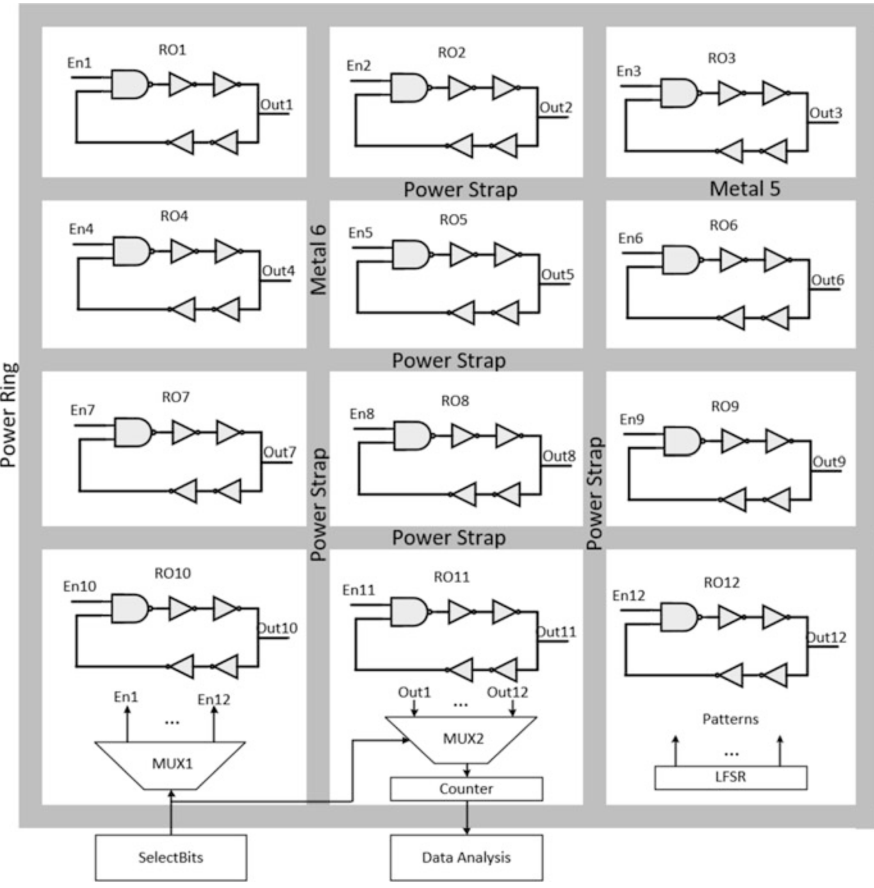


Fig. 6.8 Ring oscillator (RO) network detecting hardware Trojans at post-silicon [31]

behind this technique is that the activated hardware Trojans would cause additional voltage drop on the VDD rail and ground bounce on the VSS rail to be sensed by the RO monitor. In essence, RO is a combinational loop consisting of odd number of inverters; the oscillation frequency is determined by the stage of inverters and the delay of each inverter which is relevant to the voltage supply. Since the effects of Trojan and process variations on the power supply noises are localized, a single RO may not have enough sensitivity, e.g., it is hard for one RO in a corner of the device to capture the Trojan switching activities from another corner. Therefore, as can be seen in Fig. 6.8, there are an array of 12 five-stage (1 NAND + 5 NOT gates) RO instances placed in each power grid surrounded by the power straps. In the functional mode, all ROs are disabled and do not have any impact on the power supply noises. When the authentication (hardware Trojan detection) mode is active, the linear feedback shift register (LFSR) will generate a set of input patterns to

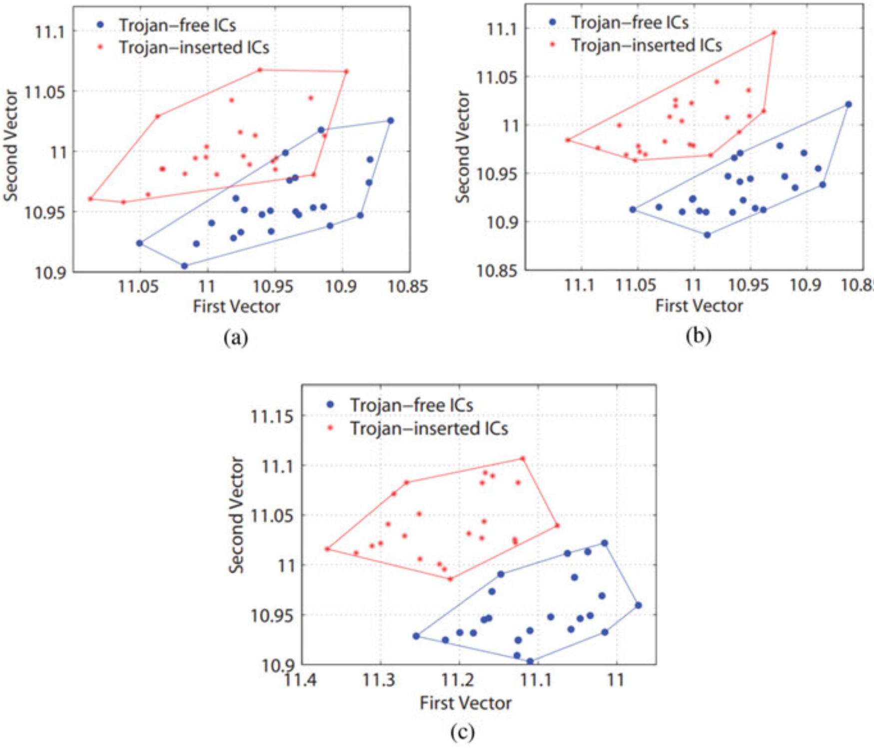


Fig. 6.9 Trojan detection results using the RO power signatures [31]. (**a**) Trojan-7 (**b**) Trojan-8 (**c**) Trojan-9

trigger at least a part of Trojan gates inside the design where the additional power consumption could change the voltage supply variations in the power grids. The multiplexers **MUX1** and **MUX2** are used to select a specific RO and accumulate its oscillation with the counter for frequency calculation. By inspecting all 12 power regions (RO instances), a set of power signatures can be generated for the target implementation. To analyze the obtained RO signatures, a statistical technique named *advanced outlier analysis* is presented to identify the power droop anomalies induced by Trojans from the noises of process variations. The experimental results on detecting three different Trojan variants **Trojan-7**, **Trojan-8**, and **Trojan-9** from 24 Trojan-free FPGAs and 24 Trojan-infected FPGAs are illustrated in Fig. 6.9 where one can observe the **blue** and **red** regions, respectively, indicating the signatures of Trojan-free and Trojan-infected designs can be clearly separated, i.e., most of Trojan variants can be detected using the proposed solution.

6.3 Hardware Trojan Detection Experiment

6.3.1 Experimental Setup

The target Trojan-infected implementation in the hardware Trojan detection experiment is an AES-128 cryptographic engine at RTL (see Fig. 6.10), which is compromised that whenever the predefined plaintext pattern (**128'h0011223344556677889 9AABBCCDDEEFF**) occurs, the output of the implementation will be the sensitive AES key instead of the ciphertext, resulting in undesirable information leakage[8, 16]. The design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch6_hw_trojan_detect/. In order to detect the Trojan logic, a formal verification technique is utilized to perform proof on the security property of strict confidentiality of the AES key. In contrast to dynamic verification solutions like simulation, formal verification refers to a collection of static analysis techniques transforming the hardware implementation into a mathematical representation. As such, a relatively high coverage could be achieved without exercising the target implementation with numerous test vectors because formal verification might not need to evaluate every possible state of the circuitry. Figure 6.11 illustrates the applications of hardware behavior formal verification. We highlight the four applications out of them, i.e., equivalence checking, model checking, SoC-level formal verification, and security path verification as follows [7]:

- **Equivalence checking.** The equivalence checking aims to prove whether two representations of a design are functionally equivalent or not, which can be useful to guarantee the design functionality is not altered after synthesis optimization or functional engineering change order (ECO) in the physical layout phase. Synopsys formality [24] is a widely used EDA tool for equivalence checking where the reference design, revised design, and library are specified at first, then

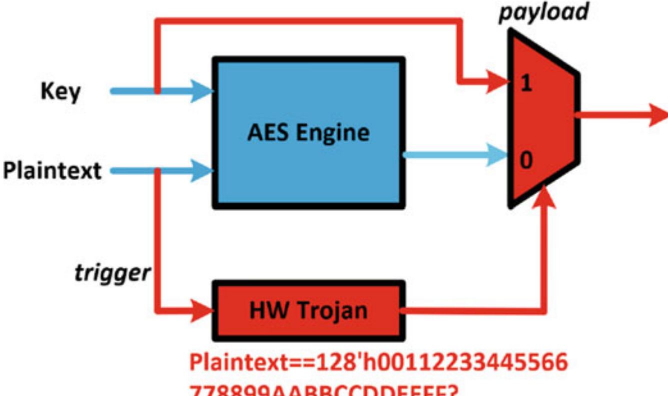


Fig. 6.10 The experimental Trojan-infected AES implementation

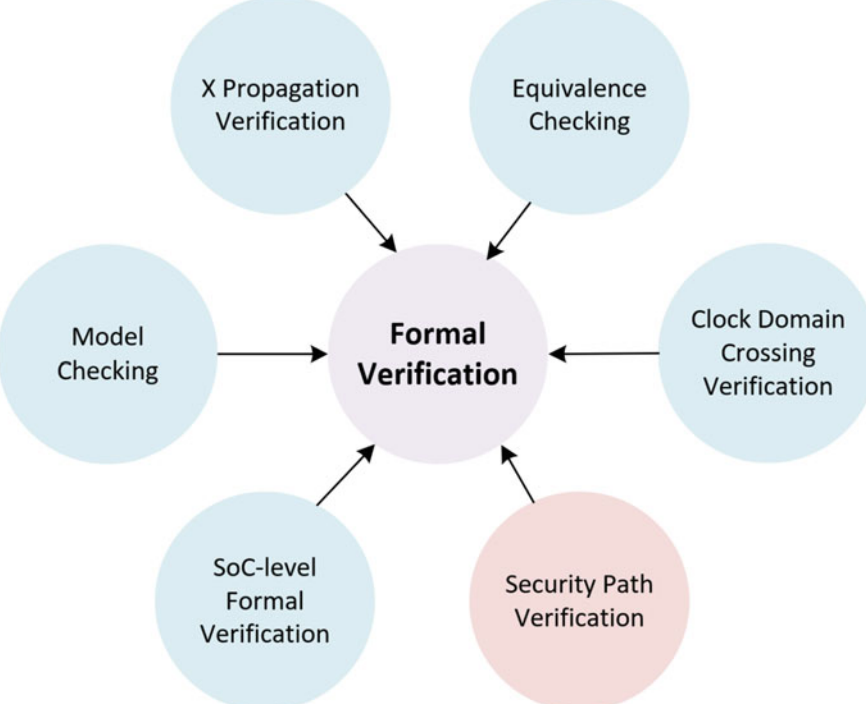


Fig. 6.11 Formal verification techniques

key points of the design are mapped and compared, and finally the mapped points are examined to see if they are equivalent or not.

- **Model checking.** Model checking is proposed to check the correctness of a particular specification. Instead of enumerating all reachable states, the binary decision diagram (BDD) manipulates the Boolean function directly for boosting the capacity handling of the realistic hardware systems. However, BDD-based model checking suffers from the state space explosion problem of running out of memory quickly. In order to address the issues, bounded model checking (BMC) is presented to search for a counterexample in the executions bounded by the specified integer k such that the BMC problem can be reduced to a satisfiability one to be solved by the SAT solver.
- **SoC-level formal verification.** Limited by the scalability issues, it is hard to apply formal verification techniques on the entire SoC design of high complexity. A feasible direction is verifying critical coverage points using formal verification and focusing on the power of dynamic simulation vectors on the uncovered parts for time-saving.
- **Security path verification.** By using path sensitization technology, security path verification techniques can detect the potential vulnerabilities in propagating data

from/to a defined secure region in the hardware design such that any violations of confidentiality and integrity of sensitive data can be examined.

The particular formal verification environment in this experiment is Cadence JasperGold [11] which is an industrial formal verification platform supporting a wide spectrum of applications including formal property verification (FPV), security path verification (SPV), and X-propagation verification. In this experiment, the SPV application is a good fit for Trojan detection since it can exhaustively prove the secure data (AES key here) cannot be read or overwritten illegally. Otherwise, counterexamples will be reported and analyzed to showcase why and how the specified security property is violated. The Trojan-triggering condition can be extracted from the counterexample if the proof fails. Note that a *white box* scenario is assumed here to provide readers a more intuitive understanding and hands-on experience on Trojan detection, which is different from the *black box* case when it comes to third-party IPs.

6.3.2 Experimental Steps

The experimental steps on detecting the Trojans inserted in the AES engine as illustrated in Fig. 6.10 using Cadence JasperGold SPV application are articulated as follows:

1. Access to the MEST ECE server where commercial Cadence JasperGold licenses are installed, using the provided credentials through VNC viewer terminals.
2. Start JasperGold by executing the script: *source start_jg.tcl*. The script essentially invokes the Cadence JasperGold SPV application.
3. Jaspergold SPV application graphic user interface (GUI) and execute the script *HW_Trojan_Detection.tcl* for automatic Trojan detection. The Jaspergold SPV GUI is depicted in Fig. 6.12.
4. The *HW_Trojan_Detection.tcl* is the core in this experiment. It first loads the Trojan-infected AES RTL implementations in Cadence JasperGold as illustrated in Fig. 6.13 where the first four files from *aes_128.v* to *table.v* belong to the original Trojan-free AES implementation and the last two items *Trojan_Trigger.v* and *TSC.v* correspond to the inserted hardware Trojan at RTL. Next, the script sets up the design by specifying the top module, clock port, and reset condition.
5. The security property is written as an assertion in the *HW_Trojan_Detection.tcl* as **check_spv -create -from key -to out -to_precond {key == out}**. This assertion regulates the data taints cannot propagate from the AES *key* input to its *out* port. Note that the *-to_precond* constraint means the *out* value should not equal to *key* either to avoid information leakage. If the proof procedure (**check_spv -prove -all**) on this assertion fails, a counterexample will be reported to inform when and how the *key* can reach the *out* port.

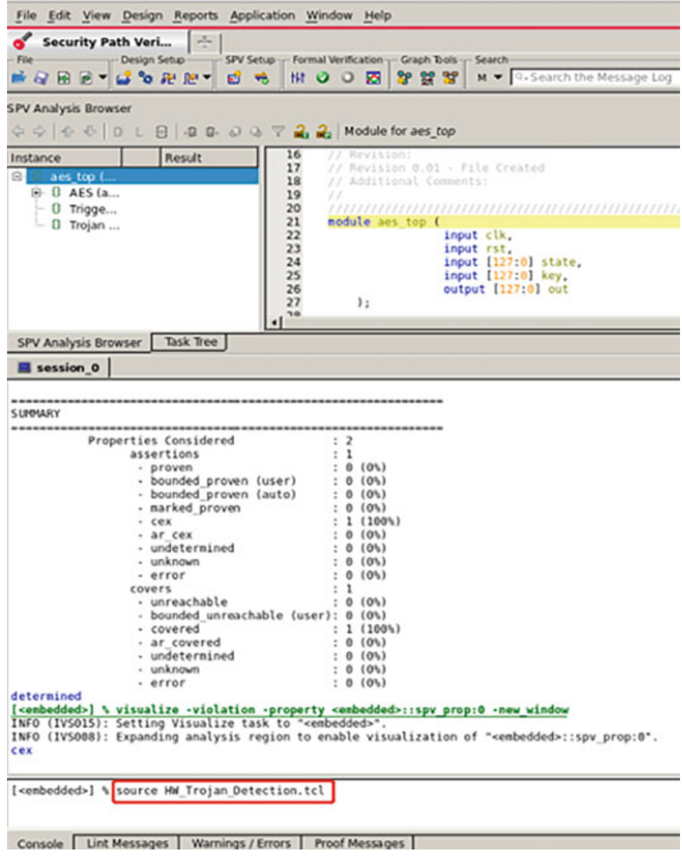


Fig. 6.12 Jaspergold SPV GUI

```
set src_path "./src"

analyze -verilog $src_path/aes_128.v;
analyze -verilog $src_path/AES_top.v ;
analyze -verilog $src_path/round.v ;
analyze -verilog $src_path/table.v ;
analyze -verilog $src_path/Trojan_Trigger.v ;
analyze -verilog $src_path/TSC.v ;

elaborate -top {aes_top};
clock clk;
reset rst;

check_spv -create -from key -to out -to_precond {key == out}
check_spv -prove -all
```

Loading Design

Set up Design

Security Property

Fig. 6.13 Screenshot of the script *HW_Trojan_Detection.tcl*

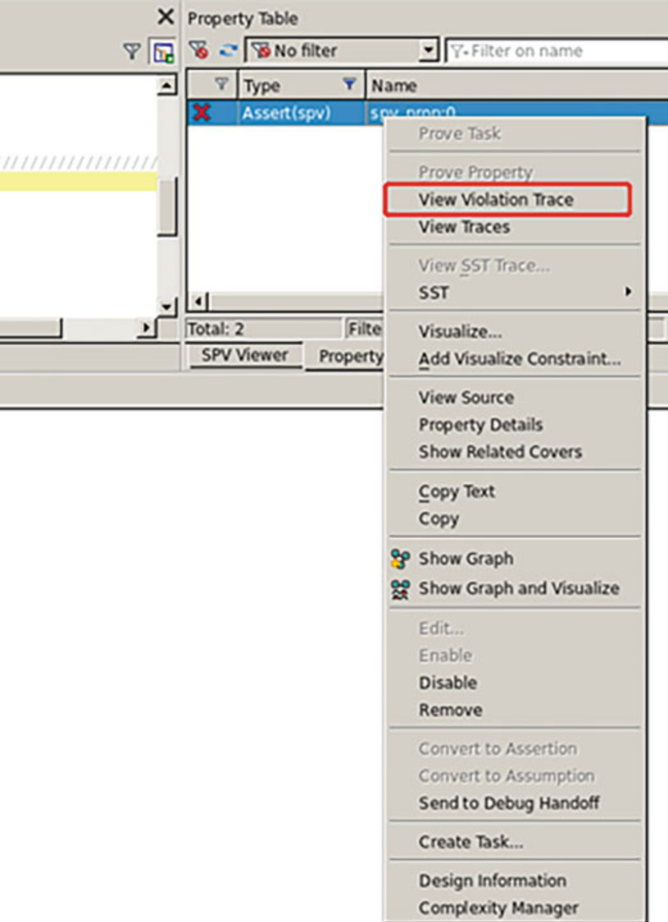


Fig. 6.14 Visualize the counterexample

6. By running the *HW_Trojan_Detection.tcl* script, one can see the proof fails. The counterexample is informative for diagnosis purposes which can be visualized by following the instructions in Fig. 6.14.
7. By visualizing the counterexample, one can easily identify that the data taints on the *key* input can reach the *out* port (visually, both signals are labeled to be red in Fig. 6.15). Meanwhile, the values on *key* and *out* are the same, i.e., **128'h7fffffffffffffffffffffffffffffff**.
8. As the security property only concerns about the *key* and *out* signals, the plaintext input of AES, i.e., *state*, is not displayed in the counterexample window by default. It can be added for visualization by following the instructions in Fig. 6.16.

Fig. 6.15 Add signal in the counterexample window

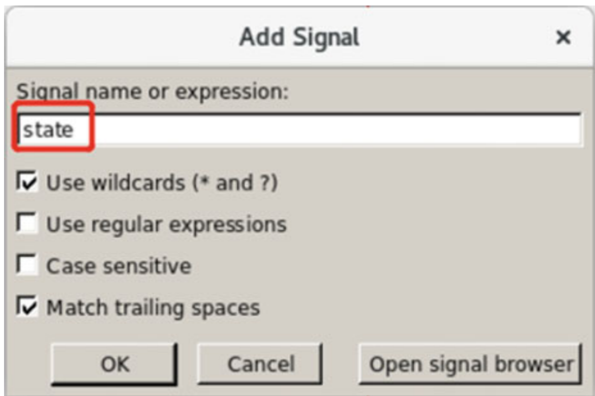


Fig. 6.16 Add the state signal

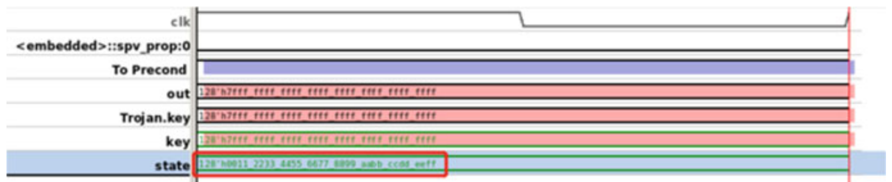


Fig. 6.17 Identified Trojan trigger pattern

9. As presented in Fig. 6.17, when Trojan is activated, the plaintext *state* signal should be assigned with the predefined pattern **128'h00112233445566778899 AABBCCDDEEFF**. In other words, Jaspergold Cadence SPV application can report the Trojan-triggering sequence by performing formal security verification without any prior knowledge.

6.4 Conclusion

Hardware Trojans have emerged as serious security concerns for today's semiconductor devices. How to effectively and efficiently detect hardware Trojans from the hardware designs present formidable challenges to researchers in industry and academia. In this chapter, the state-of-the-art hardware Trojan detection solutions at both pre-silicon and post-silicon stages including code coverage analysis, formal verification, and side-channel-based detection are covered. Moreover, in order to provide hands-on experience on hardware Trojan detection, an experiment is presented to showcase how to utilize the commercial EDA formal verification tool, i.e., Cadence JasperGold security path verification application, to effectively identify the Trojan-triggering sequence of an infected AES implementation at RTL.

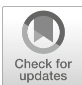
References

1. Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P., Sunar, B.: Trojan detection using IC fingerprinting. In: 2007 IEEE Symposium on Security and Privacy (SP'07), pp. 296–310. IEEE (2007)
2. Ahmed, Q.A., Wiersema, T., Platzner, M.: Malicious routing: circumventing bitstream-level verification for FPGAs. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1490–1495. IEEE (2021)
3. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms. Preprint arXiv:2204.07909 (2022)
4. Anandakumar, N.N., Rahman, M.S., Rahman, M.M.M., Kibria, R., Das, U., Farahmandi, F., Rahman, F., Tehranipoor, M.M.: Rethinking watermark: providing proof of IP ownership in modern SoCs. In: Cryptology ePrint Archive (2022)
5. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Los Altos (2018)
6. Chakraborty, R.S., Saha, I., Palchaudhuri, A., Naik, G.K.: Hardware Trojan insertion by direct modification of FPGA configuration bitstream. IEEE Des. Test **30**(2), 45–54 (2013)
7. Formal Verification. https://www.techdesignforums.com/practice/guides/formal-verification-guide/. Accessed 5 October 2022
8. Giri, N., Anandakumar, N.N.: Design and analysis of hardware Trojan threats in reconfigurable hardware. In: 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), pp. 1–5. IEEE (2020)
9. Guo, X., Dutta, R.G., Jin, Y., Farahmandi, F., Mishra, P.: Pre-silicon security verification and validation: a formal perspective. In: Proceedings of the 52nd Annual Design Automation Conference, pp. 1–6 (2015)
10. Hicks, M., Finnicum, M., King, S.T., Martin, M.M.K., Smith, J.M.: Overcoming an untrusted computing base: detecting and removing malicious hardware automatically. In: 2010 IEEE Symposium on Security and Privacy, pp. 159–172. IEEE (2010)
11. Jaspergold Platform. https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.htm. Accessed 5 October 2022
12. Jin, Y., Makris, Y.: Hardware Trojan detection using path delay fingerprint. In: 2008 IEEE International Workshop on Hardware-Oriented Security and Trust, pp. 51–57. IEEE (2008)

13. Jin, Y., Yang, B., Makris, Y.: Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 99–106. IEEE (2013)
14. Kelly, S., Zhang, X., Tehranipoor, M., Ferraiuolo, A.: Detecting hardware Trojans using on-chip sensors in an ASIC design. J. Electron. Test. **31**(1), 11–26 (2015)
15. Li, M., Davoodi, A., Tehranipoor, M.: A sensor-assisted self-authentication framework for hardware Trojan detection. In: 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1331–1336. IEEE (2012)
16. Manivannan, S., Anandakumar, N.N., Devi, M.N.: Key retrieval from AES architecture through hardware Trojan horse. In: International Symposium on Security in Computing and Communication, pp. 483–494 (2018)
17. Rahman, M.T., Rahman, M.S., Wang, H., Tajik, S., Khalil, W., Farahmandi, F., Forte, D., Asadizanjani, N., Tehranipoor, M.: Defense-in-depth: a recipe for logic locking to prevail. Integration **72**, 39–57 (2020)
18. Rajendran, J., Vedula, V., Karri, R.: Detecting malicious modifications of data in third-party intellectual property cores. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2015)
19. Rajendran, J., Dhandayuthapany, A.M., Vedula, V., Karri, R.: Formal security verification of third party intellectual property cores for information leakage. In: 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), pp. 547–552. IEEE (2016)
20. Salmani, H., Tehranipoor, M.: Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level. In: 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 190–195. IEEE (2013)
21. Salmani, H., Tehranipoor, M., Karri, R.: On design vulnerability analysis and trust benchmarks development. In: 2013 IEEE 31st International Conference on Computer Design (ICCD), pp. 471–474. IEEE (2013)
22. Sturton, C., Hicks, M., Wagner, D., King, S.T.: Defeating UCI: building stealthy and malicious hardware. In: 2011 IEEE Symposium on Security and Privacy, pp. 64–77. IEEE (2011)
23. Sun, S., Zhang, H., Cui, X., Dong, L., Fang, X.: Electromagnetic side-channel hardware Trojan detection based on transfer learning. IEEE Trans. Circuits Syst. Express Briefs **69**(3), 1742–1746 (2021)
24. Synopsys Formality. https://www.synopsys.com/glossary/what-is-equivalence-checking.html. Accessed 5 October 2022
25. Tehranipoor, M., Koushanfar, F.: A survey of hardware Trojan taxonomy and detection. IEEE Des. Test Comput. **27**(1), 10–25 (2010)
26. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer Science & Business Media, Berlin (2011)
27. TrustHub Benchmark. https:https://trust-hub.org. Accessed 5 October 2022
28. Vashistha, N., Lu, H., Shi, Q., Rahman, M.T., Shen, H., Woodard, D.L., Asadizanjani, N., Tehranipoor, M.. Trojan scanner: detecting hardware Trojans with rapid SEM imaging combined with image processing and machine learning. In: ISTFA 2018: Proceedings from the 44th International Symposium for Testing and Failure Analysis, pp. 256. ASM International (2018)
29. Waksman, A., Suozzo, M., Sethumadhavan, S.: FANCI: identification of stealthy malicious logic using Boolean functional analysis. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 697–708 (2013)
30. Xiao, K., Forte, D., Jin, Y., Karri, R., Bhunia, S., Tehranipoor, M.: Hardware Trojans: lessons learned after one decade of research. ACM Trans. Des. Autom. Electron. Syst. **22**(1), 1–23 (2016)
31. Zhang, X., Tehranipoor, M.: RON: An on-chip ring oscillator network for hardware Trojan detection. In: 2011 Design, Automation & Test in Europe, pp. 1–6. IEEE (2011)

32. Zhang, J., Yuan, F., Xu, Q.: Detrust: defeating hardware trust verification with stealthy implicitly-triggered hardware Trojans. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 153–166 (2014)
33. Zhang, T., Wang, J., Guo, S., Chen, Z.: A comprehensive FPGA reverse engineering tool-chain: from bitstream to RTL code. IEEE Access **7**, 38379–38389 (2019)
34. Zhang, T., Park, J., Tehranipoor, M., Farahmandi, F.: PSC-TG: RTL power side-channel leakage assessment with test pattern generation. In: 2021 58th ACM/IEEE Design Automation Conference (DAC), pp. 709–714. IEEE (2021)

Chapter 7
Security Verification



7.1 Introduction

Pre-silicon verification [17] involves the use of functional properties to formally evaluate the expected behavior of a design and its specification. The verification efforts of design houses can be reduced to a great extent by defining functional properties and utilizing them in formal verification tools. In this way, formal verification tools will be able to prove or disprove whether the design behavior matches the specification. Apart from formal verification, functional properties can also be used in the case of simulation-based verification to identify design bugs [1]. Moreover, some of these properties can be utilized in post-silicon validation as well, by synthesizing and placing them in real design to monitor specific events and raise exceptions when needed [11]. At the pre-silicon design stages of the SoC life cycle, the functional properties are usually insufficient for security validation to prove the design's trustworthiness. A whole supplementary set of properties aimed at ensuring the secure operation of a design needs to be developed since an SoC may encounter different security vulnerabilities throughout its life cycle [13].

Since vulnerabilities can be introduced at different stages of the SoC life cycle, security properties should be checked from the very beginning of the SoC design life cycle [26] to reduce cost and verification time. Security properties can also be checked in a variety of ways, just like functional properties. They can be verified formally/dynamically using model checking/simulation tools. Such verification can enable the designer to uncover and fix security bugs at an early stage. Moreover, these properties can be mapped to reconfigurable fabrics and enforced as security policies/rules during run-time to protect the SoC from zero-day attacks.

We have organized this chapter as follows: Section 7.2 explains how properties are developed and formally represented in general. Sections 7.3.1 and 7.3.2 provide the concept of security assets and identify how different threat models make security assets vulnerable. Section 7.3.3 describes how security properties are generated to protect against security vulnerabilities. Section 7.4 shows some examples of

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_7

security properties for a design example and the experimental setup for checking the properties. Finally, Sect. 7.5 concludes the chapter.

7.2 Background: Writing Properties

The concept of a property in verification refers to a statement that can verify assumptions, conditions, and expected behaviors in a design. A property can be represented in the form of an *assertion* or *cover* statement for formal verification. An *assertion* can check if everything is working correctly in the design and notify if an illegal event has happened based on the specification [14]. A single bit associated with the *assertion* indicates the pass or fail status. If the *assertion* is violated, it provides a counterexample that can be helpful for finding design bugs. Likewise, a *cover* statement can check if a scenario under consideration can ever happen in the design. Hence, the *cover* statement provides a specific scenario if it can possibly occur during simulation if it passes. If the *cover* statement is undetermined/unreachable, it means the scenario can never occur. *Assertions* are mainly written in two ways: (i) *immediate assertions* and (ii) *concurrent assertions*. *Immediate assertion* checks if an event can happen at any time if a condition is passed. On the other hand, *concurrent assertion* checks for expected behavior over a period of time. The difference between *immediate* and *concurrent assertion* is that an *immediate assertion* occurs within a procedural block (an initial or always block), whereas a *concurrent assertion* can occur within a procedural block or within a module (e.g., assign, always, or initial block). Moreover, *assume* statement is used very often to create the verification environment and reduce the search space for the verification tool. The formal verification process is also referred to as assertion-based security verification (ABV) because of the widespread use of assertions for verification.

SystemVerilog Assertions (SVA) [28] and Property Specification Language (PSL) [16] are two popular assertion languages used for describing interesting features of a design. *Assertions* use temporal logic representations such as linear temporal logic (LTL) [25] and computational tree logic (CTL) [8]. Languages based on LTL and CTL are often described using Boolean expressions which are the most common way of describing properties. Logical operators, e.g., "AND" and "OR" can be used to evaluate Boolean expressions. Temporal sequences can be written using ## operators and the number of specific clock cycles needed for an event to happen. For example, *a ##5 b* assertion indicates that *b* will be valid after five clock cycles if *a* is valid. In addition, different operators such as [low : high], [*], [=] or [$\rightarrow$] represent bounded or unbounded repetition, the repetition of zero or more consecutive instances; however, [=] and [$\rightarrow$] denote one or more nonconsecutive repetitions. The sequences of different events can be combined using several operators, e.g., "AND," "INTERSECT," "OR," "UNTIL," "THROUGHOUT," "WITHIN," "EVENTUALLY," etc.

7.3 SoC Security Verification Using Property Checking

A system on chip (SoC) may encounter different security threats in its life cycle, whereas an attacker's motive is to extract/control its valuable resources that are worth protecting from adversaries, known as "security assets" [13]. Property-driven security verification, which formally describes the expected behaviors of the design in the context of a security threat model and associated assets, can protect well-defined security assets that reside inside the SoC. The developed security properties are then fed to formal verification tools for checking (as shown in Fig. 7.1). If all the properties pass, it provides a designer with good confidence in having a secure design. Otherwise, a designer can go back and fix his design to make it more secure.

Building on the pre-silicon framework as described in Fig. 7.1, we have developed the workflow as shown in Fig. 7.2 for security validation. In our proposed framework, we introduce two steps before property generation. These two steps, "security asset identification" and "threat model identification," help designers to generate security properties which can be checked by formal tools.

7.3.1 *Security Asset Identification*

SoCs form the core of personal computing devices such as mobile phones, laptops, etc., which handle users' data, such as bank details, medical history, passwords, etc., daily. SoC designs have become more complex to manage the various tasks required

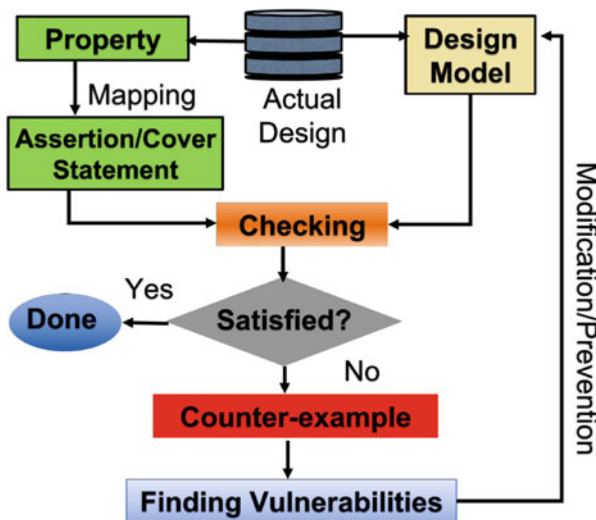


Fig. 7.1 Workflow of pre-silicon verification using property-driven formal analysis

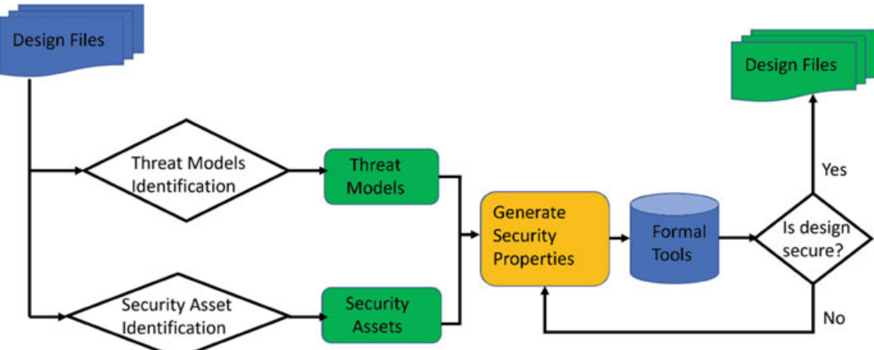


Fig. 7.2 Property-driven verification framework for security validation

of the modern user. Current SoCs incorporate and integrate hundreds of different hardware blocks, each carrying out a specific function, and are named "Hardware Intellectual Property" (Hardware IP). Present-day SoCs are an amalgamation of these hardware IP blocks, capable of performing various tasks at high speeds. As the SoCs become bigger, incorporating a higher number of hardware IP blocks and the detailed protocols designed to manage communication between them, securing the whole SoC from adversaries has become a mammoth task.

Securing the whole SoC against all possible threats is nearly impossible in this competitive market, where the time to market for SoC production shrinks with technological advances. Thus, a secure SoC can be defined as "One incorporating measures to deny an adversary any power to modify, extract or deny access to any of the users' critical data or an SoC resource that protects the users' data." The user data or any SoC resource to be protected is called a "security asset" [13]. Design houses can ensure security by incorporating measures to shield the defined security assets for the SoC under design. But how do we identify security assets for an SoC?

Security asset identification requires the designer to understand the environment in which the SoC will be deployed, the various functionalities of the SoC, and the various threats it may face during its lifetime. Design houses develop SoC with different specifications depending on the environment in which they will be deployed. Data handled by an SoC varies on its specification. For example, an SoC integrated into a personal computing device is designed to address critical data such as users' financial and medical data, whereas SoCs developed to be incorporated into space or defense missions are designed to handle more complex data such as telemetry, etc. Hence, defining the environment in which the SoC will function can give the designer the type of data being addressed and that needs to be protected. Once the environment for the SoC has been defined, the designer needs to know the specific regions of the SoC that are responsible for handling this data. Various SoC components such as memory units (RAM and ROM), encryption units (AES, RSA), and system bus store, utilize, and handle various user data. Depending on the SoC environment and data being handled, SoC integrate various such components.

Identifying these SoC components and the conditions under which they handle users' data and ensuring they are not accessible by an adversary ensure the security of the SoC. These SoC components are denoted as "security assets." The security assets identified can be further categorized into primary and secondary assets as defined in [13]. Security assets can be identified through manual efforts, such as inspecting the RTL code & functional specification, or automated using techniques such as those described in [12].

7.3.2 Threat Model Identification

Identifying security assets is the primary step toward property-driven verification of SoC security validation. The other vital component is the identification of the threat model. As described in the above section, different SoCs are employed in various diverse environments depending on their specification. This results not only in a myriad of set security assets but also in the threats that they face. Property-driven verification requires the designer to understand the various threat models that apply to an SoC under consideration. A threat model is defined as the "operating conditions under which an adversary can gain access to an SoC component and extract, modify, or deny access to the component or the data held by it." Acknowledging the various threat models encountered by an SoC during its life cycle, a designer can then introduce security measures into the SoC design to protect its security assets.

Similar to security assets, the threat models faced by an SoC can be defined by the operating environment in which they will be operational. Threat models to an SoC are broadly divided as follows:

Confidentiality [4] Confidentiality refers to the unauthorized access of users' critical data by an adversary. The confidentiality threat model dictates that under no condition should an adversary or outside party be capable of extracting secrets stored in the SoC.

Let us consider the sample SoC block shown in Fig. 7.3. As described in Sect. 7.3.1, the AES and RSA key is security assets that must be protected. These keys can be hard-coded into the encryption IP, generated using security primitives such as TRNGs [3], or stored in secure memory regions. During operation, the key value is propagated along the system bus to the encryption cores. The adversary can create an environment in which he can access this information directly by observing output ports or probing the chip [5, 29] or injecting faults [10] is a confidentiality threat model. Some examples of the confidentiality threat model have been utilized in attacks such as Spectre [20], Meltdown [21], etc., where the attacker takes advantage of speculative execution. Then using cache timing attacks, extracts secret information.

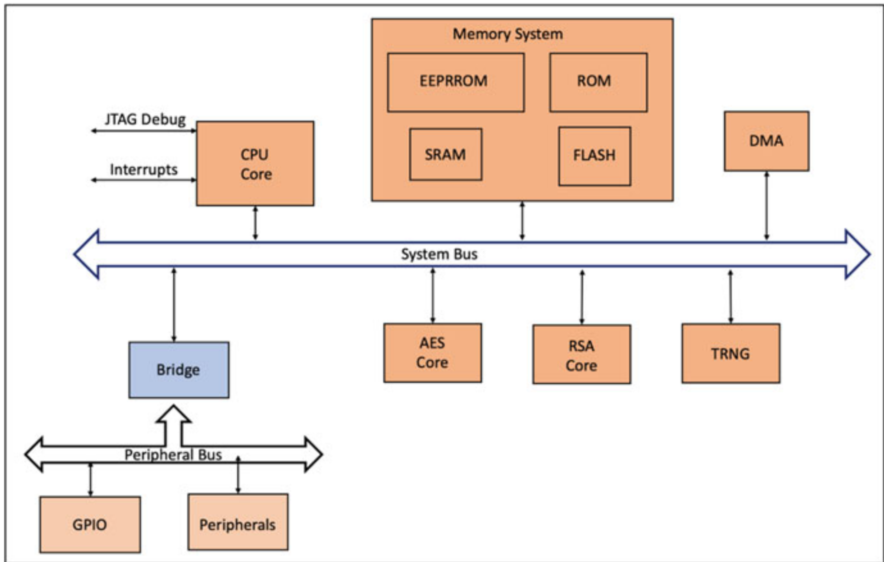


Fig. 7.3 Sample system-on-chip block consisting of a CPU core; memory system; GPIO; peripherals; DMA controller; encryption IPs AES, RSA, and TRNG; and a system and peripheral bus for communication

Integrity [4] Integrity refers to an adversary's unauthorized modification of secure data. The integrity threat model dictates that under no condition should an adversary or outside party be capable of modifying data stored in the SoC.

Referring back to the sample SoC in Fig. 7.3, user personal data, such as password credentials, are stored encrypted in the memory. An adversary can create conditions such as inducing laser faults, inducing voltage [15] and clock [24] glitches, etc., to modify the data itself or the control sequence of the SoC, resulting in modifying data. Such a threat environment is considered an integrity threat model. In attacks such as Rowhammer [19], the adversary creates an integrity threat model that results in SoC data modification.

Threat model identification for an SoC is a critical task requiring a complete understanding of its operational environment and security assets and the available attack methods used by an adversary. Considering an SoC to be incorporated into a mobile computing device, the adversary can gain physical access to the SoC through reverse engineering and hence can utilize both physical attack methods such as reverse engineering, fault injection, probing, and remote attacks. Therefore, the design house has to incorporate security measures against both physical and remote attacks. Considering an SoC to be incorporated into a space mission, the adversary cannot have physical access; hence, attacks such as reverse engineering and probing are impossible. For such cases, the design house resource is better served by utilizing them to protect the SoC against remote attacks.

7.3.3 Generating Security Properties

The above two sections discussed the primary steps toward property-driven verification for security validation. This section details how to generate the security assertions required for security validation. Once the security assets and the threat models for the SoC have been identified, the designer then needs to define the secure behavior for the SoC. Equipped with the security assets of the design, the designer needs to identify the SoC functionalities in which the security asset is involved. Consider the sample SoC shown in Fig. 7.3. From Sect. 7.3.1, we have determined that the AES key, which is utilized to encrypt confidential user data, is a security asset that needs to be protected. The key is utilized under two functions of the SoC, encryption/decryption of data and the bus protocol that transports it to the AES core. The designer needs to identify the exact conditions under which these two functionalities take place, i.e., control sequences under which encryption or decryption takes place, bus protocols that propagate the key, etc.

The designer then needs to consider various threat models that can affect the security asset under consideration and the possible attack vectors for that threat model. This equips the designer with the potential attack scenario to extract/modify the security asset. The designer can then formulate the behavior of the security asset that allows for proper functionality but ensures that the security asset isn't affected by the threat model under consideration.

7.4 Experimental Setup

In our hands-on experiment, we utilized the Cadence JasperGold [18] verification tool for the formal analysis of the generated properties. The source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch7_security_verification/. Cadence JasperGold is an electronic design automation (EDA) supplier of high-level formal functional verification software, among other tools such as Synopsys VC [27] and Cyucity Radix-S [9]. It enables exhaustive and complete verification, provides rapid bug detection, and completes end-to-end proofs of expected design behavior. It gives a counterexample showcasing the exact conditions under which the failed property is violated, equipping the designer with the required information to analyze the design and make the required changes. JasperGold comprises multiple applications such as the formal property verification (FPV) [6] used for property checking, security path verification (SPV) [7] utilized for checking the confidentiality and integrity of the information flow within the SoC, and many such other applications [18].

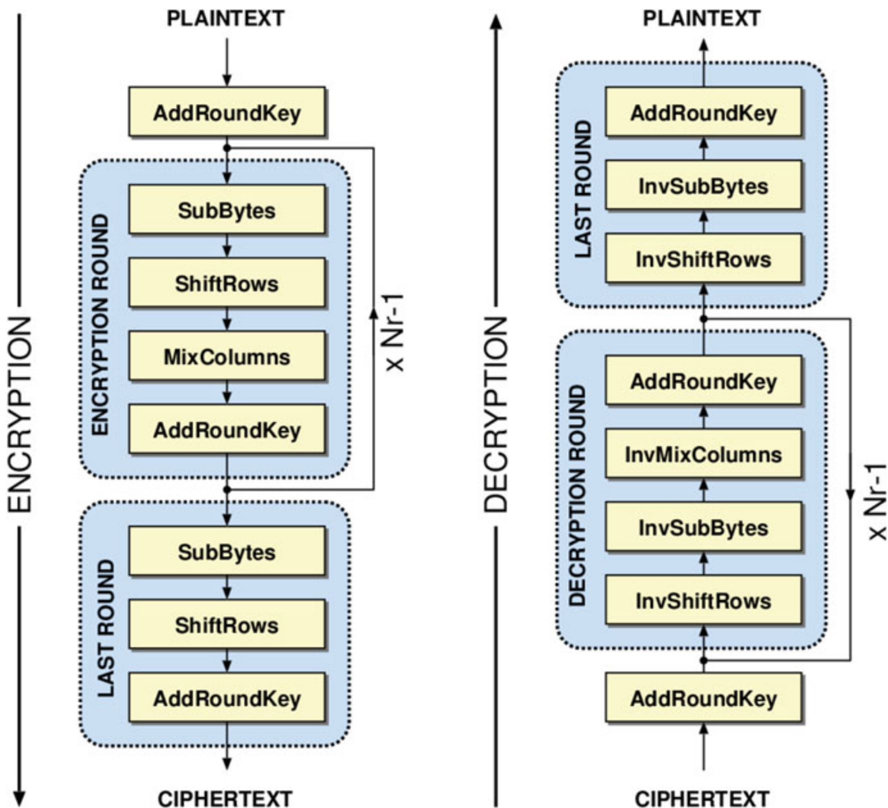


Fig. 7.4 Hardware AES block diagram showing the various suboperations in each round of encryption (left hand side) and decryption (right hand side). All rounds of operation consist of an SubBytes, ShiftRows, MixColumns, and AddRoundKey, except for the last round. (Image credits: https://i.stack.imgur.com/HyRg4.png)

7.4.1 AES Design

For the experiment, an AES implementation is utilized for which properties for security validation are derived and tested. A high-level block diagram of an AES implementation that completes one round of AES per clock cycle is shown in Fig. 7.4.

AES-128 is a 128-bit symmetric key cryptographic module used for the encryption and decryption of data in an SoC. It consists of ten rounds of encryption [2]. The first nine rounds consist of a substitution operation, a nonlinear substitution step of each byte of state data according to a look-up table. This is followed by a ShiftRows operation where a transposition of the bytes occurs cyclically. Following is a mixed column operation that operates on the state columns and is XORed with the round key derived from the input key in the AddRoundKey. The final round is

a substitution operation, followed by a ShiftRow operation and finally XORed with the round key in the AddRoundKey operation.

For this AES-128 design, we have identified the plaintext and key inputs as critical security assets. Both confidentiality and integrity are considered threat models. Any leakage or manipulation of these resources could compromise the whole security of the system, giving an adversary access to essential and sensitive information. We have also identified information leakage and access control as threat models that can affect this design.

7.4.2 Security Property Development for Verification

The designer is now equipped with the security assets (key and plaintext inputs) for the AES design and the corresponding threat models (confidentiality and integrity). For the threat model under consideration, we can define some natural language properties that describe the secure behavior of the security assets.

Let us now consider the confidentiality threat model, where the adversary aims to extract the information regarding the security asset but causes it to leak or flow to an observable point accessible by the adversary. The below properties were developed for the AES design described in Sect. 7.4.1:

Security Assets: Key, plaintext, intermediate results of encryption Threat Model: Confidentiality – leaking sensitive information of an AES design unauthorizedly

Developed Properties

- *Property P1:* Input data of the encryption module should not flow/should not be observable to the output port.
- *Property P2:* Encryption keys/registers that store encryption keys should not flow/should not be observable to the output port.
- *Property 3:* Registers storing plaintext should not be observable from an output signal.
- *Property P4:* Key control registers should not be observable from the output signal.
- *Property P5:* Plaintext control registers should not be observable from the output signal.
- *Property P6:* The encryption key should not be shared with other modules of the design.
- *Property P7:* Intermediate results of encryption should not flow to an output port.

Property P1 describes the need to ensure that no direct path flows from the input key and plaintext ports resulting in a direct leakage of the security asset. Such leakage paths can be introduced unintentionally through designer negligence, intentionally through a malicious implant [22], or through CAD tool optimizations [23].

Properties P2 and *P3* are to check for any information flow path from the register/memory units storing the key, and plaintext data do not leak partial or complete information regarding them to the outside world.

Properties P4 and *P5* check for the possibility of access by an adversary to the control registers that control the key and plaintext data. An adversary can induce faults (laser, voltage, or clock glitching) to override controls to the security assets and then extract them.

Property P6 checks to see if, during the key propagation, any information regarding the key flows to any other design modules (the key should only flow between its memory register and the AES IP). An adversary may tap into another module and extract the key information.

Property P7 analyzes for the scenario where the intermediate results of the encryption leak to the output port. An adversary with access to the intermediate results of an AES encryption reverse engineered the key and plaintext values.

7.4.3 Property-to-Assertion Conversion

The security properties are in natural language, so they need to be converted into assertions/cover statements to make them readable for verification tools. For a specific design example, a user must identify appropriate signals and concatenate them using necessary operators based on the associated security property. Thus, the security property is converted into an assertion/cover statement. Multiple assertion/cover statements may need to be written for a single property. Moreover, the same property can be converted into different assertions for different design implementations and abstraction levels. The conversion of security properties (mentioned earlier) to assertions is shown in Table 7.1 for an encryption module.

7.4.4 Compiling Target Design and Property Verification

This section presents detailed step-by-step instructions on compiling and testing the properties we have defined. The developed properties are input to the JapserGold tool as scripts. The first four assertions are illustrated in Table 7.1 and are checked using the JasperGold security path verification (SPV) application, a part of the complete set of applications in the JasperGold tool suite. The fifth and final assertion is checked using the JasperGold formal property verification application. Both applications can be called using a script, which will be described later in the section.

Table 7.1 Property-to-assertion conversion for a design example

Design name	Vulnerabilities	Associated assets	Threat	Property	Assertion	Tool used	IL-based property coverage
AES encryption module	Observability of key/intermediate results from the output ports	Key/plaintext	Confidentiality violation	Input data should not leak to output port	P1: check_spv -create -from {key text_in} -to {text_out}	Jasper Gold SPV	6/7 =0.85
				Key/plaintext control registers should not be observable from output signal	P4,P5:check_spv -create -from {ld} -to {done}		
				Registers storing key should not be observable from output signal	P2:check_spv -create -from {key} -to {text_out} [P.S. key is a register within the AES module]		
		Plaintext		Registers storing plaintext should not be observable from output signal	P3: check_spv -create -from { text_in} -to {text_out} [P.S. plaintext is a register within the AES module]		
		Intermediate result of encryption		Intermediate result shouldn't flow to output before round count value is 10	P7:property intermediate_out; @(posedge clk) disable iff (rst) (ld==1) && aes_cipher_top.dcnt == 4'h0b |->text_out[127:120] != aes_cipher_top.sa00_sr ^aes_cipher_top.w0[31:24]; endproperty check_test3: assert property(intermediate_out)		

```
1   clear -all
2
3   set ROOT_PATH [pwd]
4   set RTL_PATH ${ROOT_PATH}/verilog
5
6   analyze -v2k   ${RTL_PATH}/aes_sbox.v                        1
7   analyze -v2k ${RTL_PATH}/aes_rcon.v
8   analyze -v2k ${RTL_PATH}/aes_key_expand_128.v
9   analyze -v2k ${RTL_PATH}/aes_cipher_top.v
10
11
12  elaborate -top {aes_cipher_top}                              2
13
14
15  clock clk                                                    3
16
17
18  reset rst                                                    4
19
20
21  check_spv -create -from {key text_in} -to {text_out}         5
22  check_spv  -create -from {ld} -to {done }
23  check_spv -create -from {key } -to { text_out }
24  check_spv -create -from {text_in } -to { text_out }
25
26
27  prove -all                                                   6
28
```

Fig. 7.5 The TCL script used for running the formal analyses to verify the defined assertions. The script consists of all commands required for setting up the SPV environment and proving the assertions

7.4.5 Tool 1: JasperGold Security Path Verification (SPV)

JasperGold SPV is an application in the JasperGold suite of tools, designed specifically for formal verification of the presence of information leakage paths. There is a predefined SPV template that can be utilized for formal check of information leakage properties. JasperGold SPV can read in a TCL file containing the various commands for compiling and elaborating the design, defining the clock and reset for the design, reading the various SPV assertions to be checked, and then proving the included assertions. A sample script is shown in Fig. 7.5:

Step 1: Setting Up JasperGold Environment JasperGold SPV is capable of reading in the RTL design and elaborating to form the entire hierarchy. Using the analyze command as highlighted in box 1 of Fig. 7.5, JasperGold can read in and compile the hardware design indicated by the path. JasperGold is capable of analyzing VHDL and SystemVerilog designs along with RTL designs. By changing the flag described below, JasperGold examines the design written in any of the above hardware languages:

- *-v2k*: Used for analyzing Verilog 2001 standard design files
- *-verilog*: Used for analyzing Verilog 1995 standard design files
- *-vhdl*: Used for analyzing VHDL design files
- *-sv*: Used for analyzing SystemVerilog design files

Once the design files have been analyzed, JasperGold elaborates on the complete design to unroll the design hierarchy. The command for elaborating a design using JasperGold SPV is highlighted in box 2 in Fig. 7.5. Once the design hierarchy is elaborated, the user needs to define the clock and reset signals for the input design. The "clock" command, highlighted in box 3, is used to specify a global clock configuration. Users can use this command to list all the clock configurations and analyze the clock tree. The analysis can properly provide the clock environment information for property checking. The "reset" command highlighted in box 4 of Fig. 7.5 commands to specify the reset condition for the design under verification. Users can also set an active low reset using the "-expression" flag while defining the reset signal and inputting an active low signal.

Step 2: Analyzing Assertions to Be Checked and Running the Tool The above set of commands sets up the JasperGold SPV environment for the formal analysis. The next step is to input the assertions that need to be checked. This is done by specifying each assertion to be checked by JasperGold in the input TCL script after setting up the environment. As seen in box 5 in Fig. 7.5, the user defines various assertions that need to be checked by the JasperGold SPV. In the figure, we input the four assertions we intend to check using the SPV application. The user then conveys to the tool to prove all the assertions defined using the command shown in box 6 in Fig. 7.5

The above two steps instruct the user on creating a simple TCL script that can set up the SPV environment for a given design and how to read the assertions to be checked and prove them. Users can invoke the JasperGold SPV Tool Gui, shown in Fig. 7.6. Once invoked, the user can instruct the tool to read the TCL script created, as highlighted in box 1 of Fig. 7.6. On running the "source" command, SPV constructs the environment, reads in the assertions, and runs its formal analysis to prove the input assertions.

Once the TCL script has been processed, the user will see a screen similar to Fig. 7.7. Box 1 highlighted in Fig. 7.7 shows the hierarchy of the compiled design, as discussed in step 1. The user can identify each instantiation of a module and see signal values for each instantiation in a waveform. The SPV tool's assertions that are analyzed and formally proven are highlighted in box 2 in Fig. 7.7. The assertions can have two possible results:

- *Green Tick*: A green tick next to an assertion indicates that the tool has formally proven the assertion. This means that SPV could not find any information leakage path from the source (security asset) to the destination (output port).
- *Red Arrow*: A red arrow next to an assertion indicates that the tool could not formally prove the assertion. This means that SPV was able to identify a path that leaks information from the source (security asset) to the destination (output port). The tool provides a counterexample, showing the exact leakage path.

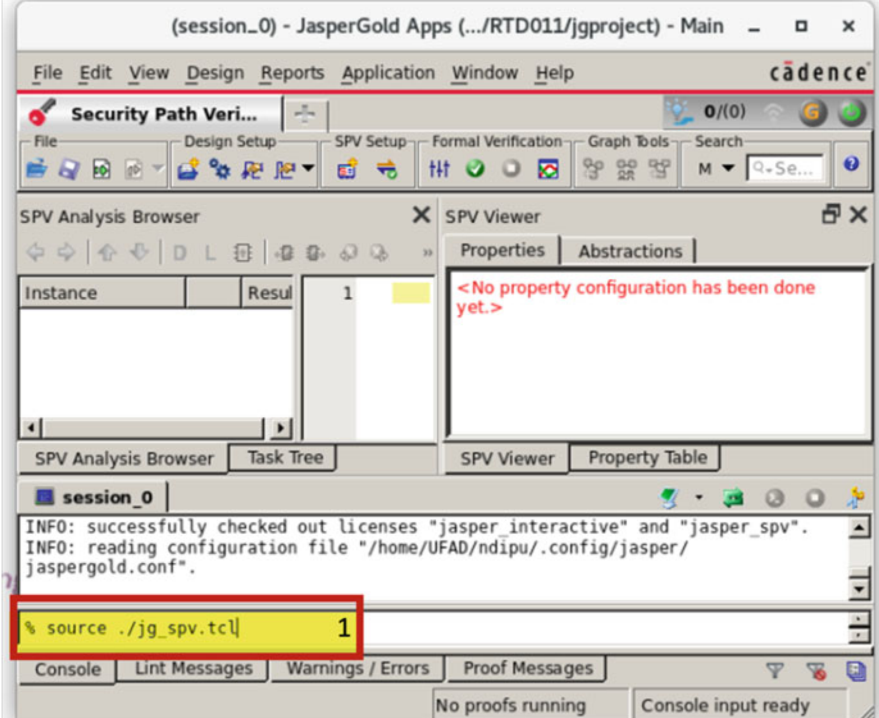


Fig. 7.6 JasperGold SPV GUI when invoked by the user

The four assertions we determined for the AES design failed the formal analysis. This indicated that leakage paths leak the key and plaintext values from the input port and corresponding registers to the output port. A summary of the formal analysis, showing the number of assertions analyzed and the number of assertions passed/failed, is highlighted in box 3 in Fig. 7.7.

7.4.6 Tool 2: JasperGold Formal Property Verification

Similar to JasperGold SPV, the FPV application is utilized for formal analysis. However, it is constrained like the SPV application and can be used to verify properties for various threat models. Also, unlike SPV, no fixed template can be used. The user needs to be able to define the specific property in the form of a SystemVerilog assertion (SVA). FPV also can read in a TCL file containing the various commands for setting up the FPV environment, analyzing property and bind files, reading the SVA assertions to be checked, and then proving the included assertions:

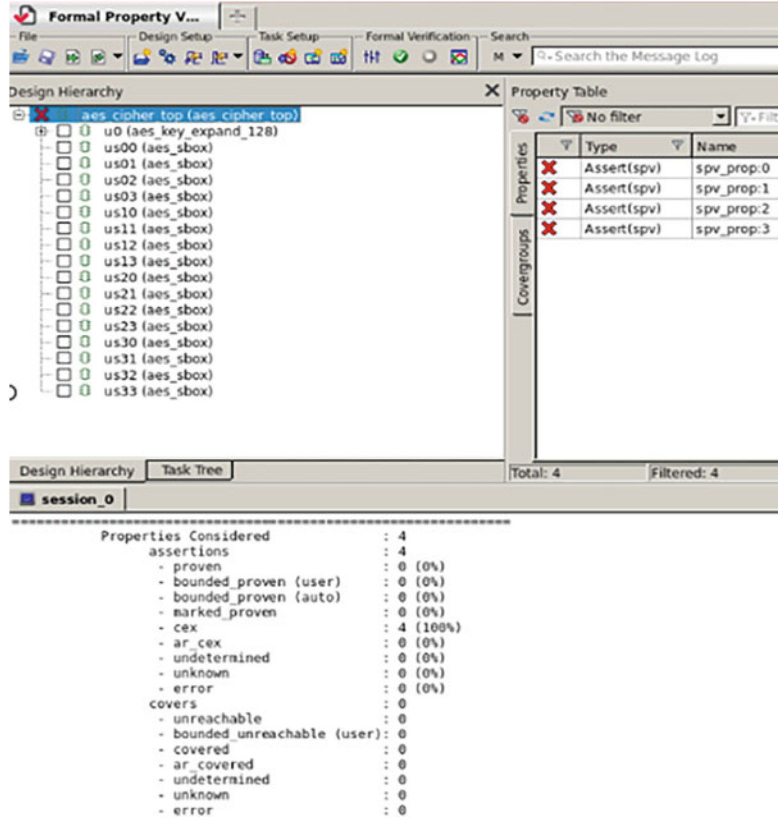


Fig. 7.7 JasperGold SPV GUI after formal verification. The shell shows the hierarchy of the design, the status of properties proven, and a summary of the complete formal analysis

Step 1: Create Property and Bind Files FPV analysis requires all properties to be expressed as SystemVerilog assertions. Defining the SVA assertions needs to create a property file (.sva). The property file looks similar to a Verilog module file, and the property file for property P7 for the AES design is shown in Fig. 7.8.

The module defined in the property file is incorporated into the design hierarchy, where it reads all input values read by design under consideration. The property file-defined module also reads the output port values from the design. The property file module reads in all design port values, and the defined properties are then evaluated using these values. The property file module definition is similar to the design under consideration, except all the I/O ports defined for the design are input ports to the property module file. This is highlighted in box 1 in Fig. 7.8. Once the ports are restricted, the user can determine the various SVA properties to be checked, as highlighted in box 2 of Fig. 7.8.

Once the property file is defined, the user must bind the property file module to the design under consideration. Binding is required to determine the port

```
module v_aes_cipher_top(clk, rst, ld, done, key, text_in, text_out );
input              clk, rst;
input              ld;
input             done;
input   [127:0] key;
input   [127:0] text_in;
input  [127:0] text_out;


  property intermediate_out;
    @(posedge clk) disable iff (rst) (ld==1 ) && aes_cipher_top.dcnt == 4'h0b  |-> text_out[127:120] !=
    aes_cipher_top.sa00_sr ^ aes_cipher_top.w0[31:24];
  endproperty

  check_test3: assert property(intermediate_out);


endmodule
```

Fig. 7.8 Sample property file module as required for the FPV analysis

```
module binding_module();
bind aes_cipher_top
  v_aes_cipher_top s_aes_cipher_top
  (.clk(clk), .rst(rst), .ld(ld), .done(done), .key(key), .text_in(text_in), .text_out(text_out));
endmodule
```

Fig. 7.9 Sample binding module required for binding the design under consideration to the property file module

connections between the property file and the design under consideration. The bind file (.sva) for our AES example is shown in Fig. 7.9.

Step 2: Defining the FPV Environment and Running the Tool Once the property and bind files have been defined, the user can set up the FPV environment. The FPV environment is the same as the SPV environment setup, as discussed in step 1 of Sect. 7.4.5, with one additional step. The FPV environment requires the compilation of the property and bind files to perform the formal analysis. FPV can analyze and compile the property and bind files utilizing the "analyze" command, as highlighted in box 1 in Fig. 7.10 The user can assess the general complexity of formal analysis by getting various design information such as the number of gates, flops, set of counters, any special values of the counters, the set of finite-state machines, etc. This can give the user a brief overview of the design complexity and, thus, formal analysis complexity. A user can gain this information using the "$get_{d}esign_{i}nformation''$ command, as highlighted in box 2 in Fig. 7.10. The user can set various conditions for property verification, such as

- *set_max_trace_length*: Specify the maximum length for trace limit for the proof depth. If one engine reaches the length limit for some property, all other engines will stop working on that property.
- *set_prove_per_property_time_limit*: Used to specify the maximum time the tool spends in proving any individual assertion.
- *set_engine_mode*: To select default engines for proving properties. FPV has multiple engines that can be chosen, each tuned for different properties.

All the above optional commands can be seen highlighted in box 3 in Fig. 7.10. The above steps instruct the user on creating a simple TCL script that can set up the

```
clear -all

# Analyze design under verification files
set ROOT_PATH /home/UFAD/ndipu/formal_verification/dipu/aes_core
set RTL_PATH ${ROOT_PATH}/verilog
set PROP_PATH ${ROOT_PATH}/properties

analyze -verilog \
  ${RTL_PATH}/aes_key_expand_128.v \
  ${RTL_PATH}/aes_rcon.v \
  ${RTL_PATH}/aes_sbox.v \
  ${RTL_PATH}/aes_cipher_top.v


# Analyze property files                          1
analyze -sva \
 ${PROP_PATH}/v_aes_128.sva
 ${PROP_PATH}/bindings.sva \


# Elaborate design and properties
elaborate -top aes_cipher_top
#elaborate


# Set up Clocks and Resets
clock clk
reset rst


# Get design information to check general complexity
get_design_info              2


# Prove properties
# 1st pass: Quick validation of properties with default engines
set_max_trace_length 10                           3
prove -all

#
# 2nd pass: Validation of remaining properties with different engine
set_max_trace_length 50
set_prove_per_property_time_limit 30s
set_engine_mode {K I N}
prove -all


# Report proof results
report
```

Fig. 7.10 Sample FPV script used to run FPV check for the assertions defined for the AES module

FPV environment for a given design, creating the property module along with the binding file, and how to read the assertions to be checked and proving them. For each assertion given as input, FPV also generates a cover statement for the antecedent of the assertion. This is to ensure that the antecedent condition is covered, i.e., it occurs at least once in the design before proving the property.

Users can invoke the JasperGold Tool SPV Gui, shown in Fig. 7.11. Once invoked, the user can instruct the tool to read the TCL script created, as highlighted in Fig. 7.11. On running the "source" command, FPV constructs the environment, reads in the assertions, and runs its formal analysis to prove the input assertions.

Once the TCL script has been processed, the user will see a screen similar to Fig. 7.12. Similar to the SPV tool, the FPV tool GUI also outputs the hierarchy of

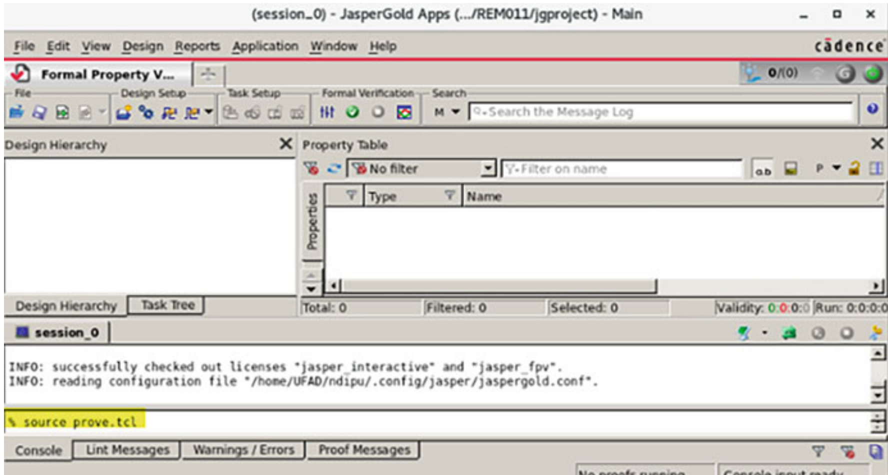


Fig. 7.11 JasperGold FPV GUI when invoked by the user

the design, assertions analyzed and formally proven, and the summary of the proof. Assertions analyzed by the FPV tool can either pass or fail.

- *Pass (Green Tick)*: The FPV tool formally proved the assertion. Thus, the design adheres to the behavior described by the assertion under all circumstances.
- *Fail (Red Cross)*: The FPV tool could not formally prove the assertion. This means the tool found a circumstance under which the behavior described by the tool has been violated. A counterexample waveform with the simulation for the failure is shown.

For our AES design, we utilized the FPV tool to check the validity of behavior as defined by property P7. We observe that the cover statement is proven. Hence, the antecedent condition is met. However, the assertion indicates a condition in which the intermediate result leaks to the output port before the completion of the AES operation.

7.5 Conclusion

The purpose of this chapter is to help readers to learn about how to use property-based verification for the security validation of designs. This chapter provides the framework required for generating properties for security validation. We describe in detail how to identify the security assets and threat models needed for formulating the security properties and then developing security properties for security validation. We took an example AES design and, using the defined framework, identified the design's various security assets and threat models. Then we developed natural

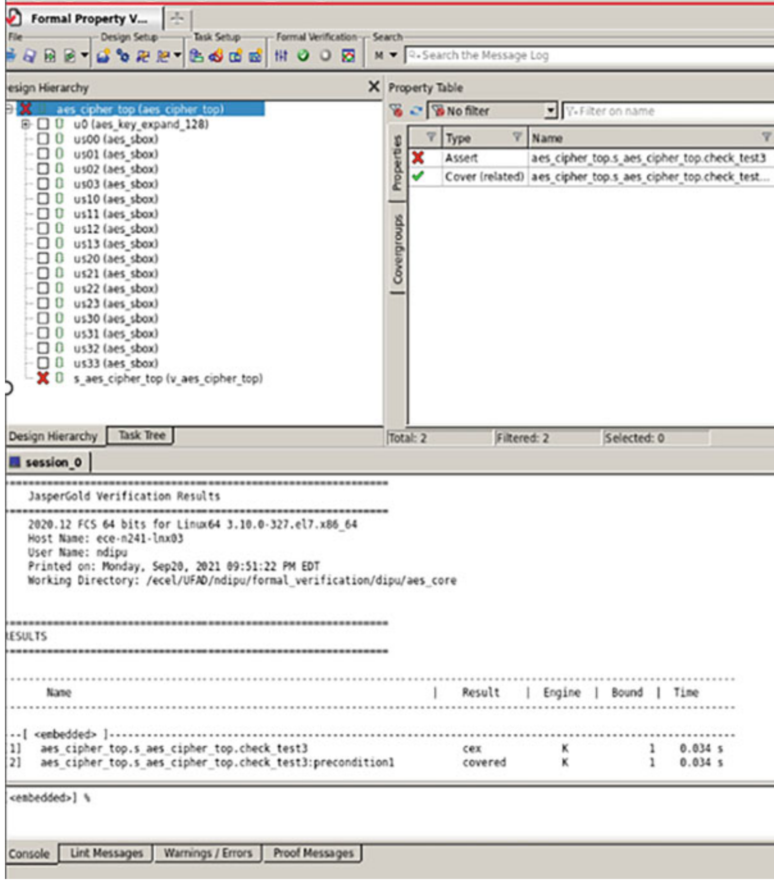


Fig. 7.12 JasperGold FPV GUI with results for the AES design

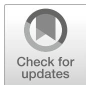
language properties defining the secure behavior of the identified security assets. These properties were then converted into tool-understandable assertion form. The chapter then dives into two formal tools, JasperGold SPV and JasperGold FPV, explaining the different types of assertions that can be proven using these tools. The chapter then details a step-by-step process on how to verify the assertions that were generated. We hope this framework will help us understand how to develop security properties focusing on different threat models and protecting various assets and then how to convert those properties into assertions and use industry-standard verification tools to verify the assertions from a security perspective formally.

References

1. Ahmed, A., Farahmandi, F., Iskander, Y., Mishra, P.: Scalable hardware Trojan activation by interleaving concrete simulation and symbolic execution. In: 2018 IEEE International Test Conference (ITC), pp. 1–10. IEEE (2018)
2. Anandakumar, N.N., Dillibabu, S.: Correlation power analysis attack of AES on FPGA using customized communication protocol. In: Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology, CCSEIT '12, pp. 683–688 (2012)
3. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: FPGA-based true random number generation using programmable delays in oscillator-rings. IEEE Trans. Circuits Syst. Express Briefs **67**(3), 570–574 (2020). https://doi.org/10.1109/TCSII.2019.2919891
4. ARM, L.: Arm security technology-building a secure system using trustzone technology. Tech. rep., PRD-GENC-C. ARM Ltd. Apr.(cit. on p.), Tech. Rep (2009)
5. Biswas, L.K., Lavdas, L., Rahman, M.T., Tehranipoor, M., Asadizanjani, N.: On backside probing techniques and their emerging security threats. IEEE Des. Test **39**(6), 172–179 (2022). https://doi.org/10.1109/MDAT.2022.3185797
6. Cadence: JasperGold Formal Propert Verification. URL https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/formal-property-verification-app.html
7. Cadence: JasperGold Security Path Verification. URL https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/security-path-verification-app.html
8. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model checker. Int. J. Softw. Tools Technol. Transfer **2**(4), 410–425 (2000)
9. Cycuity: Cycuity Radix Solutions
10. Eslami, M., Ghavami, B., Raji, M., Mahani, A.: A survey on fault injection methods of digital integrated circuits. Integration **71**, 154–163 (2020). https://doi.org/10.1016/j.vlsi.2019.11.006. URL https://www.sciencedirect.com/science/article/pii/S016792601930402X
11. Farahmandi, F., Morad, R., Ziv, A., Nevo, Z., Mishra, P.: Cost-effective analysis of post-silicon functional coverage events. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 392–397. IEEE (2017)
12. Farzana, N., Ayalasomayajula, A., Rahman, F., Farahmandi, F., Tehranipoor, M.: SAIF: Automated asset identification for security verification at the register transfer level. In: 2021 IEEE 39th VLSI Test Symposium (VTS), pp. 1–7. IEEE (2021)
13. Farzana, N., Rahman, F., Tehranipoor, M., Farahmandi, F.: SoC Security verification using property checking. In: 2019 IEEE International Test Conference (ITC), pp. 1–10. IEEE (2019)
14. Foster, H.D.: Property specification: the key to an assertion-based verification platform. In: Proceedings of Electronic Design Processes (EDP) Workshop (2003)
15. Gomina, K., Rigaud, J.B., Gendrier, P., Candelier, P., Tria, A.: Power supply glitch attacks: design and evaluation of detection circuits. In: 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 136–141 (2014). https://doi.org/10.1109/HST.2014.6855584
16. Gruninger, M., Menzel, C.: The process specification language (PSL) theory and applications. AI Mag. **24**(3), 63–63 (2003)
17. Guo, X., Dutta, R.G., Jin, Y., Farahmandi, F., Mishra, P.: Pre-silicon security verification and validation: a formal perspective. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2015). https://doi.org/10.1145/2744769.2747939
18. JasperGold Formal Fundamentals Cadence . URL https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html
19. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: an experimental study of dram disturbance

errors. SIGARCH Comput. Archit. News **42**(3), 361–372 (2014). https://doi.org/10.1145/2678373.2665726. URL https://doi.org/10.1145/2678373.2665726
20. Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: exploiting speculative execution. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1–19 (2019). https://doi.org/10.1109/SP.2019.00002
21. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown: reading kernel memory from user space. In: 27th USENIX Security Symposium (USENIX Security 18) (2018)
22. Nahiyan, A., Sadi, M., Vittal, R., Contreras, G., Forte, D., Tehranipoor, M.: Hardware Trojan detection through information flow security verification. In: 2017 IEEE International Test Conference (ITC), pp. 1–10. IEEE (2017)
23. Nahiyan, A., Xiao, K., Yang, K., Jin, Y., Forte, D., Tehranipoor, M.: AVFSM: a framework for identifying and mitigating vulnerabilities in FSMs. In: Proceedings of the 53rd Annual Design Automation Conference, pp. 1–6 (2016)
24. Ning, B., Liu, Q.: Modeling and efficiency analysis of clock glitch fault injection attack. In: 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), pp. 13–18 (2018). https://doi.org/10.1109/AsianHOST.2018.8607175
25. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (SFCS 1977), pp. 46–57. IEEE (1977)
26. Ray, S., Peeters, E., Tehranipoor, M.M., Bhunia, S.: System-on-chip platform security assurance: architecture and validation. Proc. IEEE **106**(1), 21–37 (2018). https://doi.org/10.1109/JPROC.2017.2714641
27. Synopsys: Synopsys VC Formal
28. Vijayaraghavan, S., Ramanathan, M.: A Practical Guide for SystemVerilog Assertions. Springer Science & Business Media, Berlin (2005)
29. Wang, H., Forte, D., Tehranipoor, M.M., Shi, Q.: Probing attacks on integrated circuits: challenges and research opportunities. IEEE Des. Test **34**(5), 63–71 (2017). https://doi.org/10.1109/MDAT.2017.2729398

Chapter 8
Power Analysis Attacks on AES



8.1 Introduction

Hardware security is a domain of enterprise security that focuses on protecting all hardware devices including microcontrollers, FPGA, and ASICs, among other similar hardware. Hardware-oriented security research focuses mainly on exploring both attack and defensive aspects of hardware devices. Traditionally, the main task of cryptographic hardware is the acceleration of operations frequently used in cryptosystems. In applications, hardware devices are also required to store secret or private keys securely. Hence, a cryptographic device must prevent the extraction and other sensitive information [25]. Naturally, to achieve the goal of building defensive capabilities in hardware, one must first understand and be aware of the exploits it is susceptible to. One such exploit which may be applied to hardware security is side-channel attacks (SCA) [26]. These attacks pose a serious threat to the security of systems and cryptography libraries. Indeed, many side-channel analysis techniques have proven successful in breaking algorithmically robust cryptographic functions/operations (such as encryption, key exchange, and signature generation) and extracting the secret key. A program or its code is not directly targeted by a side-channel attack. Instead, a side-channel attack uses measurements to extract secret information from a device or system by analyzing various physical parameters [1, 7]. Examples of such parameters include heat, sound, time, electromagnetic emission, and power consumption.

Power analysis attacks have gained much interest in the cryptography community since they were first published in 1998. They have so far been used successfully in a variety of (unsecured) public-key and symmetric cryptographic algorithm implementations. Power analysis attacks are used to gain sensitive information by observing a device's power consumption [19, 23]. The attack is noninvasive, and it requires physical access to the target device. Typically, this is done by incorporating a current path at Vdd or Gnd pin of the chip that is performing the cryptographic operation, to capture power dissipation for such an operation while the device is

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_8

undergoing an operation. The device's power consumption captures the switching activity of the relevant transistors, which depends on inputs to a cryptographic function, such as the plaintext and the key.

As part of the modern design flow, FPGAs are becoming increasingly important. This is due to the fact that they are inexpensive and have access to a variety of EDA tools. An FPGA implementation of a circuit can be easily ported from high-level descriptions (such as VerilogHDL). Naturally, it is also important to test the developed circuits for resistance to power analysis attacks using the resulting FPGA implementation. In this chapter, we focus on correlation power analysis attacks on AES crypto hardware. In particular, this chapter can help a reader to better understand and will gain hands-on experience on how to obtain a secret key from advanced encryption standard (AES) block cipher using correlation power analysis on FPGA step by step. The most common and useful power analysis attack against cryptographic blocks is correlation power analysis (CPA). The correlation between the device's power consumption and the data produced by the computation is used by attackers.

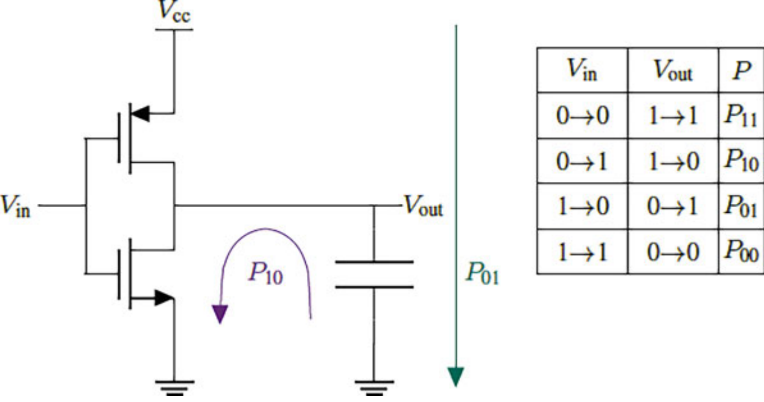
The remaining chapters are structured as follows: A detailed introduction to power analysis attacks will be given in Sect. 8.2. An explanation of the various power analysis attacks will be given, so that the context of SPA, DPA, and CPA attacks can be better understood. Introduction to FPGA and AES algorithm overview will be given 8.3. The experimental setup for power capture will be given in Sect. 8.4. Power measurements on the AES Chip will be given in Sect. 8.5. Performing AES CPA attack will be described in Sect. 8.6. Section 8.7 will wrap up this chapter.

8.2 Power Analysis Attacks

Kocher et al. [14] first described the powerful side-channel attack known as the power analysis attack. Attackers using power analysis techniques must measure a device's power consumption without actively manipulating it, i.e., by using the target device in its intended mode (i.e., passive mode). For instance, in the case of attacks against smart cards, the attacker can choose to allow the device to carry out an internal authenticate command. While the device is executing this command, the attacker monitors the power usage of the device. A power trace is a representation of every power signal recorded throughout the analysis. After that, statistical techniques enable effectively the extraction of information on the secret key that is contained in the power trace.

8.2.1 Power Consumption Characteristics of CMOS

Nowadays, CMOS (complementary metal-oxide semiconductor) technology is used to implement nearly all computer and mobile processors. The CMOS technology uses Vss and Vdd to represent the numbers 0 and 1, respectively. The power



V_{in}	V_{out}	P
0→0	1→1	P_{11}
0→1	1→0	P_{10}
1→0	0→1	P_{01}
1→1	0→0	P_{00}

Fig. 8.1 Example of a CMOS inverter circuit [7]

consumption in an integrated circuit is dynamic and dependent upon the operations that are taking place inside of the circuit [21]. This can be better understood when looking at a single gate. Figure 8.1 shows an inverter circuit with a bypass capacitor. The table shows the possible transitions that can occur between two clock cycles. Depending on the transition, power consumption can take on one of four states. Power is only consumed when the states change; the corresponding states are represented in the table by P01 and P10 and represented by green and violet arrows in the figure. There is obviously more than one gate in an integrated circuit, but the basic principle remains the same. These transitions are determined by the operations taking place in the device and the values that are being processed. The simultaneous switching of the gates on every rising edge of the clock results in a current flow that can be seen through both Vdd and Vss. By, for instance, connecting a tiny resistor between the device's Vss or Vdd and the genuine Vdd, this current flow can be seen on the outside of the device. A digital oscilloscope can measure the voltage produced by the resistor's current flow.

As the predominant type of SCA, power analysis attacks have received substantial investigation from academic and industrial researchers. Different power analysis attacks have been developed to reveal important information about the target device, including differential power analysis (DPA), simple power analysis (SPA), and correlation power analysis (CPA) [6, 15]. A set of power measurements is required for each side-channel analysis to be applied; these sets vary in scope and form, depending on the type of attack, the complexity of the design, and the accuracy of the data collection process. An attacker usually needs to use a large number of power traces in all attack modes before applying the power analysis attack.

8.2.2 Simple Power Analysis (SPA)

A method called simple power analysis (SPA) includes directly evaluating power consumption measurements that is gathered during cryptographic operations [5].

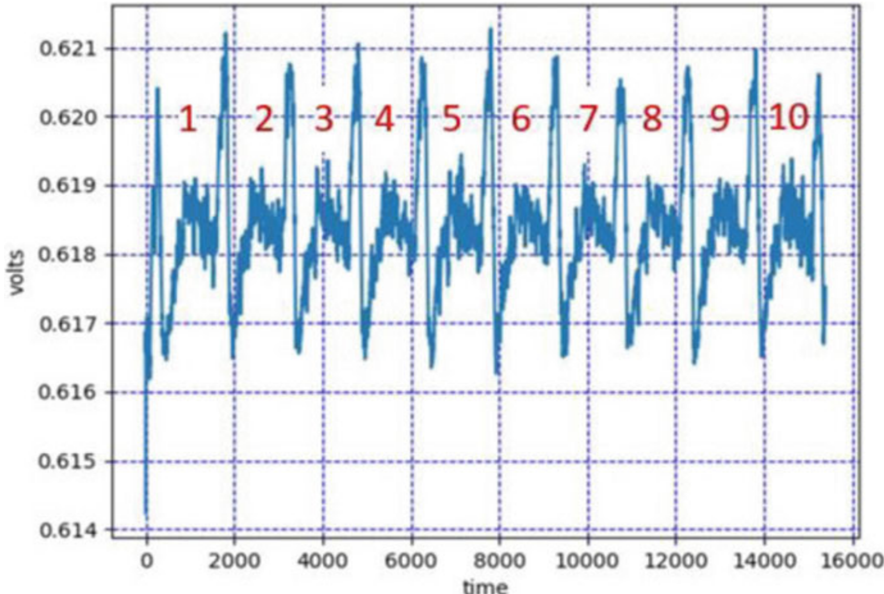


Fig. 8.2 SPA trace showing an AES operation [24]

By performing a side-channel attack, which entails visually inspecting graphs of the current consumed by a device over time, SPA can gather details about a device's operation as well as secret information [4]. The device uses varied amounts of power depending on the operation it is performing. A CPU, for instance, will have varying power consumption profiles depending on the instructions it executes. For example, one can distinguish a multiplication function from an addition function, since multiplication consumes more current than addition. Also, when reading data from a memory, the ratio of 1s vs. 0s will be reflected in the power profile. With a standard oscilloscope, one can capture the resulting current signature and from it deduce the type of operation. For example, Fig. 8.2 reveals the ten rounds of the advanced encryption standard (AES) [9].

SPA takes use of the fact that a device's consumption usage is based on the operation being carried out inside the device, while DPA exploits the data dependence of the device.

8.2.3 Differential Power Analysis (DPA)

The most common power analysis attack is a DPA attack [16, 22]. DPA attacks aim to discover secret cryptographic device keys by collecting a large number of power traces when the devices encrypt or decrypt various data blocks. DPA attacks have the

primary benefit over SPA attacks in that in-depth understanding of the cryptographic device is not required [16]. Adversaries gather a collection of power traces as part of a conventional DPA attack and then utilize statistical techniques to determine the key using the observed power traces. By examining how input data affect the observed value, they can infer the secrets. This method computes the difference between the average of many traces of two sets of data. If the difference is close to zero, then the two sets are not correlated. If the sets are correlated, then the difference will be a nonzero number. Regardless of the amount of noise present in the system, even minute correlations can be detected with enough traces because the noise will be practically cancelled out during the averaging. Normally, the encryption key is a 128-bit value. In order to test every single value, it would take 2^{128} attempts which is pretty much impossible to do. However, the 128-bit AES key can be divided into 16 bytes, and each byte can be solved separately. It would only take 2^8 or 256 attempts for each byte attack, which means it would only take 16×256 or 4096 attempts to be able to decipher the complete encryption key.

To see how this can be used, take, for example, the advanced encryption standard (AES). The equation for the encrypted data is given by $Output = S[X_n \oplus K_n]$, where S is a look-up table and $\oplus$ is the XOR of a known input X_n and the encryption key K_n. We try a few different hypotheses in order to figure out the value of K_n. The first set of traces belongs to the set where the output's LSB is "0," while the second set of traces belongs to the set when the output's LSB is "1." The difference between the two sets' averages is then examined. Here, we have a trace showing (see Fig. 8.3) the results of five different K_n values, where the correct key corresponds to the third trace.

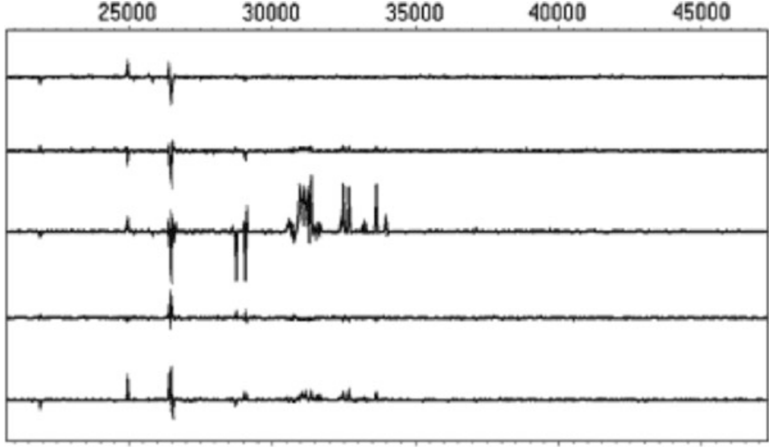


Fig. 8.3 DPA result for different key values [13]

8.2.4 Correlation Power Analysis (CPA)

Correlation power analysis (CPA) is an attack that allows us to find a secret encryption key that is stored on a victim device. The challenge for an attacker now is to effectively exploit the secret information about the secret key that is kept on a victim device. The attacker builds a hypothetical model of the device for this reason. This hypothetical model describes the device's instantaneous power consumption when it performs certain cryptographic encryption. It is necessary to guess at least a small portion of the unknown key for this purpose. Thankfully, any algorithm used in practice only makes use of a small portion of the secret key at once (similar to DPA where each byte can be solved individually). The hacker creates a straightforward computer program that runs the algorithm and attacks discrete pieces (subkeys) of the secret key. The attacker considers every possible option for the subkey. For each guess and each trace, use the known plaintext and the guessed subkey to calculate the power consumption according to our model. In the final phase of the attack, the attacker feeds the same input values which he used in the model to the real device and evaluates its power consumption. Then the attacker compares the model's predictions to the actual power consumption values [2]. For every incorrect key guess, the predictions will not agree with the actual measurements, but for the right key guess, the correlation trace will show a peak. A more advanced technique is a CPA attack which detects the keys by analyzing the correlation between the hypothetical power model and the power dissipation of the device, as illustrated in Fig. 8.4.

In this chapter, we implement the most widely used block cipher AES [16] on the ChipWhisperer CW305 FPGA board and perform correlation power analysis (CPA) attack and reveal the AES cryptography key. In our attack, we can be incorporated

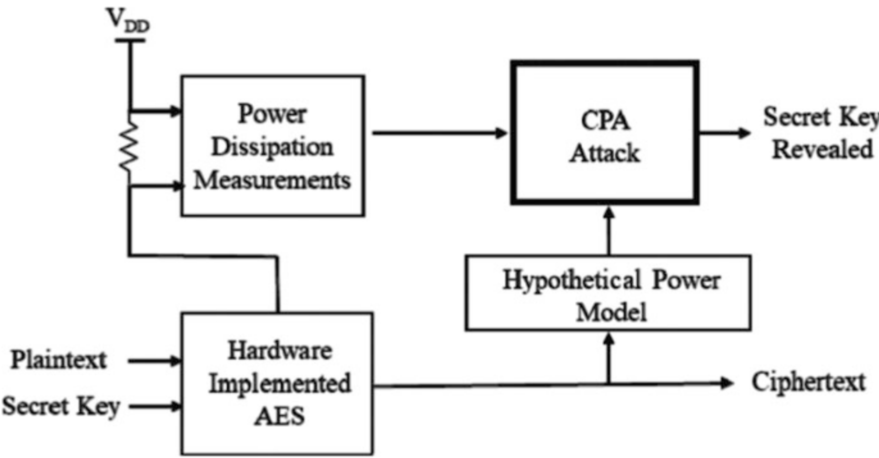


Fig. 8.4 Illustration of a CPA attack [20]

into the leakage model by calculating the Hamming distance between the prior value in the register and the new value (i.e., the Hamming weight of the two values XOR'd), and we may incorporate our attack into the leaking model. The main concern with attacks on hardware AES on FPGA is the Hamming distance of registers. In contrast, microcontrollers often set their register bits to a value between 0 and 1 before updating them because doing so will typically conserve power and reduce the voltage swing when changing a value [8].

8.3 AES Implementation on FPGA

8.3.1 Field-Programmable Logic Arrays

An FPGA consists of an array of configurable logic blocks (CLBs), surrounded by programmable I/O blocks and connected with programmable interconnections as shown in Fig. 13.3. The amount of logic blocks and flip-flops in a standard FPGA ranges from 64 to tens of thousands. A 100% connectivity between the logic blocks is not typically provided by FPGAs. The logic is instead placed and routed on the device by complex software.

Static random-access memory (SRAM), a technology akin to microprocessors, is used in the construction of the highest-density FPGAs. The other widely used

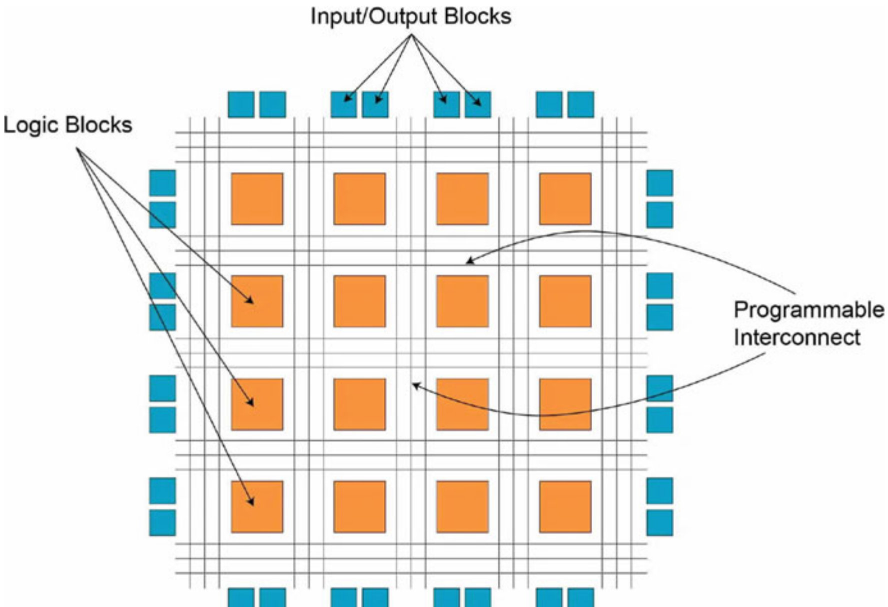


Fig. 8.5 The FPGA architecture

process method is known as anti-fuse, which offers the advantages of a larger number of programmable interconnects. Even in-system reprogramming is possible with SRAM-based devices by nature. It is necessary to load the program data specifying the logic configuration in the SRAM after applying power to the circuit [17]. The FPGA either loads its configuration memory on its own or downloads it via an external CPU. Anti-fuse devices, on the other hand, allow for one-time programming (OTP). They cannot be changed once programmed, but they also keep their program even after the power is turned off. Either the end user, the manufacturer, or the distributor can program anti-fuse devices in a device programmer.

8.3.2 AES Algorithm Overview

AES is a symmetric-key algorithm, which means the same key is used for both encryption and decryption. It was chosen as the successor of the Data Encryption Standard (DES) and named advanced encryption standard (AES) by the National Institute of Standards and Technology (NIST). AES is a subset of the Rijndael block cipher [9]. The NIST selected three members of the Rijndael family each having a 128-bit block size but with an optional 128-bit, 192-bit, or 256-bit key size. For a complete description and explanation of AES, please refer to [9]. In our case, we will be attacking a target that uses AES-128. This is a version of AES with a key length of 128 bits, which is 16 bytes. The plaintext and ciphertext length are also 128 bits. A high-level block diagram of an AES implementation that completes one round of AES per clock cycle is shown in Fig. 12.3. Internally, the AES operations are carried out on a two-dimensional array of bytes called the state. The input to the algorithm is the plaintext, arranged into the 4×4 state matrix. The 128-bit key can also be arranged into a 4×4 matrix of bytes. AES-128 consists of ten rounds. Before the rounds are carried out, the plaintext and secret key are XORed and stored into the State Register. After the necessary rounds of AES, the ciphertext is again loaded into the State Register. The First nine rounds consist of four stages: SubBytes, ShiftRows, MixColumns, and AddRoundKey. In the tenth round, the MixColumns operation is not performed as can be seen in Fig. 8.6. For encryption, each round consists of the following four steps:

- AddRoundKey is a function that bitwise XORs the state input with the round key.
- ShiftRows process where the bytes in a row are transposed by a predetermined number of positions along the row.
- MixColumns is a mixing operation where the four bytes in a column are merged by modulo multiplication with a fixed polynomial $C(x)$.
- Subbyte operation is the only nonlinear function that consists of multiplicative inverse in $GF(2^8)$ and a linear affine transformation. Please refer to [18], for a thorough explanation of how S-box implementations are employed in composite field inverters.

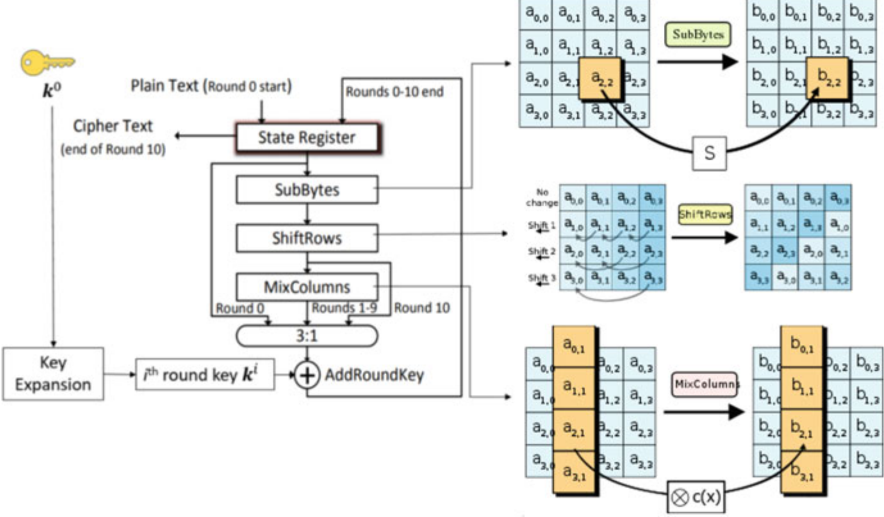


Fig. 8.6 Hardware AES block diagram

The output state of the State Register will not depend on a single byte of the key in the first transition because of the presence of MixColumns and another AddRoundKey, which significantly expands the attack's search space. As an alternative, the Hamming distance between the final two states—ciphertext and InvSubBytes (InvShiftRows (AddRoundKey (ciphertext)))—should be used as the leaking model for this implementation. We should emphasize that this is only one potential AES implementation. The last round is also the most straightforward to attack because it lacks the MixColumns function.

8.4 Experiment Setup

In this section, we will use the Chipwhisperer CW305 board and ChipWhisperer-capture devices and ensure it is correctly set up for power capture [22]. A standardized capture method for evaluating new power analysis algorithms is offered by the open-source ChipWhisperer project. For researching embedded hardware security, ChipWhisperer is a collection of numerous tools. There are ChipWhisperer hardware targets, ChipWhisperer target device firmware, ChipWhisperer target device FPGA blocks, and ChipWhisperer analysis software and libraries (which execute sampling of power measurements). As a standalone target, the CW305 board enables the use of a larger FPGA target to implement the AES core.

8.4.1 Hardware and Software

The following hardware, software, and equipment were used for the implemented attacks:

- Laptop/PC (i.e., installed Windows 10).
- The AES design is developed using Xilinx Vivado 2020.2 and coded in VerilogHDL, whereas Python is utilized for communication between the FPGA board and the PC using USB interface.
- Chipwhisperer software (v5.6.4): The ChipWhisperer software includes a Python API for talking to ChipWhisperer hardware (ChipWhisperer capture) and also a Python API for processing power traces from ChipWhisperer hardware (ChipWhisperer Analyzer).
- ChipWhisperer CW305 FPGA board: This is a target board in which we can implement crypto algorithms, and the board uses the Artix-7 FPGA.
- ChipWhisperer-Lite/Pro Capture board: This is a capture device that has gotten a new firmware update that gives it a USB-CDC serial port for talking over UART. The 10-bit analog-to-digital converter (ADC) on the NewAE ChipWhisperer-Pro and Lite capture boards has a sampling rate of 105 MS/s, while the buffer sizes are 98,119 and 24,573 samples, respectively. To interact with the workstation depicted in Figure [7], both can be linked via an SMA connector on the target board and a USB port.

8.4.2 Firmware Setup

All ChipWhisperer scopes and FPGA targets have a ROM base bootloader, meaning it is effectively impossible to brick the ChipWhisperer by updating or erasing its firmware. The firmware is small-footprint software usually found in embedded devices, for instance, the microcode of the hard drive. The bootloader is part of the firmware that usually ran during the boot sequence which allows loading new firmware to update it from SPI and USB. Three separate pieces of firmware are used on the ChipWhisperer hardware:

1. The capture board has a USB controller (in C).
2. An FPGA for high-speed captures (in Verilog) with open-source firmware.
3. The target device has its own firmware.

The above firmware is not automatically updated, but it tends to change less frequently. Many firmware examples for different targets are available in the repository: https://github.com/newaetech/chipwhisperer. First download the ChipWhisperer software from the following link, https://github.com/newaetech/chipwhisperer/releases, and install it on the control computer (PC). The ChipWhisperer Python library can be used for communication with both the NewAE Capture and target boards. Once installed the ChipWhisperer software, we may need

to update the ChipWhisperer firmware. For details to update the ChipWhisperer firmware, please refer to the following links:

1. https://chipwhisperer.readthedocs.io/en/latest/scope-api.html#api-scope-update
2. https://wiki.newae.com/Manual_SAM3U_Firmware_Update.

8.4.3 Hardware Setup

In this chapter, we'll make sure the CW305 target board [12] is configured properly for power capture by using it. The ChipWhisperer hardware consists of a target board and a capture board to record power traces. The setup for the experiment is shown in Fig. 8.7.

The CW305 FPGA board features a USB interface to talk to the FPGA, an external PLL for clocking the FPGA, and a programming VCC-INT supply. The *Algorithm Under Test* is the algorithm we want to test. The goal of performing side-channel power analysis on this method is supported by the remaining circuitry. We may easily load input, keys, output, or trigger operations by using the *Register*

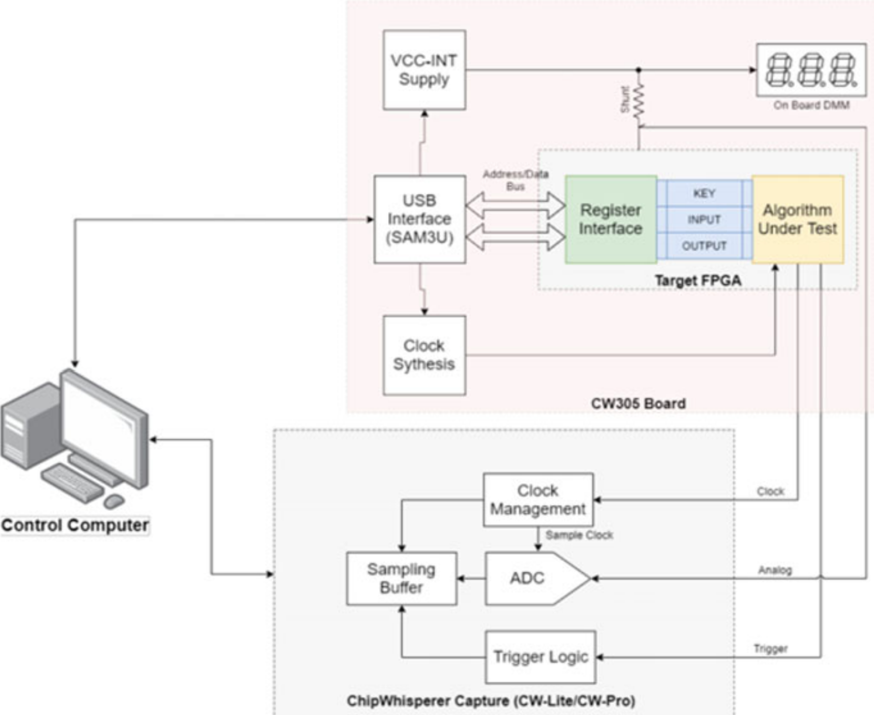


Fig. 8.7 CW305 board setup for power capture

Interface to match our Python code on the control computer. Physically, the CW305 acts as an address/data bus between the FPGA and the microcontroller for the USB interface. This address/data bus enables you to establish a conventional address/data bus on the FPGA instead and write any data into the FPGA.

8.4.3.1 CW305 Default Setup

The CW305 is a standalone FPGA target board as shown in Fig. 8.7. It features a USB interface for talking to the FPGA, an external PLL for clocking the FPGA, and a programming VCC-INT supply. The CW305 board is available in a number of configurations. It requires an external device for side-channel power analysis or fault injection and features the standard ChipWhisperer 20-pin/SMA interface. The CW305 consists of five configuration switches (see Fig. 8.8) such as the following: (1) DIP S1 switch (bottom-side, lower-left corner) is used to configure the FPGA bitstream mode. (2) DIP S2 switch (top-side, lower-right corner) is used to configure if the clock comes from the on-board PLL or from an external clock. (3) The small surface mount switch is used to configure the FPGA POWER. (4) The SPDT1 switch is used to select the VCC-INT power source. (5) Another SPDT2 switch is used to select the input power source. It decides whether to use the DC power jack or the USB-A connector to power the board. The readers are curious about how jumper and switch configurations work; for more information, view the complete documentation on https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA/.

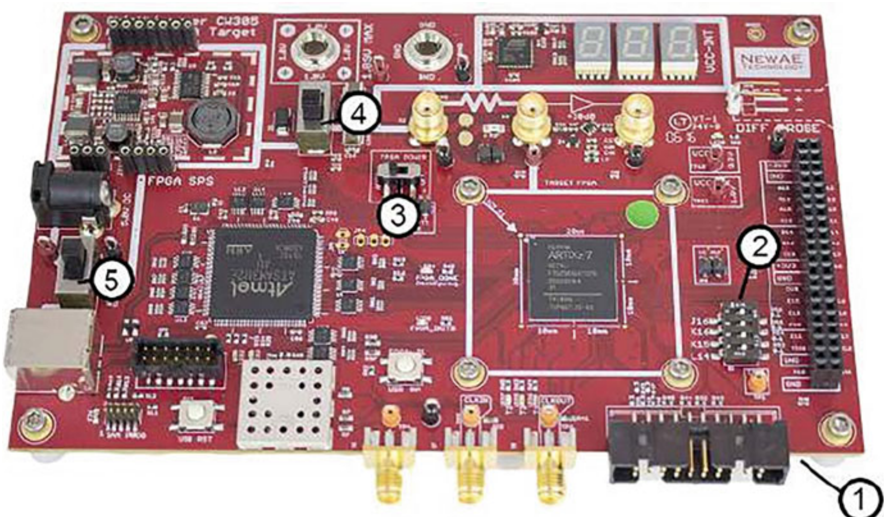


Fig. 8.8 The CW305 configuration switches

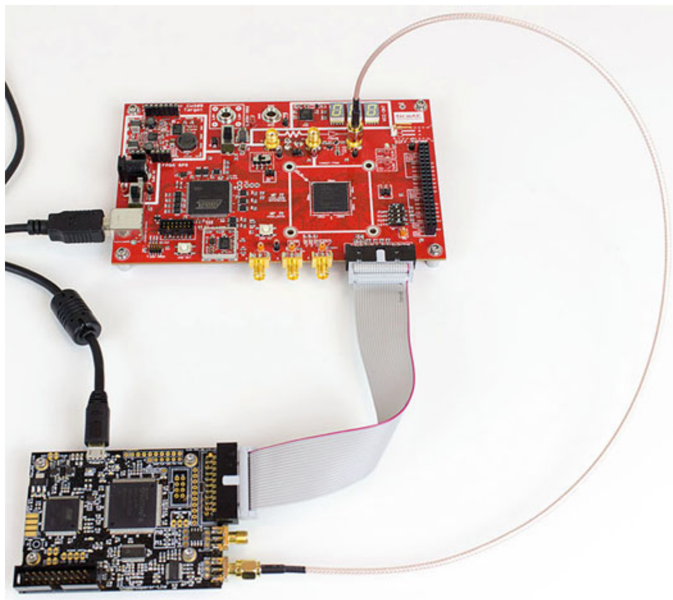


Fig. 8.9 The CW305 interconnected to the ChipWhisperer-Lite capture board [7]

8.4.3.2 Connect a CW305 board to a ChipWhisperer-Lite/Pro board

Simply attach a ChipWhisperer-Lite/Pro Capture board [10, 11] to the CW305 board as indicated in Fig. 8.9 once the CW305 board has been configured to the default settings above. In this situation, we only need to take the following actions:

- Turn off the CW305 board.
- Connect JP1 on the CW305 with the 20-pin "target connector" from the ChipWhisperer capture board.
- SMA cable should be connected from the "measure" SMA on the ChipWhisperer capture to the CW305's X4 (amplified shunt output).
- Utilize a Mini-USB to connect the ChipWhisperer-Lite or ChipWhisperer-Pro to a computer.
- Turn on the CW305 board (or plugin if not plugged in yet).

8.5 Power Measurements on the AES Chip

To get started, we need to make three connections to the CW305 target:

1. We will control the Artix-7 directly from the computer via USB. This is done through the USB-B port on the left side of the board. (On the target, the

Atmel SAM3U chip converts these USB packets into signals that the FPGA can understand.)
2. We will get control information for our power measurements through the 20-pin connector at the bottom of the board. This needs to be connected to the capture hardware (a ChipWhisperer Lite).
3. Our power measurements will be done through an SMA cable.

Bit file configuration, arming CWlite capture boards, and tracing from the CWlite board are also handled by our Python script. Then, the code triggers the encryption of a secret key and plaintext on the targeted hardware. At the end of the encryption, the program stops the capture and collects the power trace out of the ChipWhisperer-Lite.

8.5.1 AES Bitstream Generation

- The first thing we will need is the Xilinx Vivado tool for AES bitstream generation. In this connection, the fully featured versions of the Xilinx Vivado software with a license are required. However, the WEBPACK version is free for our target Artix-7 FPGA. To download and install Xilinx Vivado design suite, refer to the link https://www.youtube.com/watch?v=DIOIl3P65hg:
- Design entry can be done in various ways. The most intuitive method is drawing the schematics by connecting some predefined functional modules together. It is better and recommended to write our behavioral implementation in the form of hardware description languages (HDL) like Verilog and VHDL. For this tutorial, we will provide a preexisting AES-128 encryption example with a couple of project files to build a project using the Xilinx Vivado software. When we open the project file, we should be greeted with a screen as shown in Fig. 8.10. There are three steps that Vivado takes to turn our Verilog into a bitstream code.

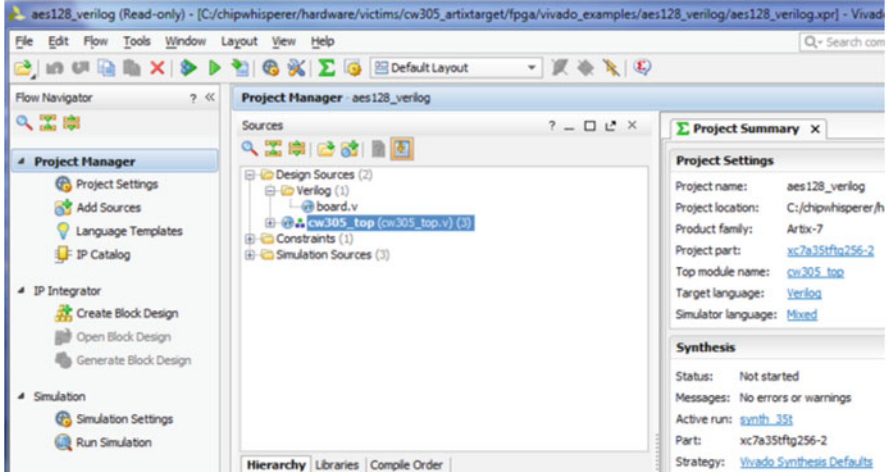


Fig. 8.10 Build a project using the Xilinx Vivado software

1. Synthesis: The Verilog code is synthesized into a gate-level representation. During the synthesis stage, the HDL code composed at the design entry stage will be converted into a circuit in the form of a netlist by the electronic design automation (EDA) tools. Our HDL code is going to be parsed to check syntax and then optimized to reduce redundant logic according to the specified settings. The generated netlist will contain the needed logic elements and the connectivity among them as described by the HDL code.
2. Implementation: The synthesized logic is routed to fit onto the device. The implementation phase will then technology map the logic elements in the netlist to the primitives available in the selected FPGA model so that the design could be implemented on our physical chip. Also, this step will place and route the primitives on the FPGA layout virtually per the constraints from designers and physical aspects to make the final design meet the power, performance, and area requirements.
3. Bitstream Generation: Finally, the placed and routed netlist will be translated to the binary configuration data, the so-called bitstream with the vendor-specific tool. The bitstream is stored in the desktop directory. For example:

```
D:AES_CPA\AES_cw305_top.bit
```

4. Then download it to the target device to fulfill the functionality. Bitstreams are to be stored in the FPGA, but they will be volatile meaning that once lost the FPGA loses power. Persistent storage is available on the CW305 in the form of an SPI flash chip.

8.5.2 Capture a Power Trace

With our FPGA bitstream in hand, we are now ready to capture a power trace. A valid key should be loaded on the target, and a key text pair object should be obtained using ChipWhisperer before capture. Using the Xilinx Impact tool, the device is configured once the programmable bit file for the target FPGA is generated. Plaintexts are generated in the host system and supplied via the capture board CWlite board from the PC (Python) to the target FPGA. The corresponding power trace is recorded across the resistor R2 that is put in the core VDD line on the cw305 during encryption. In order to reduce trace misalignment during the power trace acquisition phase, a status signal produced from the target FPGA is employed as a trigger for the capture board. Just prior to the first round of AES, the trigger signal is set. Ten thousand distinct plaintexts are used in the measurements, and the associated 10,000 power traces are recorded. Usually, a basic capture loop consists of the following steps:

1. First, open Python 3.3.7. shell from the taskbar.
2. Click the “file” and click the open in the above Python 3.3.7. shell.

3. Then open the Testcapture_Aes_CWlite.py file using the above Python 3.3.7. shell from the source directory. For example:

```
D:AES_CPA\PythonScripts
```

4. Immediately prior to the first round of AES, the trigger signal is set.
5. Run the python script (Testcapture_Aes_CWlite) as shown below. We have given the comments inside the python script (i.e., how to bit file program into FPGA and how to assign a number of traces). Ktp.next() is to generate the same key and different plaintext for every capture. The Verilog design files and all sources can be found at http://cad4security.org/index.php/trainings/hsl/ch8_psc_on_aes/:

```
# Breaking Hardware AES on CW305 FPGA.

# In[1]:

import chipwhisperer as cw
scope = cw.scope()
scope.gain.db = 25
scope.adc.samples = 129
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "extclk_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "disabled"


# After that, we'll join the CW305 board.
# We must provide the bitstream file to load here.

# Pick the correct bitfile for our CW305 board.
# To set ‘force=True‘ if you have generated a
new bitfile otherwise to set ‘force=False’


# In[2]:

bitstream = r"D:AES_CPA\AES_cw305_top.bit"

target = cw.target(scope, cw.targets.CW305,
bsfile=bitstream, force=True)

# In[3]:

project = cw.create_project(r'D:AES_CPA\
          Tutorial_CW305.cwp', overwrite=True)


# Next we set all the PLLs. We enable CW305's PLL1;
# this clock will feed both the target and the CW ADC.
# make sure the DIP switches on the CW305 board are
set as follows:
# - J16 = 0
# - K16 = 1
```

```
# In[4]:

target.vccint_set(1.0)
# we only need PLL1:
target.pll.pll_enable_set(True)
target.pll.pll_outenable_set(False, 0)
target.pll.pll_outenable_set(True, 1)
target.pll.pll_outenable_set(False, 2)

# target.pll.pll_outsource_set("PLL0", 0)

# run at 10 MHz:
target.pll.pll_outfreq_set(10E6, 1)

# 1ms is plenty of idling time
target.clkusbautooff = True
target.clksleeptime = 1


# In[5]:
# ensure ADC is locked:

tries = 100
for i in range(tries):
    scope.clock.reset_adc()
    if scope.clock.adc_locked:
        break
if not scope.clock.adc_locked:
    print("Couldn’t lock")

# In[6]:

# ## Trace Capture
# The capture loop is shown below. # In the main portion
of the loop, new plaintext is loaded, the scope is
armed, the key and plaintext are sent, and finally, our
new trace is recorded and added to the list of "traces[]".

from tqdm import tnrange
from tqdm.notebook import trange
import numpy as np
import time
from Crypto.Cipher import AES

ktp = cw.ktp.Basic()

traces = []
textin = []
keys = []

# Because we’re capturing 5000 traces

N = 10000  # Number of traces

# initialize cipher to verify DUT result:
key, text = ktp.next()
cipher = AES.new(bytes(key), AES.MODE_ECB)

for i in tnrange(N, desc=’Capturing traces’):
```

```
    key, text = ktp.next()
# manual creation of a key, text pair can be substituted here
    textin.append(text)
    keys.append(key)

    ret = cw.capture_trace(scope, target, text, key)
    if not ret:
        print("Failed capture")
        continue

    traces.append(ret.wave)
    project.traces.append(ret)

# After that, the capture traces must be
saved so that the attack can be repeated
in the future without having to go
through the trace acquisition process again.

# In[7]:

project.save()
# This shows how a captured trace can be plotted:

# In[8]:

import matplotlib.pyplot as plt
plt.figure()
for i in range(N):
    plt.plot(traces[i])
plt.show()

# Finally disconnect the scope and target.

# In[9]:
scope.dis()
target.dis()
```

6. The screen will appear during the capture as shown in Fig. 8.11.

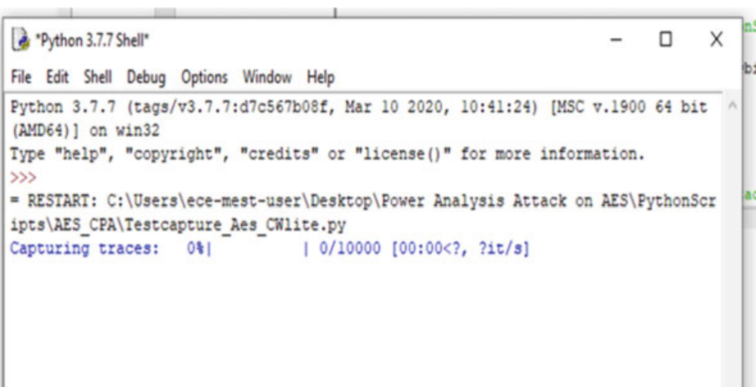


Fig. 8.11 Capturing screen

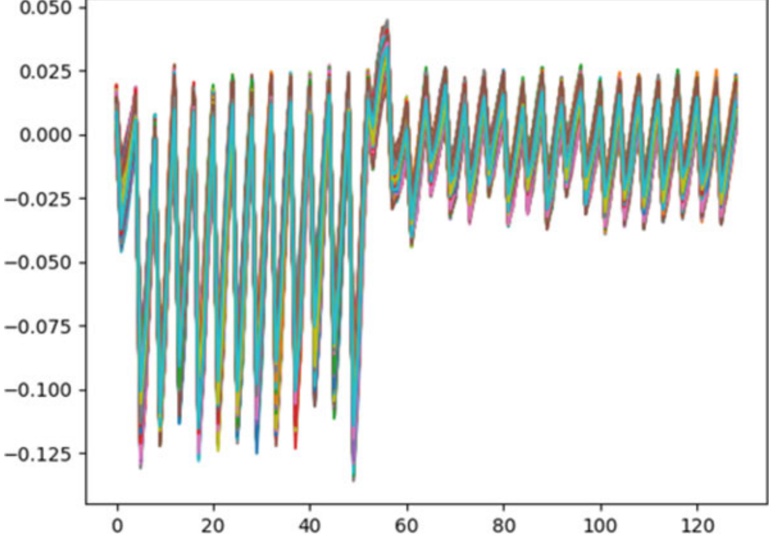


Fig. 8.12 Captured power trace

7. The measurements are repeated over 10,000 different plaintexts, and the corresponding 10,000 power traces are captured. In Fig. 8.12, captured power trace of encryption can be seen.

8.6 Performing AES CPA Attack

Assuming we actually read Sect. 8.2.4, it should be apparent that there are a few things we need to accomplish:

- Reading the data, which consists of the analog waveform (see Fig. 8.12) and input text sent to the encryption core
- Making the power leakage model, where it takes a known input text along with a guess of the key byte implementing the correlation equation 8.3 and then loops through all the traces.
- Ranking the output of the correlation equation to determine the most likely key.

8.6.1 CPA Attack Steps

The CPA attack flow is shown in Fig. 8.13. For this hardware AES implementation, we use a hamming distance leakage model. Keep in mind that CPA is used for the final round of AES encryption. The final round of AES encryption is subjected to

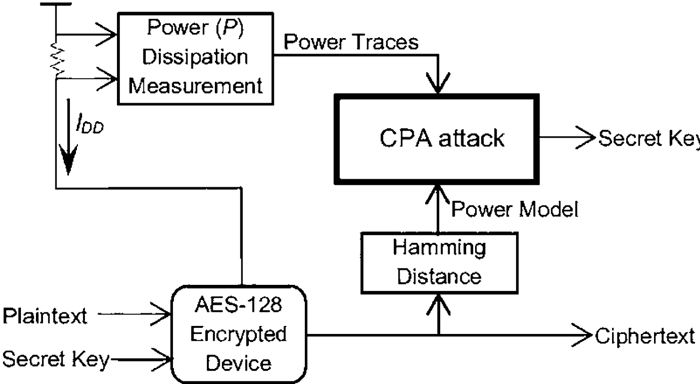


Fig. 8.13 CPA attack flow

Table 8.1 Attack keys

Secret key	Final round attack key
2B 7E 15 16 28 AE D2 A6	D0 14 F9 A8 C9 EE 25 89
AB F7 15 88 09 CF 4F 3C	E1 3F 0C C8 B6 63 0C A6

CPA. In order to obtain the tenth round key bytes at a time using CPA, ciphertext is assumed to be a known input, and the tenth round key is assumed to be an unknown input. One advantage of attacking the final round in AES is that the MixColumns step is bypassed in this round, which speeds up calculation for the attack. Performing the inverse ShiftRows and the inverse SubBytes operations on the selected byte and guessing a byte of the key, one receives 256 possible values, which were in the State Register in the previous round. Computing the Hamming distance between the value that was in the register after the last round (the cipher text) and the guessed 256 values from the previous round, we obtain the expected power consumption model based on the transitions made on the State Register. The round key computing process is reversible; therefore, the round key can be used to compute the original key. Table 8.1 lists the first secret keys and their equivalent final round attack keys.

- The first step of CPA is to compute a hypothetical intermediate value, which is hypothetical ninth round output. Equation 8.1 illustrates the operation to be followed in order to calculate the hypothetical ninth round output ($9^{th} roundOutput_{hyp}$):

$$9^{th} roundOutput_{hyp} = Sbox^{-1}(Shiftrows^{-1}(ciphertext \oplus K_{guess})) \quad (8.1)$$

- The second step is to translate the hypothetical output of the ninth round to the hypothetical power consumption, as indicated in Eq. 8.2. Hypothetical power consumption is computed by taking the hamming distance between the $9^{th} roundOutput_hyp$ with the corresponding ciphertext byte:

$$X = HD(ciphertext; 9^{th} roundOutput_hyp) \quad (8.2)$$

- The final stage in determining the proper tenth round key is to compare these hypothetical power consumption statistics to actual power traces [3]. The correlation between the estimated power consumption values (X) and the measured power traces (Y) is calculated using Eq. (8.3). In this equation, E stands for expectation; μ_X and μ_Y are the mean values of X and Y, respectively; and σ_X and σ_Y are the standard deviations of X and Y, respectively:

$$Correlation - coefficient\ (X, Y) = \frac{E[(Y - \mu_Y)(X - \mu_X)]}{\sigma_Y * \sigma_X} \quad (8.3)$$

Once we have our capture data, the analysis is straightforward: a standard CPA attack is easy to do with another python script, AES_Attack.py, from the directory D:AES_CPA:

```
# We now open our previously saved project again and
specify the attack parameters for the CPA assault. We
employ a different leaking model and attack for this
hardware AES implementation than we do for software AES
implementations.

# Only the ciphertext is needed for this attack; the
plaintext is not necessary.

# In[1]:

import chipwhisperer as cw
import chipwhisperer.analyzer as cwa
import numpy as np
from tqdm.notebook import trange
import matplotlib.pyplot as plt

project_file = r'D:AES_CPA\Tutorial_CW305.cwp'
project = cw.open_project(project_file)

attack = cwa.cpa(project,
cwa.leakage_models.last_round_state_diff)

# In[2]:

# This runs the attack:

attack_results = attack.run()

# Without having to repeat the attack, the attack results
can be saved for subsequent inspection or processing:

# In[3]:
```

```
#import pickle
#pickle_file = project_file + ".results.pickle"
#pickle.dump(attack_results, open(pickle_file, "wb"))

# In[4]:
# key_guess

attack_results.key_guess()

#print(bytearray(project[0].keys[0]))
print(attack_results.find_maximums()[4][0][2])
print(attack_results)

# In[5]:


plot_data = cwa.analyzer_plots(attack_results)

#plt.figure()
#plt.plot(plot_data)
#plt.show()
# The correlation between the best guess and the next
best guess now shows a significant difference. In fact,
we were able to extract the key from the previous AES
cycle. To obtain the true AES key, we must utilize CW
analyzer:

from chipwhisperer.analyzer.attacks.models.aes.key_schedule
import key_schedule_rounds
recv_lastroundkey = [kguess[0][0] for kguess in
attack_results.find_maximums()]
print(recv_lastroundkey)
recv_key = key_schedule_rounds(recv_lastroundkey, 10, 0)
print(recv_key)
for subkey in recv_key:
    print(hex(subkey))

# ## Tests
#Verify that the key that was used is the key that was
obtained during the attack. In order to compare it to the
key we provided, we must roll it back because this attack
targets the last round key.


ktp = cw.ktp.Basic()
key, text = ktp.next()

key = list(key)
assert (key == recv_key), "Failed to recover encryption
key\nGot: {}\nExpected: {}".format(recv_key, key)
```

```
Python 3.7.7 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\ece-mest-user\Desktop\Power Analysis Attack on AES\PythonScripts\AES_CPA\Testcapture_Aes_CWlite.py
Capturing traces:   0%|          | 0/10000 [00:00<?, ?it/s]
>>>
= RESTART: C:\Users\ece-mest-user\Desktop\Power Analysis Attack on AES\PythonScripts\AES_CPA\AES_Attack.py
0.20090454798316584
Subkey KGuess Correlation
  00    0xD0    0.18784
  01    0x14    0.17389
  02    0xF9    0.18451
  03    0xA8    0.18222
  04    0xC9    0.20090
  05    0xEE    0.18353
  06    0x25    0.22126
  07    0x89    0.19653
  08    0xE1    0.17343
  09    0x3F    0.17490
  10    0x0C    0.20258
  11    0xC8    0.15160
  12    0xB6    0.16558
  13    0x63    0.17688
  14    0x0C    0.19082
  15    0xA6    0.20086
```

Fig. 8.14 Correlation values and all bytes of the last round key of AES-128

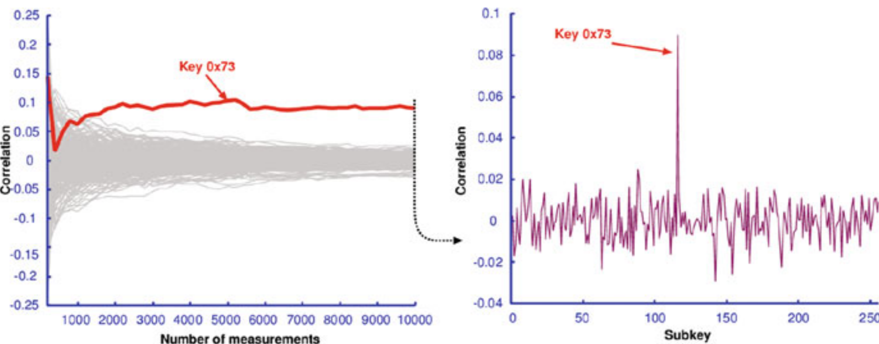


Fig. 8.15 Results of 1-byte attack on AES-128 using CPA

As shown in Fig. 8.14, the correlation values successfully recovered all bytes of the last round key of AES-128 by mounting a CPA attack using 10,000 traces. Figure 8.15 on the left shows the correlation of all $K = 256$ subkey permutations to the measurement results of another key pair as a function of the number of measured samples. On the right, the correlation of all $K = 256$ subkey permutations is given for 10,000 traces.

8.7 Conclusion

This chapter aimed to perform attacks on the AES-128 algorithm using correlation power analysis techniques. The theoretical operation of power analysis attacks has been illustrated in this chapter, and subsequently it has been applied to an FPGA

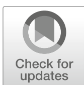
version of an AES core. By executing a CPA attack with 10,000 traces, we were able to successfully extract every byte of the final round key of AES-128. We hope the methodology demonstrated in this article provides a meaningful stepping stone in achieving such attacks, while results presented in CPA prove useful for researchers who wished to learn more about the theory and practical aspects of power analysis attacks.

References

1. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms. Preprint arXiv:2204.07909 (2022)
2. Anandakumar, N.N.: SCA Resistance analysis on FPGA implementations of sponge based MAC PHOTON. In: Innovative Security Solutions for Information Technology and Communications, pp. 69–86. Springer, Cham (2015)
3. Anandakumar, N.N., Dillibabu, S.: Correlation power analysis attack of AES on FPGA using customized communication protocol. In: Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology, CCSEIT '12, pp. 683–688 (2012)
4. Anandakumar, N.N., Das, M.P.L., Sanadhya, S.K., Hashmi, M.S.: Reconfigurable hardware architecture for authenticated key agreement protocol over binary edwards curve. ACM Trans. Reconfigurable Technol. Syst. **11**(2), 1–19 (2018)
5. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Design and analysis of FPGA based PUFs with enhanced performance for hardware-oriented security. ACM J. Emerg. Technol. Comput. Syst. **18**, 1–26 (2022)
6. Anandakumar, N.N., Sanadhya, S.K., Hashmi, M.S.: Design, implementation and analysis of efficient hardware-based security primitives. In: 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), pp. 198–199. IEEE (2020)
7. Bhunia, S., Tehranipoor, M.: Hardware security: a hands-on learning approach. Morgan Kaufmann, Elsevier (2018)
8. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11–13, 2004. Proceedings, Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004)
9. Dworkin, M.J., Barker, E.B., Nechvatal, J.R., Foti, J., Bassham, L.E., Roback, E., Dray, J.F.: Advanced Encryption Standard (AES). FIPS PUB 197 (2001). https://www.nist.gov/publications/advanced-encryption-standard-aes
10. Inc., N.T.: Chipwhisperer-lite (cw1173) two-part version. https://store.newae.com/chipwhisperer-lite-cw1173-two-part-version/
11. Inc., N.T.: Cw1200 chipwhisperer-pro. https://rtfm.newae.com/Capture/ChipWhisperer-Pro/
12. Inc., N.T.: Cw305 artix fpga target. https://www.newae.com/products/NAE-CW305
13. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011)
14. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology—CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15–19, 1999, Proceedings, Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25. URL https://doi.org/10.1007/3-540-48405-1_25
15. Lo, O., Buchanan, W.J., Carson, D.: Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA). J. Cyber Secur. Technol. **1**(2), 88–107 (2017)

16. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks—Revealing the Secrets of Smart Cards. Springer, Berlin (2007)
17. Mano, M.M., Kime, C.R., Martin, T.: Logic and Computer Design Fundamentals. Prentice-Hall, Englewood (2000)
18. Mui, E.N., Custom: Practical Implementation of Rijndael S-Box Using Combinational Logic (2007). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.180.3242&rep=rep1&type=pdf
19. Nahiyan, A., Park, J., He, M., Iskander, Y., Farahmandi, F., Forte, D., Tehranipoor, M.: SCRIPT: a CAD framework for power side-channel vulnerability assessment using information flow tracking and pattern generation. ACM Trans. Des. Autom. Electron. Syst. **25**(3), 1–27 (2020)
20. Ng, J.S., Chen, J., Kyaw, N.A., Lwin, N.K.Z., Ho, W.G., Chong, K.S., Gwee, B.H.: A highly efficient power model for Correlation Power Analysis (CPA) of pipelined Advanced Encryption Standard (AES). In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020). https://doi.org/10.1109/ISCAS45731.2020.9180778
21. Örs, S.B., Oswald, E., Preneel, B.: Power-analysis attacks on an FPGA —first experimental results. In: Cryptographic Hardware and Embedded Systems—CHES 2003, 5th International Workshop, Cologne, Germany, September 8–10, 2003, Proceedings, Lecture Notes in Computer Science, vol. 2779, pp. 35–50. Springer (2003)
22. Park, J., Anandakumar, N.N., Saha, D., Mehta, D., Pundir, N., Rahman, F., Farahmandi, F., Tehranipoor, M.M.: PQC-SEP: power side-channel evaluation platform for post-quantum cryptography algorithms.In: IACR Cryptology ePrint Archive, p. 527 (2022)
23. Pundir, N., Park, J., Farahmandi, F., Tehranipoor, M.: Power side-channel leakage assessment framework at register-transfer level. IEEE Trans. Very Large Scale Integr. VLSI Syst. **30**, 1207–1218 (2022)
24. Randolph, M., Diehl, W.: Power side-channel attack analysis: a review of 20 years of study for the layman. Cryptography **4**(2), 15 (2020)
25. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer Science & Business Media, Berlin (2011)
26. Zhang, T., Park, J., Tehranipoor, M., Farahmandi, F.: PSC-TG: RTL power side-channel leakage assessment with test pattern generation. In: 2021 58th ACM/IEEE Design Automation Conference (DAC), pp. 709–714. IEEE (2021)

Chapter 9
EM Side-Channel Attack on AES



9.1 Introduction

Side-channel information describes quantifiable physical manifestations connected to multiple ICs (integrated circuits) or unrelated computer system processes. Researchers have employed a variety of side-channel parameters, such as temporal variation and power consumption, leakage current, electromagnetic (EM) radiation, temperature, sound, light, and infrared radiation. Since its inception several side-channel analysis (SCA) attacks have been indicated such as power monitoring attacks, timing, cache attacks, audio-based attacks, electromagnetic (EM) emission attacks, data remanence, etc [2, 5–7]. These attacks exploit vulnerabilities in the ICs of the Internet of Things (IoT) devices, smart cards, mobile devices, and computers to infer sensitive information. Compared to other side-channel attacks, EM-based side-channel attack has significant advantages, including non-contact measurement, location awareness, and high-frequency information. EM radiation can generally be divided into two types: direct radiation and modulated radiation [18]. Direct radiation is directly caused by current flow with sharp rising/falling edges, while modulated radiation occurs when a signal carrier modulates the signals, creating outwardly propagating EM radiation. EM radiation includes, but is not limited to, radio and microwaves, visible light, infrared, X-rays, ultraviolet, and gamma rays. In ICs, currents and charges emit near-field EM radiation, which is described by Maxwell's equations. Currents generate magnetic fields, fluctuating currents generate electric fields, and the combined magnetic and electric fields produce near-field electromagnetic radiation (EM radiation). When executing logic operations, ICs produce electromagnetic (EM) radiation, which reflects the operational conditions of the ICs and can be used for side-channel analysis (SCA) attacks [22, 23].

The electromagnetic analysis (EMA) attacks, which are side-channel attacks that take advantage of a device's electromagnetic radiation, were reported by Grandolfi et al. in 2001 [13]. Similar to correlation power analysis (CPA) attacks (see chapter reference Chap. 8), correlation electromagnetic analysis (CEMA) attacks determine

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_9

correlation coefficients between electromagnetic traces and intermediate values (i.e., a hypothetical model). The leaked model of these attacks is divided into two primary models [19]. One is the Hamming weight (HW) model against the current state of the circuit (target), which is the input/output values for the nonlinear logic gate such as the OR/AND [20]. The other is the Hamming distance (HD) model against the number of bit transitions between the circuit's present state (target) and its previous state. Typically, there are two stages to an EM attack. The attacker gathers the EM emanations during the first stage of the attack using an EM probe that may be coupled to a low-noise amplifier (LNA) positioned close to the target device. The target device's secret key is extracted in the second phase using simple or differential EM analysis [10]. In this chapter, we focus on correlation EM analysis (CEMA) attacks on advanced encryption standard (AES) crypto hardware using the best suitable EM probe. In particular, this chapter can help a reader to better understand measurement equipment, the attacked design, and the capturing EM traces and will gain hand experience on how to extract secret keys from AES block cipher using CEMA attacks on FPGA step by step.

The remaining chapters are structured as follows. Section 9.2 summarizes the fundamentals of electromagnetic radiation in terms of side-channel analysis. Section 9.3 presents the implementation details of investigated AES design. Section 9.4 presents the device under attack, and the measurement setup, including all investigated probes followed by the EM measurements on the FPGA-based AES design in Sect. 9.5. Section 9.6 shows the correlation EM analysis (CEMA) attack on AES. The chapter finishes with short conclusions in Sect. 9.7.

9.2 Background

9.2.1 Measuring EM Radiation

The creation of ICs frequently utilizes CMOS technologies. In the present, semiconductor foundries accept numerous metal layers to create circuits, use interconnects to create electrical connections, and use vias to connect various layers. ICs receive electricity from an external source through their power pins. Each transistor will then get electricity from the on-chip power grid. For an integrated CMOS-based circuit, no static current flows through the circuit at a steady state. However, based on Gauss's law ($\nabla \cdot \vec{E} = \frac{\rho}{\epsilon}$), electric fields ($\vec{E}$) are generated due to the existence of stationary charges in the circuit. As the output of logic gates changes its state, moving charges (leakage and dynamic currents) cause changing electric fields, which in turn produce changing magnetic fields $\nabla \times \vec{H} = \vec{J} + \epsilon \frac{\partial \vec{E}}{\partial t}$ (known as modified Ampere's law). On the other hand, according to Faraday's law and the magnetic permeability definition $\left(\mu_0 = \vec{B}/\vec{H}\right)$, changing currents produce time-changing magnetic flux, thereby inducing an electric field ($\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t}$) [14].

Note that $\vec{H}$ represents the strength of the magnetic field, $\vec{E}$ is the electric field, J is the current density of electricity, $\vec{B}$ is the density of magnetic flux, mu is the permeability of the magnetic field, rho is the density of electricity, and is $epsilon$ the permittivity of electricity.

SCA attacks basically assume that the current through a cryptographic chip depends on the processed inputs and on the user's secret key. Thus, the changes in the current through a wire cause changes in its magnetic field. The magnetic field of the current also depends on the processed inputs and secret key and therefore the attacker can pick up the radiated side channel (EM emissions) and extract the secret key from the cryptographic chip using an electromagnetic analysis attack. The rate of magnetic field changes will be measured using probes. State-of-the-art near-field probes are used for measurements of both magnetic (H) and electric (E) fields in IC assemblies and devices [14, 24]. Electric field probes are used to collect electric-field radiation on individual IC pins, interconnect buses, distribution areas, and clock lines connecting the electric field with their surfaces. On the other hand, conducting loops are used in H-field probes to measure magnetic fields produced by changing currents flowing in conductors, power supplies, capacitors, and clock/control signals. The tester must first decide whether the signal is an H-field or an E-field in order to select the appropriate probe. Because the H-field probe design typically allows for the suppression of E-field effects and vice versa, H-field probes do not perform well in detecting the H-field signal for practical EM radiation collection. The power and frequency range of the test signal must be determined by the tester. The collected signal may optionally be amplified using pre-amplifiers [14, 24].

9.2.2 Typical EM Side-Channel Attacks

EM analysis can be used to obtain secret information by analyzing the electric and/or magnetic fields emitted from a cryptographic device, which include simple EMA (SEMA) and differential EMA (DEMA). As one of the powerful methods of side-channel attacks (SCAs), since Quisquater and Samyde [21] introduced it in 2001, a lot of work has been done, and much literature reports successful attacks in implementing symmetric and asymmetric encryption schemes [1, 8, 13, 16, 17, 24]. SEMA targets information from the sensitive computation that can be recovered from a single or a few traces. DEMA removes noise with an attack statistics technique and is therefore more powerful than SEMA. A standard DEMA attack typically has two phases: signal capture and analysis. Similar to differential power analysis (DPA), sufficient time-domain samples must be obtained during the signal capturing phase, and during the signal analysis phase, confidential information is retrieved using different statistical methods, two of which are the most popular methods: difference and Pearson correlation coefficient [3]. DEMA with correlation coefficient method is also known as CEMA. Compared to a different method using

a binary model (i.e., strong EM signals are modeled as 1 and weak EM signals as 0), CEMA exploits the linear relationship between EM signals and processed data. The nonbinary model used in CEMA is more accurate for modeling EM signals and less computationally complex, making it more widely used than DEMA. Until now, almost all correlation analysis attacks have focused on time-domain signals. A necessary condition of time domain CEMA is that the signals must be precisely aligned.

9.3 Implementation Details of Investigated AES Design

AES is a symmetric block cipher with key sizes of 128, 192, and 256 bits that is defined for 128-bit blocks. The number of rounds is varied to 10, 12, or 14 depending on the key size. We choose the AES-128 algorithm for our experiments here. The AES-128 algorithm is an iterative algorithm composed of ten rounds. After the initial secret key addition, the first nine rounds are identical, but the final round is different [11, 25]. Each of the first nine rounds consists of four state operations: AddRoundKey, SubBytes, ShiftRows, and MixColumns. The final round excludes the Mix-Columns operation. The SubBytes transformation is a nonlinear byte substitution that operates independently on each byte of the state using a substitution table (S-box). The ShiftRows perform a cyclic rotation on each row of the state. The MixColumns linearly combines the elements in each column. It can be represented as a multiplication of each column with a constant matrix. AddRoundKey ties the result to the key by XORing each element with an element from the current round key. The AES design consists of an AES controller, two 128-bit registers, and AES encryption module. The two registers store the data and key values. This AES encryption module performs the encryption operation on 128 bit of data using the AES algorithm that takes 10 clock cycles. The iterative architecture of AES encryption is shown in Fig. 9.1. The AES design is developed using Xilinx Vivado 2020.2 and coded in Verilog HDL, whereas Python is utilized for communication between the FPGA board and the PC using the USB interface. We have used the Xilinx Vivado software for AES bitstream generation. How to download and install the Xilinx Vivado Design Suite is discussed in Chap. 8. For this tutorial, we will provide pre-existing AES-128 encryption. There are three steps that Vivado takes to turn the Verilog code into a bitstream file: (1) synthesis in which the Verilog code is synthesized into a gate-level representation, (2) implementation in which the synthesized logic is routed to fit onto the device, and (3) finally, we generate the bitstream. Once the bitstream is generated, we need to identify its path. It is usually stored in the project file folder.

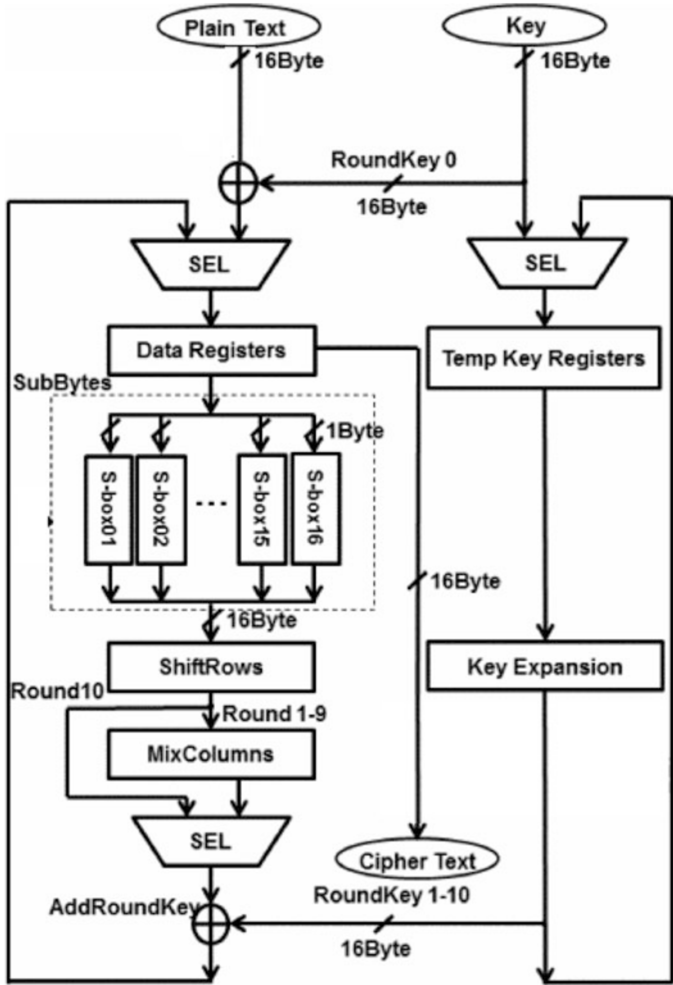


Fig. 9.1 Iterative architecture of AES encryption

9.4 Measurement Setup

In this chapter, we have used the device under attack which is the CW305 target board [15] that has a Xilinx Spartan-6 FPGA and ensure it is correctly set up for EM trace capture. We captured the EM traces using a Tektronix MSO 70404C Mixed Signal Oscilloscope with a sampling rate of 3.13 GS/s and a time scale of 160 ns/div. This results in 313 measurement points per clock cycle at 10 MHz clock frequency. The distance between the EM probe and the surface of the design under test (DUT) is as small as possible for the commercial probes. Generally, a power SCA attack typically requires the insertion of a small resistor ($\sim 0.5 - 10\Omega$) in series

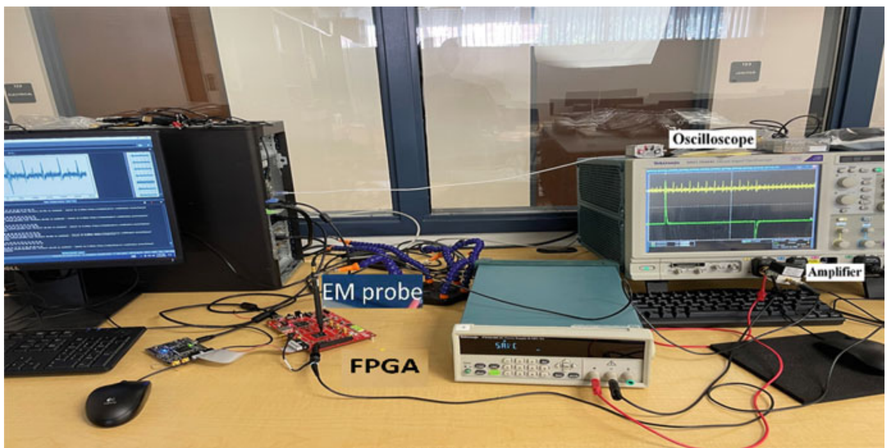


Fig. 9.2 Measurement setup: FPGA, EM probe, oscilloscope, power supply

with the power supply of the measurement device that measures the voltage drop across it. On the other hand, EM SCA attacks are noninvasive and do not require any modification of the device under attack. With the improvement in sensitivity of commercially available EM probes, EM attacks are becoming more powerful [9]. We decided to use a near-filed Langer RF-K7-4 probe with an extra amplifier [12]. Electromagnetic traces were measured by a horizontal magnetic field probe placed on the chip surface. The whole measurement setup is shown in Fig. 9.2. We use AES-128 as our encryption algorithm and the chip runs AES encryption periodically with a fixed 128-bit key and random plaintexts. The AES design is developed using Xilinx Vivado 2020.2 and coded in Verilog HDL, whereas Python is utilized for communication between the Artix-7 FPGA board and the personal computer using the USB interface. The ChipWhisperer software includes a Python API for talking to ChipWhisperer hardware and also a Python API for processing EM traces from ChipWhisperer hardware. We can define the oscilloscope setup using the following Python script:

```
def instrument_setup(dutAddr, start, end):
try:
rm = visa.ResourceManager()
scope = rm.open_resource(dutAddr)
        # Change to exact instrument ID
print(scope.query("*IDN?"))
except Exception as e:
print("Error creating instance: {0}".format(e))
sys.exit()
# Horizontal Setup
scope.write('HORizontal:DELay:MODe OFF')
scope.write('HORizontal:POSistion 10')
```

```
scope.write('HORizontal:SCAle 400e-9')
scope.write('HORizontal:RECOrdlength 10e3')
# Trigger Setup
scope.write('TRIGer:A:EDGE:COUPling DC')
scope.write('TRIGer:A:EDGE:SLOpe rise')
scope.write('TRIGer:A:EDGE:SOUrce CH2')
scope.write('TRIGer:A:LEVel:CH2 1.5')
scope.write('TRIGger:A:HOLDoff:TIMe 1.00E-03')
#Vertical Setup
scope.write('CH1:BANDWIDTH 250E6')
scope.write('CH2:BANDWIDTH 250E6')
scope.write('CH1:COUPling AC')
scope.write('CH2:COUPling DC')
scope.write('CH1:INVert OFF')
scope.write('CH2:INVert OFF')
scope.write('CH2:SCALE 1')
scope.write('CH1:SCALE 5E-03')
scope.write(':DATa:SOUrce CH1')
scope.write(":DATa:START " + start)
scope.write(":DATa:STOP " + end)
scope.write(':DATa:ENCdg ASCIi')
scope.write(':DATa:WIDth 1')
nr_pt = int(scope.query(':WFMOutpre:NR_Pt?'))
#yunit = scope.query(':WFMOutpre:YUNit?')
yoff = float(scope.query(':WFMOutpre:YOFF?'))
yzero = float(scope.query(':WFMOutpre:YZERO?'))
ymult = float(scope.query(':WFMOutpre:YMUlt?'))
xincr = float(scope.query(':WFMOutpre:XINcr?'))
xdely = float(scope.query(':HORizontal:POSition?'))
print('----- Instrument Connected! -----')
print('Number of Points: %d' % nr_pt)
print('Y zero: %f' % yzero)
print('Y off: %f' % yoff)
print('Y mult: %f' % ymult)
print('X incr: %f' % xincr)
print('X delay: %f' % xdely)
scope.timeout = 10000
scope.chunk_size = 20480
#scope.write('*CLS')
scope.write('TRIGger:A:HOLDoff:TIMe?')
print(scope.read())
return scope, yoff, ymult, yzero, xincr, nr_pt
```

We'll then establish a connection with the CW305 board. Here, we must specify the bitstream file that we want to load.

9.5 EM Measurements on the AES Chip

9.5.1 Tool Setup

To get started, we need to install Anaconda, Spyder, and ChipWhisperer tools (see Fig. 9.3) based on the following steps:

1. First download and install the Anaconda 2.1.2 tool from the following URL:

   `https://docs.anaconda.com/anaconda/install/windows/`

2. After a successful Anaconda installation, please create a new environment using Anaconda Navigator.
3. Then, install and launch the Spyder from the Anaconda navigator.
4. Next, to install the ChipWhisperer tool, we need to type and enter the command on the console window of the Anaconda navigator.

   ```
   pip install chipwhisperer
   ```

5. After installing the ChipWhisperer tool, we need to install PyVISA and matplotlib packages using the following commands:

   ```
   pip install -U pyvisa
   pip install matplotlib
   ```

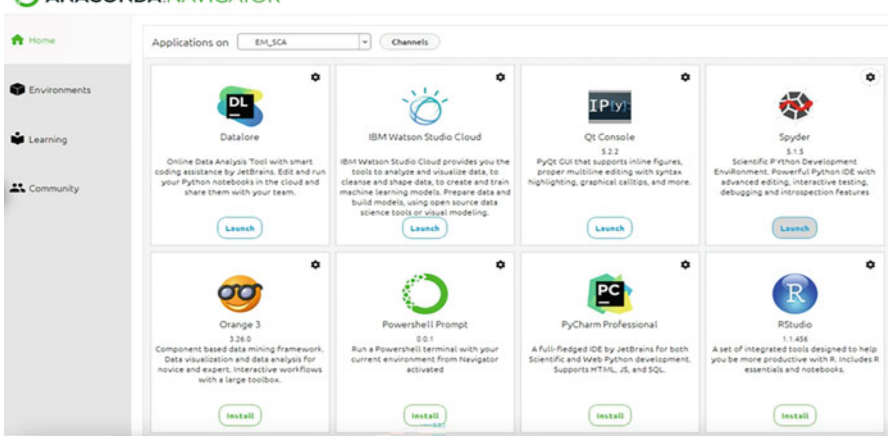


Fig. 9.3 Anaconda and Spyder tool setup

9.5.2 Capture an EM Trace

To get started, we need to make three connections to the CW305 target:

1. We'll control the Artix-7 directly from the computer via USB. This is done through the USB-B port on the left side of the board. (On the target, the Atmel SAM3U chip converts these USB packets into signals that the FPGA can understand.)
2. We'll get control information for our EM measurements through the USB at the oscilloscope.
3. Our EM measurements will be done through an EM probe.
   - The Python program responsible for collecting the traces, which runs on a normal PC, starts the capture process on the chip.
   - The code triggers the encryption of a chosen/random key and plaintext on the targeted hardware.
   - At the end of the encryption, the program stops the capture and collects the EM traces. The trace and its label (the key and plaintext used) are added to the attack dataset.

With our FPGA bitstream in hand (see Sect. 9.3), we are now ready to capture an EM trace. Once the programmable bit file is generated for the target FPGA, the device is configured using the Xilinx Impact tool. Before capture, we ensure a bitstream is loaded on the FPGA target. In the host system, plaintexts are generated and fed from the PC (Python) to target FPGA via the USB interface. During the encryption, the corresponding EM trace is captured using the near-filed Langer RF-K7-4 probe and optionally connected to a low-noise amplifier (LNA) and is recorded in an oscilloscope. During the EM trace acquisition period, a status signal output from the target FPGA is used as a trigger for the target board that minimizes trace misalignment. The trigger signal is set just before the initial round of AES. The measurements are repeated over 10,000 different plaintexts for the same secret key, and the corresponding 10,000 EM traces are captured. Our Python script is responsible for configuring the bit file, arming boards, and capturing and storing the EM traces (T). The project folders and files are shown in Fig. 9.4. At the end of the encryption, the program stops the capture and collects the EM trace out of the oscilloscope and stores it in the PC using the ethernet cable.

Typically, a basic capture loop consists of the following steps:

1. Set Environments (name as) EM_SCA in Anaconda navigator.
2. Click the EM_SCA and then open Spyder in the Anaconda navigator.
3. Open the Python script GetTraces.py and set some variables (dutAddr, key, N_traces, and plt_on) to any values in GetTraces.py as shown in Fig. 9.5.
4. We also provide the Python code for the design as given below. Run the GetTraces.py script. Moreover, all Verilog design files and all python sources can be found at http://cad4security.org/index.php/trainings/hsl/ch9_em_sca/.

```
/EM_SCA
     |----- /AES_128                    // AES_128 Implementation for ChipWhisperer FPGA
     |----- /SW                         // Capture and CPA program based on Python
             |------ /aes128_EM2_data   // For storing  EM traces, plaintexts and ciphertexts (10,000)
             |------ GetTraces.py       // For collecting EM traces
             |------ CEMA.py              // For performing CEMA attacks
             |------ AES128.py.         // AES128 API
             |----- aes128_register.py  // For defining the address of aes128 regs
             |------ aes128.cwp.        // ChipWhisperer project file
             |------ aes128_EM2.cwp     // ChipWhisperer project file
```

Fig. 9.4 EM SCA folders and files

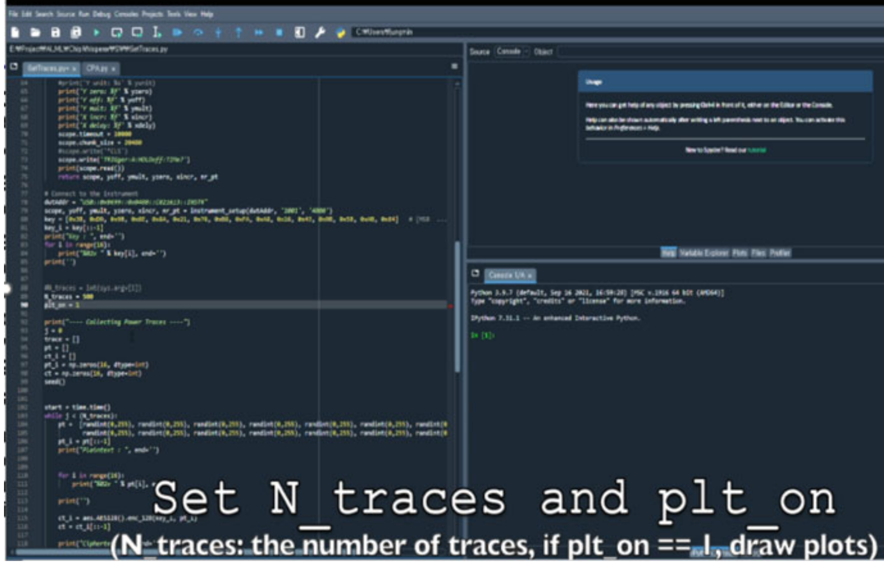


Fig. 9.5 Set *N_traces*

```
import pyvisa as visa
import AES128 as aes
import time
import matplotlib.pyplot as plt
import numpy as np
from random import seed
from random import randint
import os
import sys
import chipwhisperer as cw

#N_traces = int(sys.argv[1])  # Number of traces
```

```
N_traces = 10000
plt_on = 1

print("---- Collecting Power Traces ----")
j = 0
trace = []
pt = []
ct_i = []
pt_i = np.zeros(16, dtype=int)
ct = np.zeros(16, dtype=int)
seed()


start = time.time()
while j < (N_traces):
pt = [randint(0,255), randint(0,255), randint(0,255),
      randint(0,255), randint(0,255), randint(0,255),
      randint(0,255), randint(0,255), randint(0,255),
      randint(0,255), randint(0,255), randint(0,255),
      randint(0,255), randint(0,255), randint(0,255),
      randint(0,255)]
pt_i = pt[::-1]
print("Plaintext : ", end='')


for i in range(16):
print("%02x " % pt[i], end='')

print('')

ct_i = aes.AES128().enc_128(key_i, pt_i)
ct = ct_i[::-1]

print("Ciphertext : ", end='')
for i in range(16):
print("%02x " % ct[i], end='')

print('')

try:
values = np.array(scope.query_ascii_values('CURVe?'))
except:
pass
scope.clear()

if j == 0:
traces = values
pts = pt
cts = ct
else:
traces = np.concatenate((traces, values))
pts = np.concatenate((pts, pt))
cts = np.concatenate((cts, ct))
```

```
if (plt_on):
Volts = (values - yoff) * ymult + yzero
#Time = np.arange(0, xincr * len(Volts), xincr)
plt.plot(range(len(Volts)), Volts)
plt.pause(0.01)
if j != N_traces-1:
plt.clf()

j += 1
print("Trace %d" % j)

tr_arry = traces.reshape(N_traces, nr_pt)
pt_arry = pts.reshape(N_traces, 16)
ct_arry = cts.reshape(N_traces, 16)
pt_arry = np.array(pt_arry, dtype = np.uint8)
ct_arry = np.array(ct_arry, dtype = np.uint8)

isExist = os.path.exists('./Data')

if not isExist:
os.makedirs('./Data')

end = time.time()
print("Elasped time: %d seconds" % (end - start))
print("Done!")
np.savetxt('./Data/EM/traces.csv', tr_arry, delimiter = ',')
np.savetxt('./Data/EM/pts.csv', pt_arry, delimiter = ',')
np.savetxt('./Data/EM/cts.csv', ct_arry, delimiter = ',')

# Create a project file and Store data
key = np.array(key)
if os.path.exists('aes128_EM1.cwp'):
os.remove('aes128_EM1.cwp')

proj = cw.create_project("aes128_EM1")
for i in range(N_traces):
trace = cw.Trace(tr_arry[i,:], pt_arry[i,:],
        ct_arry[i,:], key)
proj.traces.append(trace)
proj.save()
scope.close()
```

5. The measurements are repeated 10,000 times with 10,000 distinct plaintexts, and 10,000 EM traces are recorded for each measurement. A captured EM trace of encryption is visible in Fig. 9.6.
6. Finally, $aes128_EM2.cwp$ file is generated.

Fig. 9.6 EM traces capturing screen

9.6 Performing Correlation Electromagnetic Analysis (CEMA) Attack

The CEMA is carried out in this section to confirm the efficiency of EMA on our FPGA-based implementation of AES-128. The CEMA attack flow is shown in Fig. 9.7. As stated in Sect. 9.2, EM emissions directly contribute to the CMOS device's energy consumption, and there is a correlation between the peaks of EM signals and the data that the device processes. In order to build an EM leakage model that includes the anticipated EM leakage of a device performing a specific operation during encryption (like the S-box operation in the AES), over the provided plaintexts with all possible key bytes, we can take into account electromagnetic radiation (EM radiation) and use hamming distance. For each byte of the secret key, this reduces the key search space of the AES 128 to $2^8 = 256$ possibilities. The correct tenth-round key is then found by computing the correlation coefficient between the EM hypothesis (H) and the obtained traces (T) over time. For more details, please refer to [16] for correlation analysis and its measurement costs. Equation (9.1) [4] is used to compute the correlation between the hypothetical EM emission values (H) and actual EM emission (T). In this equation, E is the expectation, μ_H and μ_T are the mean values of H and T, and σ_H and σ_T are the standard deviations of H and T, respectively.

$$Correlation - coefficient\ (H, T) = \frac{E[(T - \mu_T)(H - \mu_H)]}{\sigma_T * \sigma_H} \tag{9.1}$$

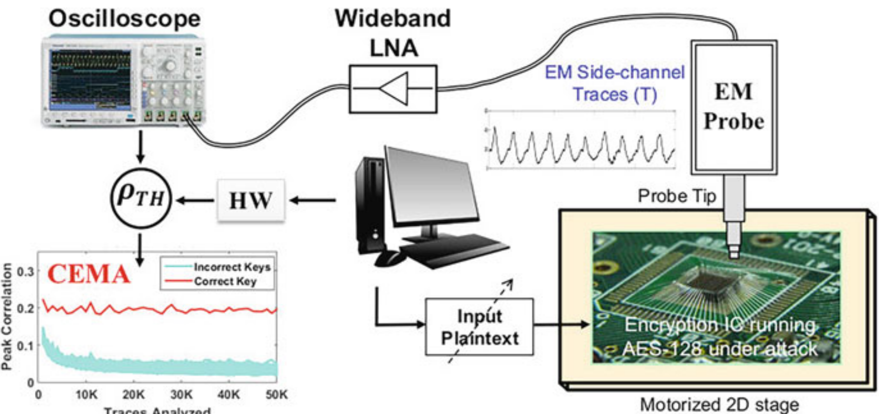


Fig. 9.7 CEMA attack flow [8]

In order to test the effectiveness of an EM-based side-channel attack, we also applied CEMA to the final subkey of the AES key along with the EM traces that were captured. The identical set of plaintexts and tenth key (AF 6F 83 AE B2 D6 A2 82 0A E2 95 FD 85 05 64 5D) were utilized in the EM traces gathered for the CEMA attack. Therefore, the ciphertext is taken as a known input and the unknown input is the tenth round key, which is to be retrieved byte by byte using CEMA. Due to the reversible nature of the round key computation algorithm, the original key (3B D9 9B 8E 8A 21 76 B8 FA A8 16 43 9B 5B AB 84) can be computed from the round key. The EM-based side-channel attack is more threatening because it is a non-contact attack compared with EM analysis attacks.

The following distinctions separate the CEMA attack from the CPA attack:

- The CEMA attack makes use of a specialized EM probe as opposed to the passive probe utilized in the CPA attack.
- The amplification factor in EM capture is larger than in power collection since the technique is non-contact and the EM radiation is more vulnerable to external noise.
- The fact that CEMA does not require accurate information about the time instance at which the targeted operation occurs is a key advantage.

Once we have our capture data, the analysis is straightforward: a standard CEMA like a CPA attack (see Chap. 8) is easy to do with another Python script: CEMA.py from the directory D:EM_SCA. The CEMA attack are shown in Fig. 9.8.

- Open CEMA.py
- Set project name, leakage model, and the range of points to suitable values in CEMA.py.
- Run CEMA.py
- Get some results (Guess keys)

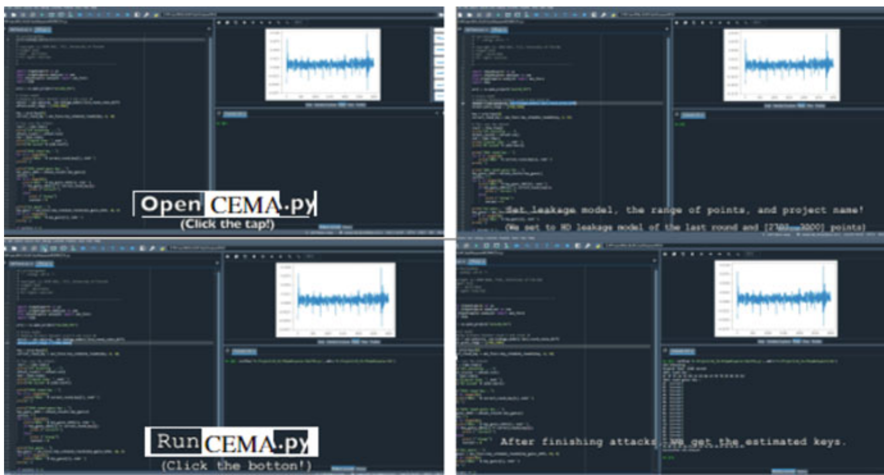


Fig. 9.8 CEMA attack steps

```
# Reopening our previously saved project, we now
specify the attack parameters for the CEMA attack.
Our HD leaking model and attack are used for this
hardware AES implementation.

# Only the ciphertext is needed for this attack;
the plaintext is not.

import chipwhisperer as cw
import chipwhisperer.analyzer as cwa
from chipwhisperer.analyzer import aes_funcs
import time

proj = cw.open_project("aes128_EM2")

# Attack model
# Hamming distance between round 9 and round 10
attack = cwa.cpa(proj,
cwa.leakage_models.last_round_state_diff)
attack.point_range = [2700,3000]

key = proj.keys[0]
correct_round_key = aes_funcs.key_schedule_rounds(key,
0, 10)

# This runs the attack:
start = time.time()
print("CEMA Attacking ....")
attack_results = attack.run()
end = time.time()
print("Elapsed time: ", end='')
print("%d second" % (end-start))
```

```
print("10th round key : ")
for i in range(16):
    print("%02x " % correct_round_key[i], end='')
print('')

print("10th round guess key : ")
key_guess_10th = attack_results.key_guess()
success = 1
for i in range(16):
    print("%02x " % key_guess_10th[i], end='')
    if key_guess_10th[i] == correct_round_key[i]:
        print (" Correct!")
    else:
        print (" Wrong!")
        success = 0

print("Key guess : ")
key_guess = aes_funcs.key_schedule_rounds(key_guess_10th,
10, 0)
for i in range(16):
    print("%02x " % key_guess[i], end='')
print('')

if success == 1:
    print ("Successful CEMA Attack!")
else:
    print ("CEMA Attack Failed!")
```

As shown in Fig. 9.9, successfully recovered all bytes of the last round key of AES-128 by mounting a CEMA attack using 10,000 traces and the secret key of the AES-128. CEMA on unprotected AES using Hamming distance EM model shows the first extracted key byte to be 10k traces (see Fig. 9.10). The correlation ratio (CR) which is defined as the ratio between the peak correlation coefficient of a correct key guess and the next highest correlation of an incorrect key guess was measured as $1.9\times$, indicating a successful attack. All the 16 key bytes were extracted by mounting a CEMA attack using 10,000 traces.

9.7 Conclusion

Electromagnetic emission from cryptographic ICs, a typical side-channel attack method, can be used to obtain the secret key without getting physical access to the device. This work has been done with the CW305 FPGA board with an oscilloscope and PC to perform the CEMA attack on AES implementation. This configuration can be applied to EM SCA on a variety of various targets, such as the usage of additional hardware cores (ECC, SHA, etc.). By the end of the course, students will have a theoretical understanding of electromagnetic (EM) attacks

```
Console 1/A ×

In [6]: runfile('E:/Project/AI_ML/ChipWhisperer/SW/CPA.py', wdir='E:/Project/AI_ML/ChipWhisperer/SW')
CPA Attacking ....
Elapsed time: 1348 second
10th round key :
af 6f 83 ae b2 d6 a2 82 0a e2 95 fd 85 05 64 5d
10th round guess key :
af  Correct!
6f  Correct!
83  Correct!
ae  Correct!
b2  Correct!
d6  Correct!
a2  Correct!
82  Correct!
0a  Correct!
e2  Correct!
95  Correct!
fd  Correct!
85  Correct!
05  Correct!
64  Correct!
5d  Correct!
Key guess :
3b d9 9b 8e 8a 21 76 b8 fa a8 16 43 9b 5b ab 84
Successful CPA Attack!
```

Fig. 9.9 Extracted all bytes of the last round key of AES-128 and secret key

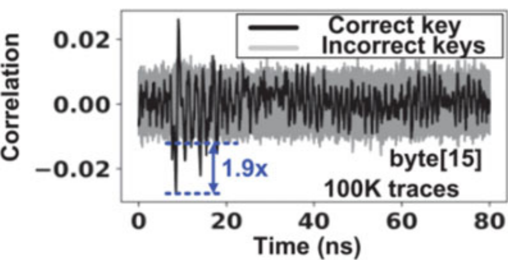


Fig. 9.10 Results of 15-byte attack on AES-128 using CEMA

and electromagnetic analysis attacks, as well as know how to use electromagnetic analysis to recover secret keys from an FPGA version of an AES core.

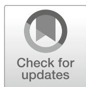
References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side—channel (s). In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 29–45. Springer (2002)
2. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms. Preprint arXiv:2204.07909 (2022)
3. Anandakumar, N.N.: SCA Resistance analysis on FPGA implementations of sponge based MAC PHOTON. In: Innovative Security Solutions for Information Technology and Communications, pp. 69–86. Springer, Cham (2015)
4. Anandakumar, N.N., Dillibabu, S.: Correlation power analysis attack of AES on FPGA using customized communication protocol. In: Proceedings of the Second International Conference

on Computational Science, Engineering and Information Technology, CCSEIT '12, pp. 683–688 (2012)
5. Anandakumar, N.N., Das, M.P.L., Sanadhya, S.K., Hashmi, M.S.: Reconfigurable hardware architecture for authenticated key agreement protocol over binary edwards curve. ACM Trans. Reconfigurable Technol. Syst. **11**(2), 1–19 (2018)
6. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Design and analysis of FPGA based PUFs with enhanced performance for hardware-oriented security. ACM J. Emerg. Technol. Comput. Syst. **18**, 1–26 (2022)
7. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Burlington (2018)
8. Das, D., Nath, M., Ghosh, S., Sen, S.: Killing EM side-channel leakage at its source. In: 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1108–1111 (2020). https://doi.org/10.1109/MWSCAS48704.2020.9184657
9. Das, D., Sen, S.: Electromagnetic and power side-channel analysis: advanced attacks and low-overhead generic countermeasures through white-box approach. Cryptography **4**(4), 30 (2020)
10. De Mulder, E., Buysschaert, P., Ors, S., Delmotte, P., Preneel, B., Vandenbosch, G., Verbauwhede, I.: Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem. In: EUROCON 2005—The International Conference on "Computer as a Tool", vol. 2, pp. 1879–1882 (2005). https://doi.org/10.1109/EURCON.2005.1630348
11. Dworkin, M.J., Barker, E.B., Nechvatal, J.R., Foti, J., Bassham, L.E., Roback, E., Dray, J.F.: Advanced Encryption Standard (AES). FIPS PUB 197 (2001). https://www.nist.gov/publications/advanced-encryption-standard-aes
12. EMV-Technik, L.: Rf1 Set, Near-Field Probes 30 MHz up to 3 GHz. https://www.langer-emv.de/en/product/rf-passive-30-mhz-3-ghz/35/rf1-set-near-field-probes-30-mhz-up-to-3-ghz/270
13. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 251–261. Springer (2001)
14. He, J., Guo, X., Tehranipoor, M., Vassilev, A., Jin, Y.: EM side channels in hardware security: attacks and defenses. IEEE Des. Test **39**(2), 100–111 (2022). https://doi.org/10.1109/MDAT.2021.3135324
15. Inc., N.T.: Cw305 Artix FPGA target. https://www.newae.com/products/NAE-CW305
16. Iyer, V., Wang, M., Kulkarni, J., Yilmaz, A.E.: A systematic evaluation of EM and power side-channel analysis attacks on AES implementations. In: 2021 IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 1–6 (2021). https://doi.org/10.1109/ISI53945.2021.9624778
17. Kumar, A., Scarborough, C., Yilmaz, A., Orshansky, M.: Efficient simulation of EM side-channel attack resilience. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 123–130 (2017). https://doi.org/10.1109/ICCAD.2017.8203769
18. Li, H., Markettos, A.T., Moore, S.: Security evaluation against electromagnetic analysis at design time. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 280–292. Springer (2005)
19. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards, vol. 31. Springer Science & Business Media, Berlin (2008)
20. Pundir, N., Park, J., Farahmandi, F., Tehranipoor, M.: Power side-channel leakage assessment framework at register-transfer level. IEEE Trans. Very Large Scale Integr. VLSI Syst. **30**, 1207–1218 (2022)
21. Quisquater, J.J.: A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions. The SEMA and DEMA methods. In: EUROCRYPT 2000 Rump Session (2000)
22. Stern, A., Botero, U., Rahman, F., Forte, D., Tehranipoor, M.: EMFORCED: EM-based fingerprinting framework for remarked and cloned counterfeit IC detection using machine learning classification. IEEE Trans. Very Large Scale Integr. VLSI Syst. **28**(2), 363–375 (2019)
23. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer Science & Business Media, Berlin (2011)

24. Wittke, C., Kabin, I., Klann, D., Dyka, Z., Datsuk, A., Langendoerfer, P.: Horizontal DEMA attack as the criterion to select the best suitable EM probe. In: Cryptology ePrint Archive (2018)
25. Zhang, T., Park, J., Tehranipoor, M., Farahmandi, F.: PSC-TG: RTL power side-channel leakage assessment with test pattern generation. In: 2021 58th ACM/IEEE Design Automation Conference (DAC), pp. 709–714. IEEE (2021)

Chapter 10
Logic-Locking Insertion and Assessment



10.1 Introduction

A shift in the integrated circuit (IC) design flow has resulted from the ever-increasing cost of maintaining a cutting-edge semiconductor fabrication facility, resulting in the creation of fabless semiconductor companies, third-party design firms, and contract foundries [8]. As a result of this trend, innovation has accelerated, costs have been lowered, and time-to-market has been shortened. Original intellectual property (IP) owners, however, are unable to monitor the entire process due to the fact that so many entities are involved in design, manufacturing, integration, and distribution throughout the world. Consequently, they are now facing threats such as IP theft/piracy, tampering, counterfeiting, reverse engineering, and IC overproduction [40]. Considering the vastly divergent nature of IP protection laws (and how they are enforced) across countries, IP protection must no longer rely merely on passive measures such as patents, copyrights, IC meters, and watermarks to deter these threats [5]. As a consequence, it is essential to develop proactive approaches that prevent such threats from ever occurring.

Several design-for-trust techniques have been proposed to address these concerns, such as IP encryption [19], logic locking [10, 34], state-space obfuscation [11], IC camouflaging [29], split manufacturing [20], and split testing [27]. A standard developed for protecting IP (IEEE-P1735) has been widely adopted in the semiconductor IP industry [19]. However, Chhotaray et al. [12] showed critical weaknesses in the standard that led to the extraction of the entire register-transfer-level (RTL) plaintext without the knowledge of the secret key. It is impossible for encryption alone to solve IC supply chain problems like overproduction and tampering, even if IEEE-P1735 limitations are overcome. To physically obfuscate the design, IC camouflaging has been introduced to implement logic gates with different functionalities, dummy vias, filler cells, etc. [29] that appear identical to prevent post-manufacturing reverse engineering. However, camouflaging does not also eliminate the threat of IC overproduction performed by the foundry after

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_10

fabrication. In addition, a foundry can extract the logical netlist since it has access to all the masking layers. In split manufacturing [20], threats introduced by an untrusted foundry are eliminated by manufacturing only the front-end of line (FEOL) layers in the high-end untrusted foundry, and back end of line (BEOL) layers in design house's trusted low-end foundry, thus hiding BEOL connections from the untrusted foundry. However, researchers proposed several attacks based on physical layout design heuristics [31], network-flow model [42], and placement and routing proximity [23] to extract missing back end of line connections. Logic locking involves adding additional logic gates to the design or increasing the state space in order to hide the functionality of the chip. Through the IC supply chain, logic locking can provide proactive hardware for trust protection against all previously mentioned threats if implemented meticulously [21]. In this chapter, we demonstrate different logic-locking techniques graphically in a sample circuit. In particular, this chapter can help a reader to better understand the concept of logic locking, different logic-locking techniques, and experimental implementation of different logic-locking techniques and perform security analysis using satisfiability-based attack and verification of the logic-locked circuit with the original circuit.

The rest of the chapter is organized as follows. Section 10.2 provides background on IP protection and logic locking. Section 10.3 discusses the existing logic-locking solutions. Section 10.4 outlines the experimental setup and details of the experimental steps. Finally, Sect. 10.5 concludes the chapter.

10.2 Background

10.2.1 Logic Locking

Logic locking or logic obfuscation hides original functionality from reverse engineering, piracy, overproduction, and tampering by inserting additional gates controlled by a key into a design. Figure 10.1 illustrates the proper operation of the design once the unlocking key inputs are provided from a tamper-proof memory along with the original functional inputs for [34]. Ideally, the locking key gates and the key itself should be chosen in a way that makes it physically impossible for an adversary to guess them (or extract them using other techniques). Researchers proposed several key-insertion techniques in search of an unbreakable logic obfuscation technique [21, 28, 30, 34]. Threats of logic obfuscation and tampering [32] have propelled government agencies like the Defense Advanced Research Project Agency (DARPA) to call for programs like "Automatic Implementation of Secure Silicon" (AISS) [14] under "Electronics Resurgence Initiative" [15] to automatically include logic locking in chip designs for IP protection.

The purpose of logic locking and obfuscation is to protect underlying hardware IPs from reverse engineering and overproduction by locking or obscuring them to some extent [41]. Through the different phases of the IC supply chain, the

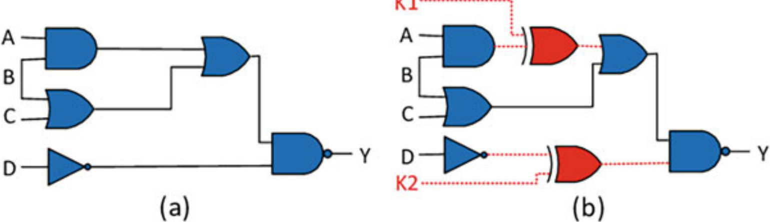


Fig. 10.1 Conceptual overview of logic locking. (**a**) an original circuit without key gates and key inputs, and (**b**) an encrypted circuit with key gates and key inputs

design remains locked. It is impossible to retrieve the correct functionality without knowing the correct locking (secret) key. A trusted facility activates locked ICs by burning a key into a tamper-proof memory after fabricated and packaged ICs are delivered. When the chip is powered on, the tamper-proof memory drives the correct key inputs. Researchers have developed various key gate insertion algorithms for developing a secure logic-locking technique, including random logic locking (RLL) [34], reconfigurable logic barriers [7], interference-based strong logic locking (SLL) [7], and fault-analysis based logic locking (FLL) [2, 28]. There is only one key to unlock the chip in all of these logic obfuscation schemes. The unlocking key remains static throughout the life cycle of the chip. In this instance, the logic is obfuscated using static obfuscation [13].

10.2.2 The Threat Model for Logic Locking

Based on a target technology library, we assume the design house performs logic design, verification, and synthesis as shown in Fig. 10.2. To perform logic locking, it inserts additional gates into the synthesized design. In order to improve testability, DFT structures are integrated once the obfuscated design has been re-synthesized. To fabricate the obfuscated GDSII, the design house sends the obfuscated GDSII to an offshore foundry (untrusted). The die is then sorted, diced, and packaged after fabrication. In a trusted facility, the dies are activated using unlocking keys. As soon as the functional ICs are ready, they can be delivered to the market. It is assumed that the attacker has access to the following:

- The attacker can either be a foundry or an end user. In the former case, the foundry can derive the gate-level netlist from the GDSII in its possession if the key gate is inserted. The latter assumes the netlist is generated from a de-processed chip's layout images.
- Unlocked IC: A single unlocked IC (i.e., Oracle) can return correct outputs for any input pattern. A rogue insider in a trusted supply chain, or on-field systems, can obtain such an IC from the open market.

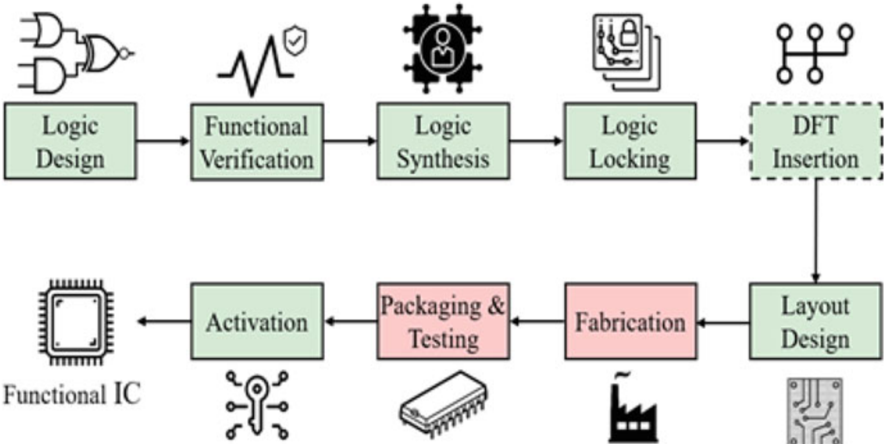


Fig. 10.2 Supply chain for a logic-locked design

- Scan chain: Since the basic SAT attack can only run on combinational circuits, scan chain access is required to partition a sequential circuit.

10.3 Review of Existing Logic-Locking Solutions

Logic-locking solutions available in the research community can be categorized into the following different groups.

10.3.1 Combinational Locking

10.3.1.1 Elementary Logic-Locking Solutions

This group refers to the very early inventions in logic-locking research. In this category, researchers used XOR gates to perform the locking as shown in Figs. 10.3 and 10.4. The approach for locking gate insertion was random [34], structural analysis-based [29], or fault analysis-based [30].

10.3.1.2 LUT and Routing Obfuscation

In this category, researchers performed logic-locking for programmable blocks, e.g., FPGA. The locking could be in the form of configuring the LUTs in the FPGAs for XOR operation [24] or hiding the connectivity using programmable crossbars [37].

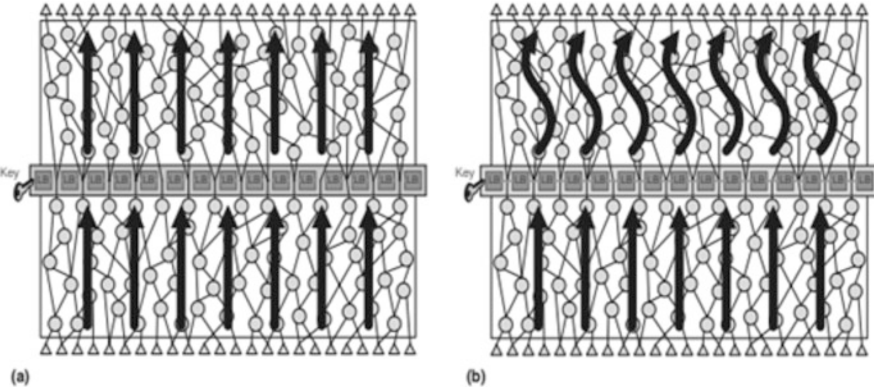


Fig. 10.3 In this study, logic barriers (LBs) have been found to block the information for (**a**) the correct key and (**b**) the incorrect key [24]

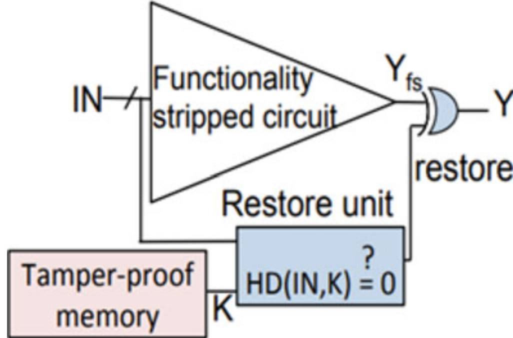


Fig. 10.4 The output is restored only for the correct key input. Otherwise, an incorrect functionality is observed at the output [44]

10.3.1.3 Point Function-Based Logic Locking

In this category, researchers performed logic-locking for programmable blocks, e.g., FPGA. The locking could be in the form of configuring the LUTs in the FPGAs for XOR operation [24] or hiding the connectivity using programmable crossbars [37].

10.3.1.4 Combinational Cyclic Obfuscation

SAT attack [38] requires the design circuit to represent a directed cyclic graph (DAG). Therefore, to resist SAT attack, a new category of the logic-locking solution has been invented that proposed inclusion of cyclic nature in the design [33, 36] by including feedback cycles under keys as shown in Fig. 10.5.

Performing SAT attack on this category of logic locking with an exit with an incorrect key significantly increases the pre-processing imperative to perform SAT attack. Nevertheless, a new variant of the SAT attack has been proposed, which

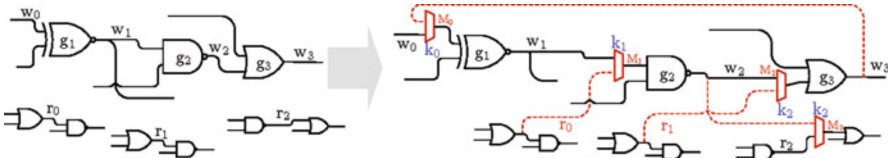


Fig. 10.5 Example of cyclic logic locking ensuring the removability of edges [36]

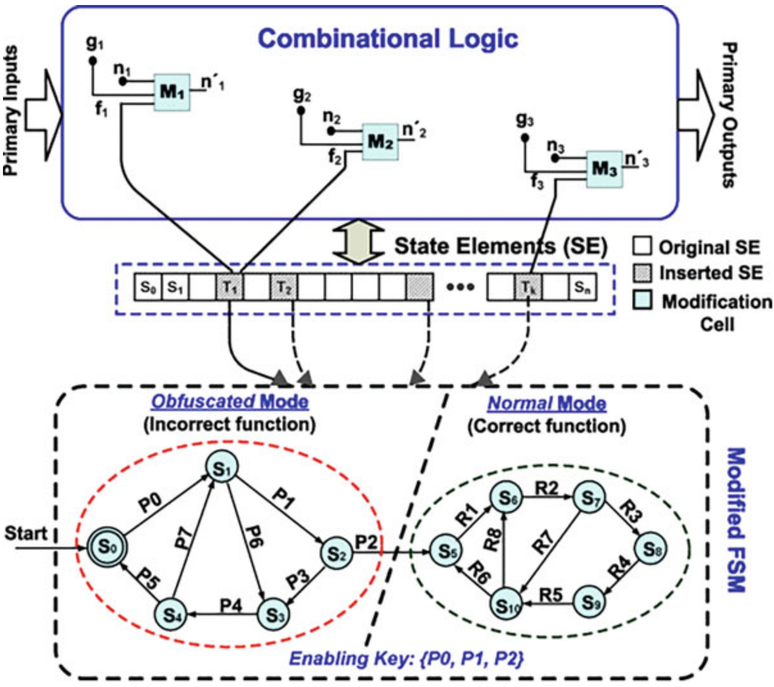


Fig. 10.6 Modification of the state-transition scheme for functional and structural obfuscation [11]

shows that feedback cycles pose extra challenges for attackers, but they can still be overcome using SAT-based approaches.

10.3.1.5 Sequential Obfuscation

The category of logic obfuscation restricts access to parts of FSM [3, 11, 16, 17]; they can alter the behavior of FSM transition if provided with an incorrect sequence of key inputs or deflect the state dynamically for gate-level logic obfuscation [17] (see Fig. 10.6).

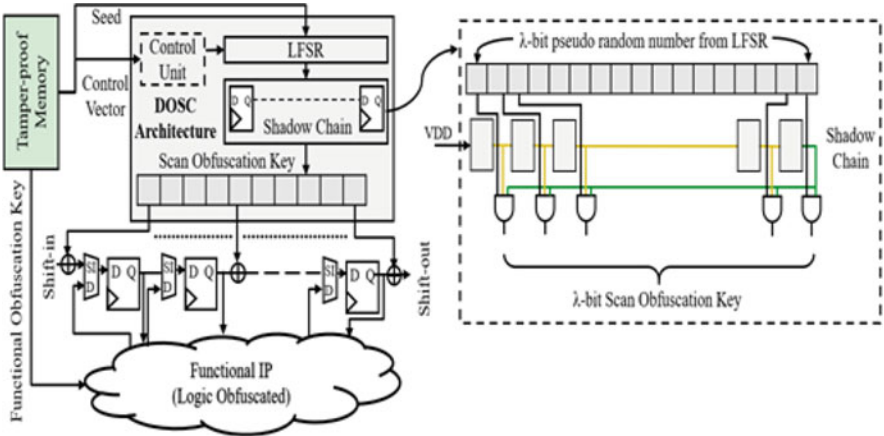


Fig. 10.7 Dynamic obfuscation of scan chain to protect against SAT attack [26]

10.3.1.6 Scan Obfuscation

SAT attack requires access to scan infrastructure when attacking sequential designs. Therefore, restricting any unauthorized party from having access to scan/debug ports will cripple the attacker from performing SAT attack. From this point of view, researchers proposed scan obfuscation of the sequential circuits along with logic obfuscation [4, 26]. This category of locking requires a functional locking method to protect the IP from reverse engineering [18]. The test patterns also need to be transformed based on the scan obfuscation technique, to finally perform the testability of the fabricated chips. The obfuscation in the scan circuitry gives the designer flexibility to utilize static [22] or dynamic locking keys [26] against oracle-guided attacks (see Fig. 10.7).

10.3.1.7 Parametric Logic Locking

While all the logic-locking categories, discussed so far, obfuscate either functionality or scan contents, parametric logic locking obfuscates the parametric features of the design, e.g., timing delay [39, 43], data flow, performance, etc. The rationale behind locking these attributes of design is to make sure that the attacker cannot model the locking features and perform the attack. However, timing-based SAT attack [9] has been proposed to deter the security of this category of locking. Like SAT attack, a satisfiability modulo theorem (SMT)-based attack [6] has been proposed that can break parametric logic-locking techniques.

10.3.1.8 Locking at Higher Level of Abstraction

Locking at the higher level of abstraction provides multiple advantages compared to gate-level abstraction. For instance, the designer knows more about the design's critical assets, operations, and states. As a result, it becomes easier for the designer to choose the locking options and candidates. Additionally, when synthesized to the gate level, the locked RTL design undergoes transformations and optimizations, providing resilience against structural and machine learning attacks. Most of the IP designs are done at the RTL level. So, it becomes imperative that the design is secured at a higher level to avoid security vulnerabilities in the next steps of the design flow. With this in mind, several techniques have been proposed to lock IP cores at RTL or even software level (at C program, before performing high-level synthesis) [25].

10.4 Experimental Demonstration

10.4.1 Experimental Setup

To run this experimental demonstration, we will require the following tools and software. Moreover, the experimental files source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch10_logic_locking/.

1. Linux platform (with Python version 3.6 or later)
2. Yosys synthesis tool (https://github.com/YosysHQ/yosys)
3. SAT attack tool (https://github.com/descyphy/Modified_SAT_Attack_on_Logic_Locking)
4. ABC synthesis tool (https://github.com/berkeley-abc/abc)
5. Any dependencies required by the above software

At the very beginning of the demonstration, we will insert locking gates into a sample benchmark (c17 from ISCAS'85 benchmark circuits) and graphically visualize how the insertion of locking gates changes the logic circuitry. The overall laboratory demonstration is divided into the following sub-tasks:

1. Locking gate insertion and graphical visualization
2. Check logic equivalency for correct unlocking key
3. Security evaluation by performing SAT attack

Now let us demonstrate each of the above sub-tasks in further detail.

10.4.2 Locking Gate Insertion

The original SAT attack tool [35] can perform some additional tasks along with performing SAT attack – basic combinational logic-locking gate insertion and logical equivalency checking. To generate a .dot graph of the locked circuit and visualize it graphically, we will utilize the open-source Yosys tool [45]. Logic gate insertion, logical equivalency checking, and SAT attack will be performed on a .bench format of the sample benchmark which is a library-independent and open-source netlist format. The general structure of the locking gate insertion command is as follows:

```
./sat_attack/bin/sle  /* tool binary
-locking_type  /* random/fault/gate type
-switches_specific_to_locking_type
/* fraction/graph etc.
-key-size or franction  /* number of key gates
name_of_the_benchmark /* input .bench format
```

Now let's get started into the different locking gate insertion methods.

10.4.3 Random Locking Gate Insertion

Please type the following command to change directory to the root directory of this module.

```
cd ~/m5_logic_locking_lab
```

Next, type the following command to insert random key gates in the c17 benchmark. The attribute value '2' of switch 'k' represents those two key gates that will be inserted. This command will generate a locked circuit with two key gates named 'c17_r2.bench' using random logic locking [34].

```
./sat_attack/bin/sle -r 1 -k 2 c17.bench > c17_r2.bench
```

To graphically visualize the locked circuit, we need to first convert this locked .bench to its equivalent structural Verilog format (.v) using ABC Synthesis tool [1]. Now, type the following command to invoke the ABC tool.

```
abc
```

Once the ABC tool is started, source the script 'bench2verilog.tcl' provided with this module to generate the associated Verilog format.

```
abc
UC Berkeley, ABC 1.01 (compiled Jul 10 2019 15:43:48)
abc 01> source ./bench2verilog.tcl
```

```
Entered genlib library with 28 gates
from file "cadence_new.genlib".
```

This action should generate a ‘c17_r2.v’ file in the current directory. Now type the following command which will load the Yosys synthesis tool and run the script ‘yosys_show.tcl’ provided with this module to visualize a network graph of the locked netlist.

```
yosys -s yosys_show.tcl
```

Network graphs are directed graphs of gates as vertices and interconnect among them as edges. Figure 10.8 shows the network graph of the original c17 benchmark, and Fig. 10.9 shows the graph of the same benchmark after two key gate insertions. The key inputs are highlighted in red in Fig. 10.9.

10.4.3.1 Fault Analysis-Based Key Gate Insertion

Key gate insertion methods have been proposed that maximize the hamming distance between correct and incorrect outputs [30]. To insert key gates using methods that model key gate insertion locations as faults, we need to type in the following command from the root directory (/m5_logic_locking_lab) of this module:

```
./sat_attack/bin/sle -t -f 0.2 -o c17_toc13xor_enc20.bench
./c17.bench
```

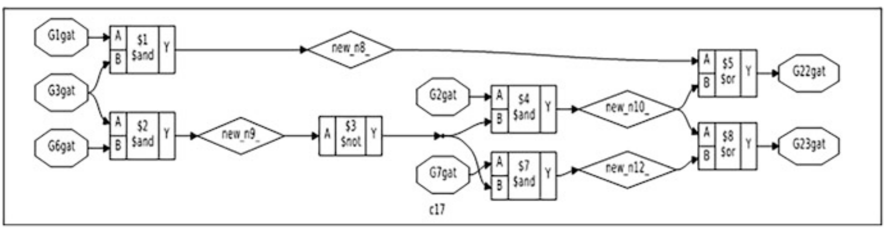


Fig. 10.8 Network graph of the original c17 circuit

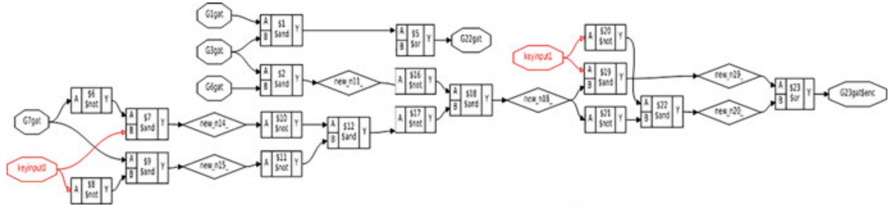


Fig. 10.9 Network graph of the c17_r2 circuit after 2-bit random locking gate insertion. Key inputs are highlighted in red

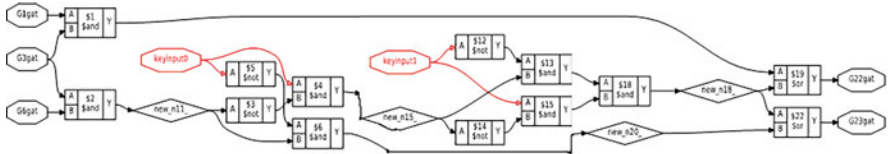


Fig. 10.10 Network graph of the c17_toc13xor_enc20 circuit. Key inputs are highlighted red

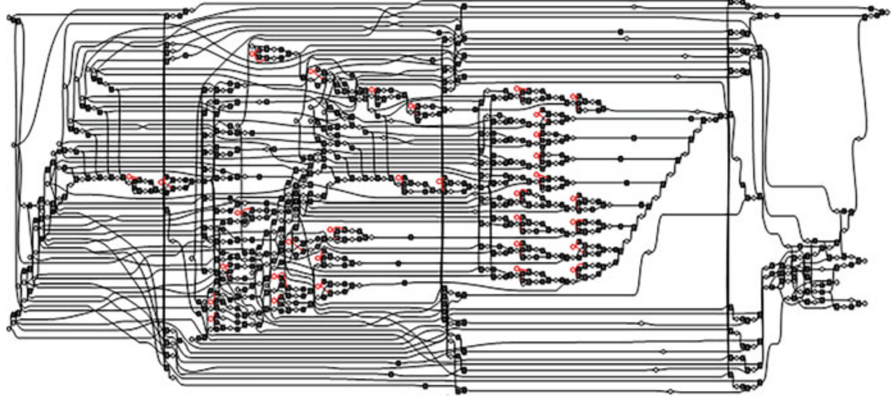


Fig. 10.11 Network graph of c432 by locking with method described in [30]. Key gates are highlighted red

In the above command, the value '0.2' of the switch '-f' represents the target hamming distance. After we invoke the above command, locked c17_toc13xor_enc20.bench will be generated in the current directory. Now to visualize the graph after fault-based key gate insertion, please run the following command to generate the associated Verilog file first:

```
abc 01> source ./bench2verilog.tcl
```

The above command will generate c17_toc13xor_enc20.v in the current directory. Once finished, please run the 'yosys_show.tcl' script using the following command to view the graph:

```
yosys -s yosys_show.tcl
```

In Fig. 10.10, the graph of c17 after fault analysis-based locking is shown. Please note the change in key gate location in comparison to Fig. 10.9.

The following graph (see Fig. 10.11) shows the graph of c432 after locking with a structural analysis-based corruptibility-guided approach [30].

10.4.3.2 Security Evaluation

Now let us perform the security analysis on the logic locking that we implemented in Sect. 10.4. We are going to perform SAT attack between the locked and unlocked .bench netlist to evaluate their resiliency. To perform SAT attack between the original circuit and the 2-bit random key gate inserted circuit, please type in the following command and hit enter:

```
attack/bin/sld c17_r4.bench c17.bench
inputs=5 keys=4 outputs=2 gates=11
iteration: 1; vars: 76; clauses: 22; decisions: 7
iteration: 2; vars: 94; clauses: 38; decisions: 9
finished solver loop. fail_count = 0
key=0010
iteration=2; backbones_count=0;
cube_count=166; cpu_time=0.010749;
```

Please note from the above output that SAT attack was able to extract the correct key input “0010” within just “0.010749” seconds in two iterations. It is noteworthy that satisfiability solvers run based on a heuristic and not a deterministic algorithm. Therefore, this attack time could be different in different trials. Next, we perform a 16-bit random logic locking and apply SAT attack to show that attack time increases with an increase in key size.

```
./sat_attack/bin/sld c17_r16.bench c17.bench
inputs=5 keys=11 outputs=2 gates=20
iteration: 1; vars: 144; clauses: 204; decisions: 27
iteration: 2; vars: 180; clauses: 312; decisions: 54
iteration: 3; vars: 216; clauses: 280; decisions: 72
iteration: 4; vars: 252; clauses: 388; decisions: 99
iteration: 5; vars: 288; clauses: 496; decisions: 164
finished solver loop. fail_count = 0
key=00100110011
iteration=5; backbones_count=0; cube_count=684;
cpu_time=0.015238; maxrss=4.5
```

From the above attack results, it is noticeable that the attack time increased with increasing key size.

10.4.4 Equivalency Checking

Using the SAT attack tool, we can also check if the extracted key is indeed the correct key by applying the key inputs to the locked circuit and performing a logical equivalency check between the locked circuit and the original circuit. Please type

in the following command to perform an equivalency check. We can notice that the logical comparison tool returned “equivalent” as the results.

```
./sat_attack/bin/lcmp c17.bench c17_r16.bench
key=00100110011 equivalent
If we apply a different key in the command above,
the outcome will be different.
./sat_attack/bin/lcmp c17.bench c17_r16.bench
key=00100110010 different; #cnt: 32
```

10.5 Conclusion

System-on-chip (SoC) logic locking protects the IPs embedded in modern SoCs from a wide range of hardware security threats at the IC manufacturing supply chain. IP piracy, reverse engineering, and overproduction are the main reasons to rely on logic locking. In this chapter, practitioners will learn how to apply logic locking in a target design, perform security analysis for logic locking, and verify that the locked design is correct. In this chapter, practitioners will learn about the fundamental concepts of logic locking as well as different logic locking techniques.

References

1. Abc synthesis tool (https://github.com/berkeley-abc/abc)
2. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms. Preprint arXiv:2204.07909 (2022)
3. Alkabani, Y.M., Koushanfar, F.: Active hardware metering for intellectual property protection and security. In: 16th USENIX Security Symposium (USENIX Security 07). USENIX Association, Boston, MA (2007)
4. Anandakumar, N.N., Hashmi, M.S., Tehranipoor, M.: FPGA-based Physical Unclonable Functions: A comprehensive overview of theory and architectures. Integration **81**, 175–194 (2021)
5. Anandakumar, N.N., Rahman, M.S., Rahman, M.M.M., Kibria, R., Das, U., Farahmandi, F., Rahman, F., Tehranipoor, M.M.: Rethinking watermark: providing proof of IP ownership in modern SoCs. In: Cryptology ePrint Archive (2022)
6. Azar, K.Z., Kamali, H.M., Homayoun, H., Sasan, A.: SMT attack: next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks. In: IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 97–122 (2019)
7. Baumgarten, A., Tyagi, A., Zambreno, J.: Preventing IC piracy using reconfigurable logic barriers. IEEE Des. Test Comput. **27**(1), 66–75 (2010)
8. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Burlington (2018)
9. Chakraborty, A., Liu, Y., Srivastava, A.: TimingSAT: timing profile embedded SAT attack. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–6 (2018). https://doi.org/10.1145/3240765.3240857

10. Chakraborty, R.S., Bhunia, S.: Hardware protection and authentication through netlist level obfuscation. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, pp. 674–677. IEEE Press (2008)
11. Chakraborty, R.S., Bhunia, S.: Harpoon: An obfuscation-based SoC design methodology for hardware protection. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **28**(10), 1493–1502 (2009)
12. Chhotaray, A., Nahiyan, A., Shrimpton, T., Forte, D., Tehranipoor, M.: Standardizing bad cryptographic practice. In: In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (2017)
13. Cui, A., Luo, Y., Chang, C.H.: Static and dynamic obfuscations of scan data against scan-based side-channel attacks. IEEE Trans. Inf. Forensics Secur. **12**(2), 363–376 (2017)
14. Darpa seeks to make scalable on-chip security pervasive (https://www.darpa.mil/news-events/2019-03-25)
15. Darpa electronics resurgence initiative. (https://www.darpa.mil/work-with-us/electronics-resurgence-initiative)
16. Desai, A.R., Hsiao, M.S., Wang, C., Nazhandali, L., Hall, S.: Interlocking obfuscation for anti-tamper hardware. In: Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop, CSIIRW '13. Association for Computing Machinery, New York, NY, USA (2013)
17. Dofe, J., Yu, Q.: Novel dynamic state-deflection method for gate-level design obfuscation. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(2), 273–285 (2018)
18. Guin, U., Shi, Q., Forte, D., Tehranipoor, M.M.: FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. ACM Trans. Des. Autom. Electron. Syst. **21**(4), 1–20 (2016)
19. IEEE recommended practice for encryption and management of electronic design intellectual property (IP). IEEE Std 1735-2014 (Incorporates IEEE Std 1735-2014/Cor 1-2015) pp. 1–90 (2015)
20. Jarvis, R.W., Mcintyre, M.G.: Split manufacturing method for advanced semiconductor circuits (2007). US Patent 7,195,931
21. Kamali, H.M., Azar, K.Z., Farahmandi, F., Tehranipoor, M.: Advances in logic locking: past, present, and prospects. In: Cryptology ePrint Archive (2022)
22. Karmakar, R., Chatopadhyay, S., Kapur, R.: Encrypt flip-flop: A novel logic encryption technique for sequential circuits. Preprint arXiv:1801.04961 (2018)
23. Magaña, J., Shi, D., Davoodi, A.: Are proximity attacks a threat to the security of split manufacturing of integrated circuits? In: 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–7 (2016)
24. Mardani Kamali, H., Zamiri Azar, K., Gaj, K., Homayoun, H., Sasan, A.: LUT-lock: a novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection. In: 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 405–410 (2018)
25. Muttaki, M.R., Mohammadivojdan, R., Tehranipoor, M., Farahmandi, F.: HLock: locking IPs at the high-level language. In: 2021 58th ACM/IEEE Design Automation Conference (DAC), pp. 79–84 (2021)
26. Rahman, M.S., Nahiyan, A., Rahman, F., Fazzari, S., Plaks, K., Farahmandi, F., Forte, D., Tehranipoor, M.: Security assessment of dynamically obfuscated scan chain against oracle-guided attacks. ACM Trans. Des. Autom. Electron. Syst. **26**(4), 1–27 (2021)
27. Rahman, M.T., Forte, D., Shi, Q., Contreras, G.K., Tehranipoor, M.: CSST: an efficient secure split-test for preventing IC piracy. In: 2014 IEEE 23rd North Atlantic Test Workshop, pp. 43–47. IEEE (2014)
28. Rajendran, J., Pino, Y., Sinanoglu, O., Karri, R.: Security analysis of logic obfuscation. In: Proceedings of the 49th Annual Design Automation Conference, pp. 83–89. ACM (2012)
29. Rajendran, J., Sam, M., Sinanoglu, O., Karri, R.: Security analysis of integrated circuit camouflaging. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 709–720. ACM (2013)
30. Rajendran, J., Zhang, H., Zhang, C., Rose, G.S., Pino, Y., Sinanoglu, O., Karri, R.: Fault analysis-based logic encryption. IEEE Trans. Comput. **64**(2), 410–424 (2013)

31. Rajendran, J.J., Sinanoglu, O., Karri, R.: Is split manufacturing secure? In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1259–1264. EDA Consortium (2013)
32. Robertson, J., Riley, M.: The Big hack: How China Used a Tiny Chip to Infiltrate U.S. Companies (https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies)
33. Roshanisefat, S., Mardani Kamali, H., Sasan, A.: SRCLock: SAT-resistant cyclic logic locking for protecting the hardware. In: Proceedings of the 2018 on Great Lakes Symposium on VLSI, GLSVLSI '18, p. 153–158. Association for Computing Machinery, New York, NY, USA (2018)
34. Roy, J.A., Koushanfar, F., Markov, I.L.: Epic: ending piracy of integrated circuits. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1069–1074. ACM (2008)
35. Sat attack tool (https://github.com/descyphy/Modified_SAT_Attack_on_Logic_Locking)
36. Shamsi, K., Li, M., Meade, T., Zhao, Z., Pan, D.Z., Jin, Y.: Cyclic obfuscation for creating SAT-unresolvable circuits. In: Proceedings of the on Great Lakes Symposium on VLSI 2017, GLSVLSI '17, p. 173–178. Association for Computing Machinery, New York, NY, USA (2017)
37. Shamsi, K., Li, M., Pan, D.Z., Jin, Y.: Cross-lock: dense layout-level interconnect locking using cross-bar architectures. In: Proceedings of the 2018 on Great Lakes Symposium on VLSI, GLSVLSI '18, p. 147–152. Association for Computing Machinery, New York, NY, USA (2018)
38. Subramanyan, P., Ray, S., Malik, S.: Evaluating the security of logic encryption algorithms. In: Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on, pp. 137–143. IEEE (2015)
39. Sweeney, J., Zackriya, V.M., Pagliarini, S., Pileggi, L.: Latch-based logic locking. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 132–141. IEEE (2020)
40. Tehranipoor, M.: Emerging Topics in Hardware Security. Springer, Berlin (2021)
41. Tehranipoor, M., Wang, C.: Introduction to hardware security and trust. Springer Science & Business Media, Berlin (2011)
42. Wang, Y., Chen, P., Hu, J., Li, G., Rajendran, J.: The Cat and Mouse in Split Manufacturing. IEEE Trans. Very Large Scale Integr. VLSI Syst. **26**(5), 805–817 (2018)
43. Xie, Y., Srivastava, A.: Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2017). https://doi.org/10.1145/3061639.3062226
44. Yasin, M., Sengupta, A., Nabeel, M.T., Ashraf, M., Rajendran, J.J., Sinanoglu, O.: Provably-secure logic locking: from theory to practice. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, pp. 1601–1618. Association for Computing Machinery, New York, NY, USA (2017)
45. Yosys synthesis tool (https://github.com/YosysHQ/yosys)

Chapter 11
Clock Glitch Fault Attack on FSM in AES Controller

11.1 Introduction

Devices that include cryptographic algorithms are susceptible to a variety of physical attacks in hostile environments, including side-channel analysis (SCA) and fault injection attacks (FIA). The security features incorporated into these devices can be successfully bypassed by these attacks, putting systems at danger. Among physical attacks, false injection attacks are a serious threat to secure devices because they are powerful and can be performed with very cheap equipment and minimal effort [5]. A device's operation is intercepted during a fault injection attack, a type of active side-channel attack that allows attackers to access sensitive data. The attacker alters the clock, temperature, and power supply connections, uses a high-powered laser, performs EM injection, or injects a fault into the system. The output bits can be corrupted by these flaws, and if they are placed carefully, they can also leak private information [7, 20].

Initially, Boneh et al. [9] suggested that computational errors that happened during the execution of a cryptographic algorithm can help to break it and uncover the secret key. They were given the idea, and it was successful in revealing the implementation's secret key. This concept, which had been proven to be effective in locating an RSA implementation's secret key, was presented to the audience. The differential fault attack (DFA) concept [16], which was developed with AES, was subsequently applied to symmetric block cipher implementations by the authors of [11]. DFAs are powerful and applicable against cryptographic hardware and compromise the security of a system-on-chip (SoC). During fault attacks, an attacker injects one or more faults during the process of SoC to produce erroneous results and then analyzes these results to extract secret information from a system and achieve illegal authentication [6]. On numerous security-critical applications, academic and professional researchers have effectively proven various fault injection attack types. This includes error correction codes (ECC) [15]; virtual machines [13]; radio-frequency tagging (RFID) [14]; microcontrollers [18]; encryption algorithms such

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_11

as DES, AES, and RSA [6, 22]; and analog sensors [17]. Nearly all platforms, including smart cards, FPGA-based embedded systems, system-on-chips (SoCs), and Internet of Things (IoT) devices, are susceptible to fault injection attacks, confirming the importance of this attack vector [4].

Over the past decade, most of the research on fault attacks are concentrated on analyzing the fault effects and developing countermeasures for fault injection on data paths. The control path's finite state machines (FSMs) are also vulnerable to fault injection attacks. As a result, if the FSMs in charge of the SoC are effectively attacked, the security of the SoC as a whole could be jeopardized. For instance, it has been demonstrated that when the FSM implementation of the AES algorithm is attacked using fault injection, the secret key of the AES encryption technique can be found [1, 19]. Thus, it is also extremely important to understand how fault injection attack works in an FSM and develop proper countermeasures to protect against fault attacks. Although there are many types of fault injection attacks, clock glitch fault injection is one of the most commonly used fault injection methods because they are low cost, easy to control the injection condition, and require little professional knowledge.

In this chapter, we demonstrate how to perform a clock glitch in an AES block implemented on an FPGA using a ChipWhisperer CW305 target board and show how to fail an AES execution by applying clock glitches that cause it which could produce inaccurate results during the AES encryption procedure. The remaining portions of the chapter are arranged as follows. Section 11.2 briefly discusses basic information on clock glitching, FSM, and AES. Section 11.3 briefly provides the experimental setup to perform fault attack using clock glitch. A performing clock glitch attack steps and glitch explorer and results are given in Sect. 11.4. Finally, conclusions are presented in Sect. 11.5.

11.2 Background

11.2.1 Fault Models

A fault model is a description of the type, reliability, effectiveness, and practicability of the introduced faults. The impact on the target device (such as bits or a nibble/byte/word) and the accuracy of location that targets one specific bit can both be used to define the nature of a problem. Repeatability refers to the attacker's capacity to reproduce the same error in either the temporal or geographic domain [10]. Efficiency describes the attacker's ability to select the fault value with accuracy. The practical characteristics (cost, equipment, time, and knowledge needed) of the fault attack are referred to as its feasibility. In general, it is acknowledged that a more exact/restrictive model is more challenging/unrealistic in a real-world setting.

Precise Bit Flip Fault attack proponents want to make as few assumptions as possible in order to demonstrate the effectiveness of their attacks. The most effective

DFA model involves flipping a specific bit (the round of the cipher as well as the location of the bit is chosen by the attacker). When performing the AND operation, $y = x0 \oplus x1$, consider the impact of modifying one input bit. Eve can get $x1$ if she can accurately lip, say, $x0$, by determining whether or not the output changes (an unchanged output means $x1 = 0$, whereas a changed output means $x1 = 1$). This makes it possible to attack any AND gate.

Single-/Multiple-Fault Adversary A single-fault adversary model is assumed in the majority of published research on fault attacks. According to this concept, the attacker Eve can inject faults into the cipher only once during a single execution, which could have an impact in several places. Injecting two sets of faults, for example, at the first and last rounds of encryption, will be regarded as a violation of the model.

Random/Deterministic Fault Model The random fault model is the one that appears most frequently in published works, e.g., [21]. Here, the attacker has some control over which round the faults can be introduced, but not over the value that is altered. Basically, the injected fault here causes one or more targeted bits of the operand value to be flipped. Attackers typically have control over the duration, position, and intensity of the external disturbance but not its precision. The target for fault injection can be a word (byte/nibble) or a string of bits, depending on the specific attack. Different models can be used instead of bit flipping, where certain bits are set to 1 or reset to 0 [8]. It is most frequently believed that random byte faults occur in byte-oriented ciphers like AES.

11.2.2 Clock Glitching

Digital hardware devices almost always expect some form of reliable clock. We can manipulate the clock being presented to the device to cause unintended behavior. The clock signal can be tampered with to produce setup or hold time violations, which is a very inexpensive and nonintrusive method of injecting faults [23]. A clock glitch is the alteration of an ideal clock signal by adding a small pulse within the larger rising clock edge, as shown in Fig. 11.1a. Figure 11.1b shows a typical sequential logic path.

In order to ensure that the sequential element receives the correct/stable value during normal operation, the clock cycle (TCLK) should be longer than the combinational logic's maximum path delay (τ). Register B may capture the incorrect data when a clock glitch occurs because T_g is smaller than tau, which will introduce a fault and cause it to spread throughout the circuit. Such a processing error may cause an instruction to be skipped or incorrect data to be stored in the memory modules [23]. Additionally, a clock error might cause flip-flops or registers to latch the incorrect data. For instance, it has been demonstrated that if the key register in a crypto engine fails to latch one bit of the key correctly, this key bit can be determined by contrasting the incorrect and correct output of the crypto engine.

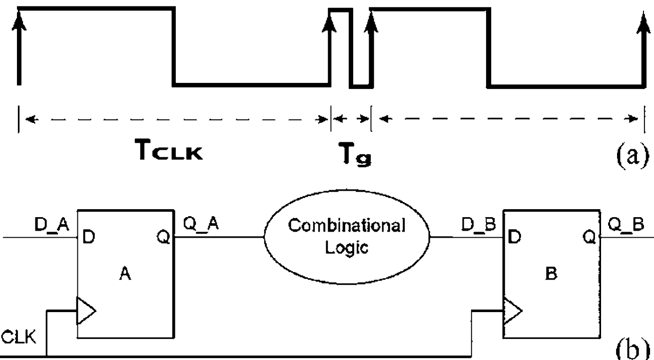


Fig. 11.1 (**a**) Clock signal glitch, (**b**) a common sequential logic path [23]

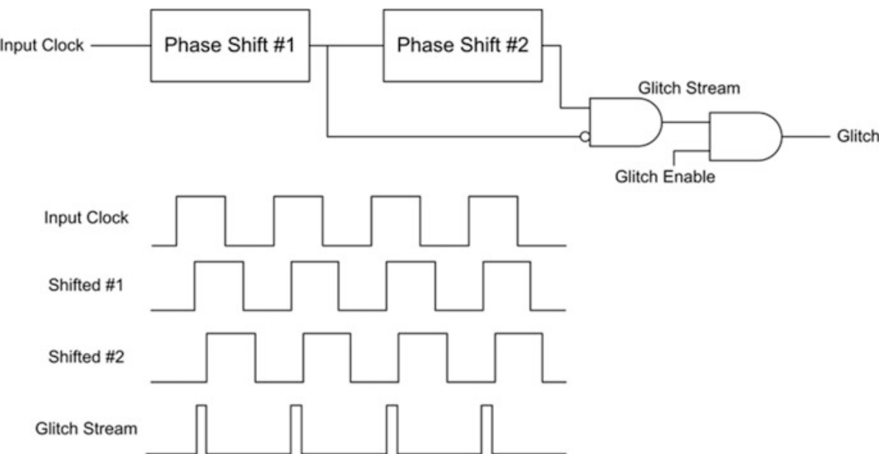


Fig. 11.2 Generation of glitches using Digital Clock Manager (DCM)

Such faults are temporary, allowing for their injection without leaving any evidence of manipulation [23]. In our experiment, we use the ChipWhisperer glitch system that uses the same synchronous methodology. A system clock is used to generate the glitches. These glitches are then inserted back into the clock. The generation of glitches is done with two variable phase shift modules configured as shown in Fig. 11.2: the FPGA's Digital Clock Manager (DCM) blocks are used by the phase shift blocks. For more details, please refer to the link: https://wiki.newae.com/Main_Page.

The enable line is used to determine when glitches are inserted. Glitches can be inserted continuously or triggered by some event. Figure 11.3 shows how the glitch can be muxd to output to the device under test (DUT).

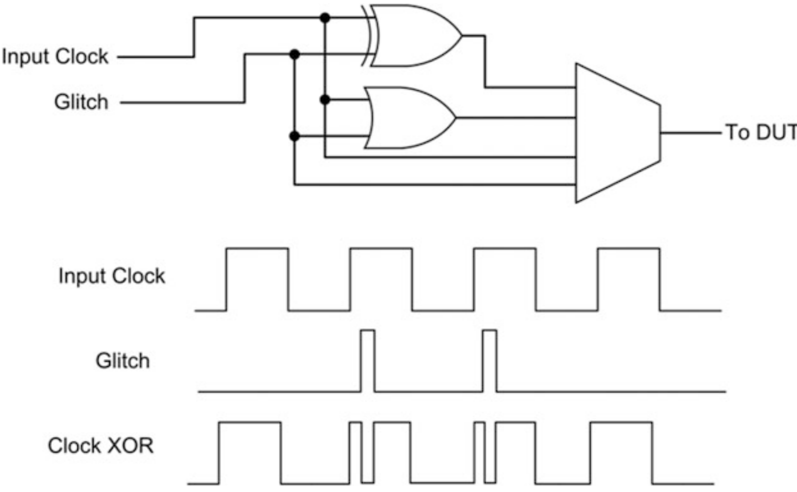


Fig. 11.3 XORing the glitch with input clock

11.2.3 Brief Description of AES

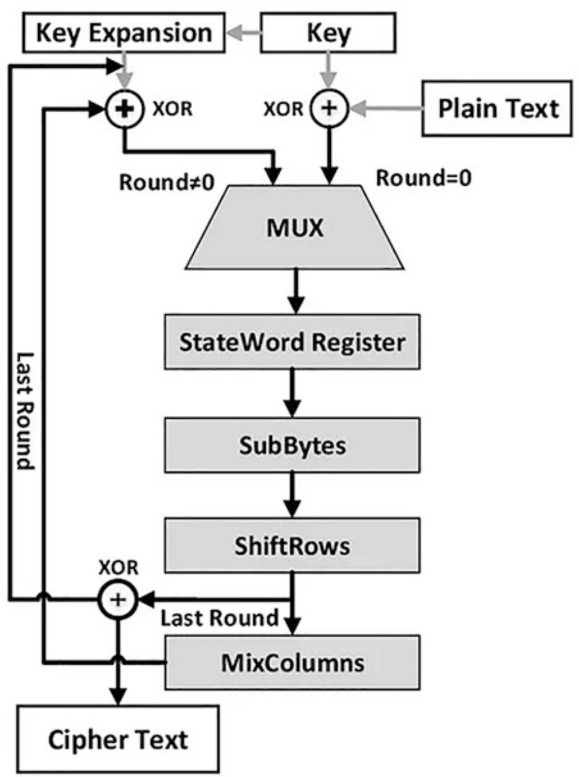
The AES algorithm is a symmetric block cipher that uses the same key to both encrypt and decrypt data.

The AES algorithm is a symmetric block cipher that can encrypt and decrypt information using the same key. The AES is defined for 128-bit blocks and key sizes of 128, 192, and 256 bits. The 128-bit plaintext is viewed as a 4×4 byte matrix, called a state byte. The AES operates on the states by iterating transformation rounds as shown in Fig. 11.4. The initial round consists of the AddRoundKey operation, the next rounds consist of applying successively the transformations SubBytes, ShiftRows, MixColumns, and AddRoundKey, but the last round omits the MixColumns transformation [2]. The state operations are defined briefly:

- SubBytes is the substitution step where a nonlinear function is applied on the input byte. This SubBytes is the composition of two transformations: an inversion in F_{2^8} and an affine transformation.
- ShiftRows performs a cyclic rotation on each row of the state.
- MixColumns linearly combines the elements in each column. It can be represented as a multiplication of each column with a constant matrix.
- AddRoundKey ties the result to the key by XORing each element with an element from the current round key.

Depending on the key size, the number of rounds is altered to 10, 12, or 14. In this chapter, we deal with the 128-bit AES due to its widespread usage. For a complete description and explanation of AES, please refer to [12].

Fig. 11.4 AES encryption process



11.2.4 Clock Glitch Attack on FSM in AES Controller

In the case of AES, this glitch could be used to cause its FSM (finite state machine) to skip states and either provide an incorrect ciphertext or give an adversary access to the secret encryption key. When focusing on attacking an FSM, we must look at how that FSM is encoded to see what kind of vulnerabilities it may have. For instance, let's say an FSM for AES has the states WAIT KEY, WAIT for DATA, INITIAL ROUND, DO ROUND, and FINAL ROUND and is binary encoding those states using the bit configurations of 001, 010, 011, 100, and 000, respectively (see Fig. 11.5). When the FSM is moving from state WAIT KEY to WAIT DATA, or 001 to 010, both the middle bit and least significant bit are flipping from 0 to 1 or 1 to 0. Because the flip flops for these bits may have different delays, if a clock glitch occurs and the state is set too early within the flip flops, the user could get the state 000 instead of the expected 010. This would mean the FSM was transferring directly from WAIT KEY to FINAL ROUND, theoretically giving the secret key as an output rather than encrypted ciphertext. Therefore, the encoding of FSM states for encryption algorithms is critical in performing attacks on the FSM. In this chapter, we demonstrate how to perform a fault attack using a clock glitch and exploit the

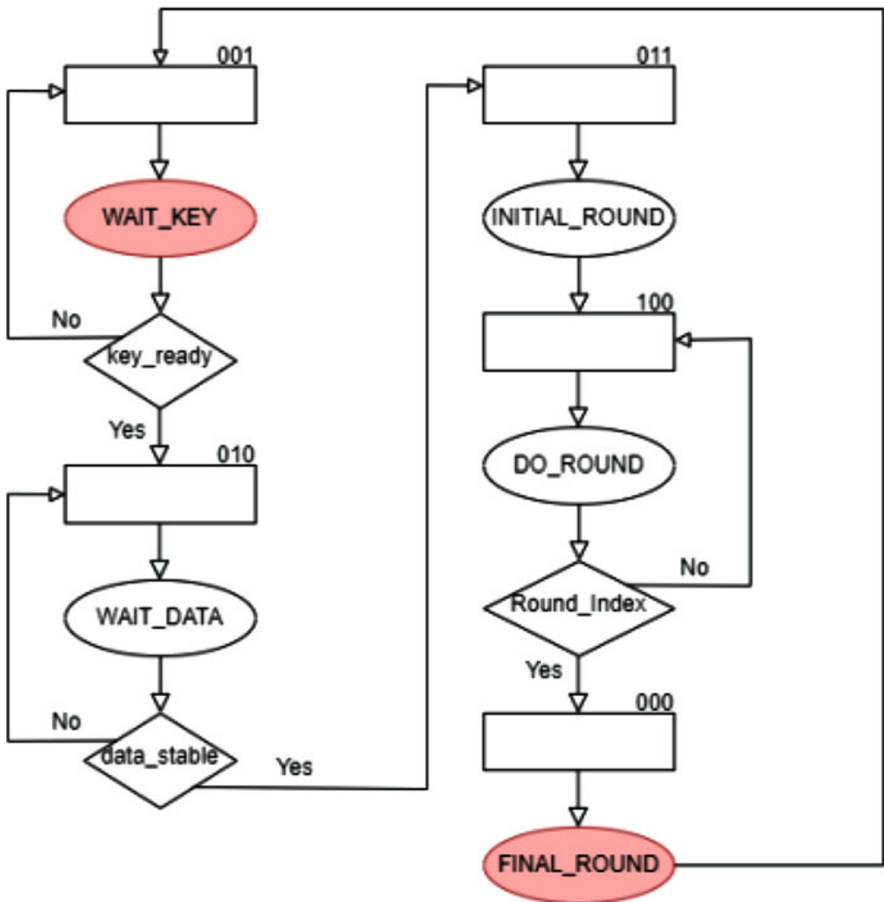


Fig. 11.5 AES finite state machine (FSM) with binary encoding

FSM vulnerabilities. This tutorial's objective is to demonstrate how to utilize clock glitches to make the AES execution fail. We may be able to corrupt some of the FSM states' bits by causing a brief clock glitch to occur immediately before a clock edge. If we're successful, the ciphertext that is produced will be completely different. Remember that diffusion is one of the main objectives of cryptographic algorithms: if we change just one bit of the input, the round function will cause that one bit to affect all 128 bits of the output.

This specific vulnerability is illustrated in Figs. 11.6 and 11.7. Figure 11.6 shows the vulnerability in terms of the intended state and captured state. Figure 11.7 illustrates the example in terms of the speed at which the individual flip-flop bits change as they are captured early by a glitched clock versus an ideal clock. When changing from state 001 to state 010, if the middle bit's delay was greater than the

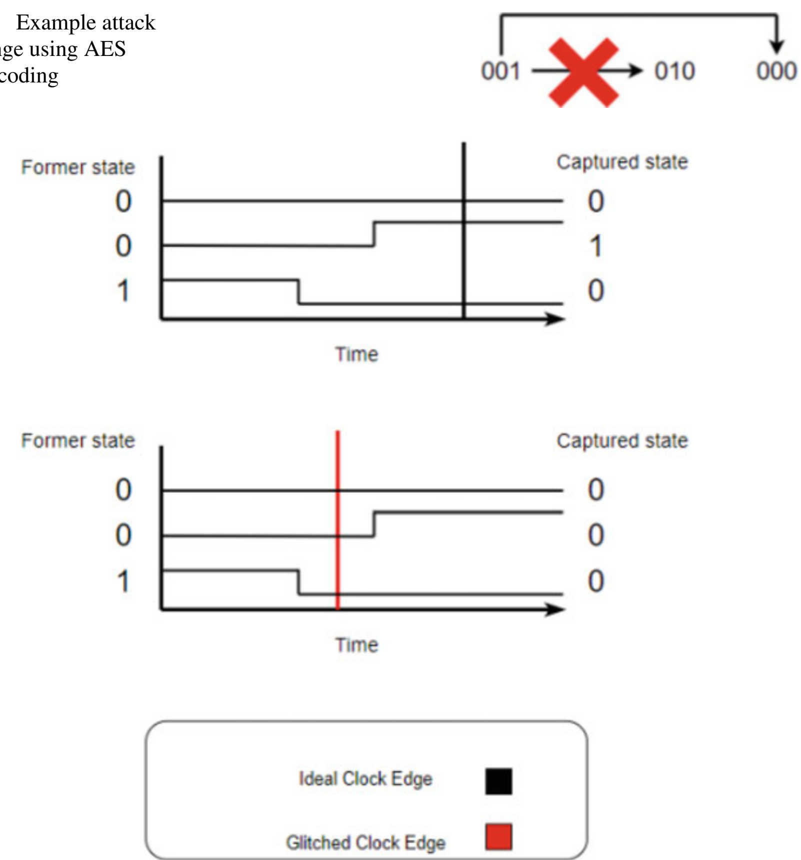


Fig. 11.6 Example attack state change using AES binary encoding

Fig. 11.7 State capturing with an ideal versus glitched clock

most and least significant bits, clock glitching could allow the register to capture a state at the wrong time, 000.

11.2.5 ChipWhisperer CW305 Board

For the study of embedded hardware security, ChipWhisperer provides a collection of several helpful tools. There are ChipWhisperer hardware targets, ChipWhisperer target device firmware, target device FPGA blocks, ChipWhisperer analysis software and libraries, as well as ChipWhisperer-Capture devices (which sample power measurements). As a standalone target, the CW305 board enables the use of a larger FPGA target to implement cores like AES and ECC [3].

11.3 Experimental Setup

In this part, we'll demonstrate a clock glitch fault attack using the ChipWhisperer-lite and ChipWhisperer CW305 boards. Figure 11.8 depicts the experiment's setup.

The CW305 FPGA board features a USB interface to talk to the FPGA, an external PLL for clocking the FPGA, a programming VCC-INT supply. The *Algorithm Under Test* is the algorithm we want to test. The goal of performing side-channel power analysis on this method is supported by the remaining circuitry. We may easily load input, keys, output, or trigger operations by using the *Register Interface* to match our Python code on the control computer. Physically, the CW305 acts as an Address/Data Bus between the FPGA and the microcontroller for the USB interface. This address/data bus enables you to establish a conventional address/data bus on the FPGA instead and write any data into the FPGA. Through ChipWhisperer-Lite, the PC (personal computer) downloads the design to the ChipWhisperer CW305 board, sends the plaintext and key to the CW305 board, and deals with the data received from the CW305 board. Please refer to Chap. 8 for firmware setup and ChipWhisperer boards configuration and how to connect a ChipWhisperer-Lite to the CW305 board. Additionally, we want to exploit the clock produced by our capture devices to inject glitches into this clock. Switch J16 must be set to 1 in order to accomplish this. By flipping this switch, the FPGA will be forced to use the ChipWhisperer's input clock rather than the PLL. To stop the output of the return clock, we can also set K16 to 0. The Switch (S2) of CW305 board should now look like the image as shown in Fig. 11.9.

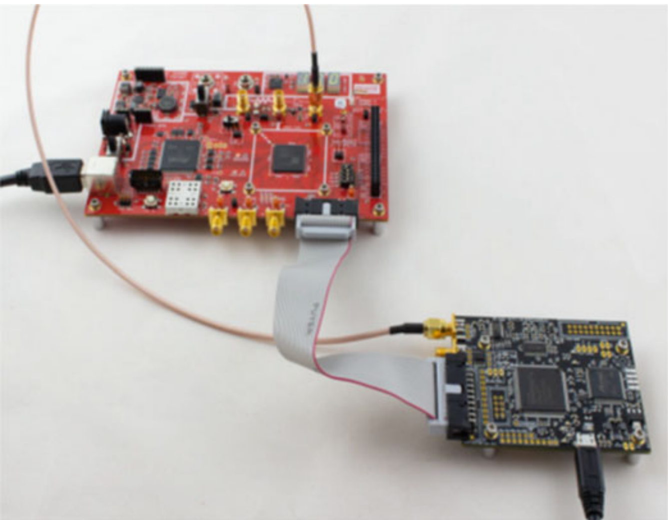


Fig. 11.8 The CW305 interconnected to the ChipWhisperer-Lite Capture board

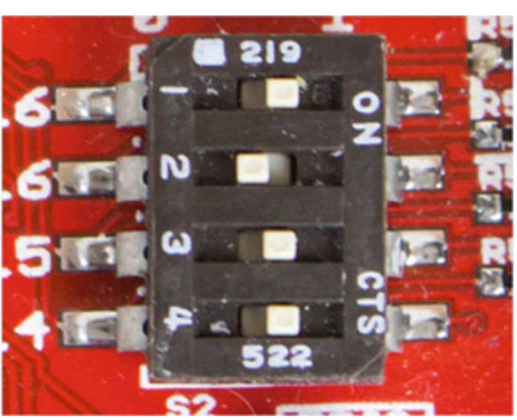


Fig. 11.9 Switch (S2) of CW305 board configuration

11.4 Performing Clock Glitch Attacks

First, we need to download the ChipWhisperer software V4.0.1 from the following link: https://github.com/newaetech/chipwhisperer/releases and install it on the control computer (PC). The ChipWhisperer Python library can be used to communicate with the NewAE Capture and target boards. Once the ChipWhisperer software is installed, we'll need the Xilinx Vivado software for AES bitstream generation. In this connection, the fully featured versions of the Xilinx Vivado software with license are required. However, the WEBPACK version is free for our target Artix-7 FPGA. How to download and Install the Xilinx Vivado Design Suite is discussed in Chap. 8. For this tutorial, we will provide a pre-existing AES-128 encryption example with a couple of project files to build a project using the Xilinx Vivado software. There are three steps that Vivado takes to turn our Verilog into a bitstream code: (1) synthesis in which the Verilog code is synthesized into a gate-level representation, (2) implementation in which the synthesized logic is routed to fit onto the device, and (3) generate the bitstream. Once the bitstream is generated, we need to identify its path. It is usually stored in the project file folder. We need its directory to insert it in the code. The CW305 Artix FPGA target and ChipWhisperer-Lite were used to load the AES algorithm and inject clock glitches reliably. Moreover, all Verilog design files and all python sources can be found at http://cad4security.org/index.php/trainings/hsl/ch11_clock_glitch_fia/.

11.4.1 Performing Clock Glitch Attack

1. First, open the ChipWhisperer capture software from the toolbar below and choose the Python console as shown in Fig. 11.10. In the middle window, you can see the files in chipwhisperer/software/chipwhisperer/capture/scripts directory.

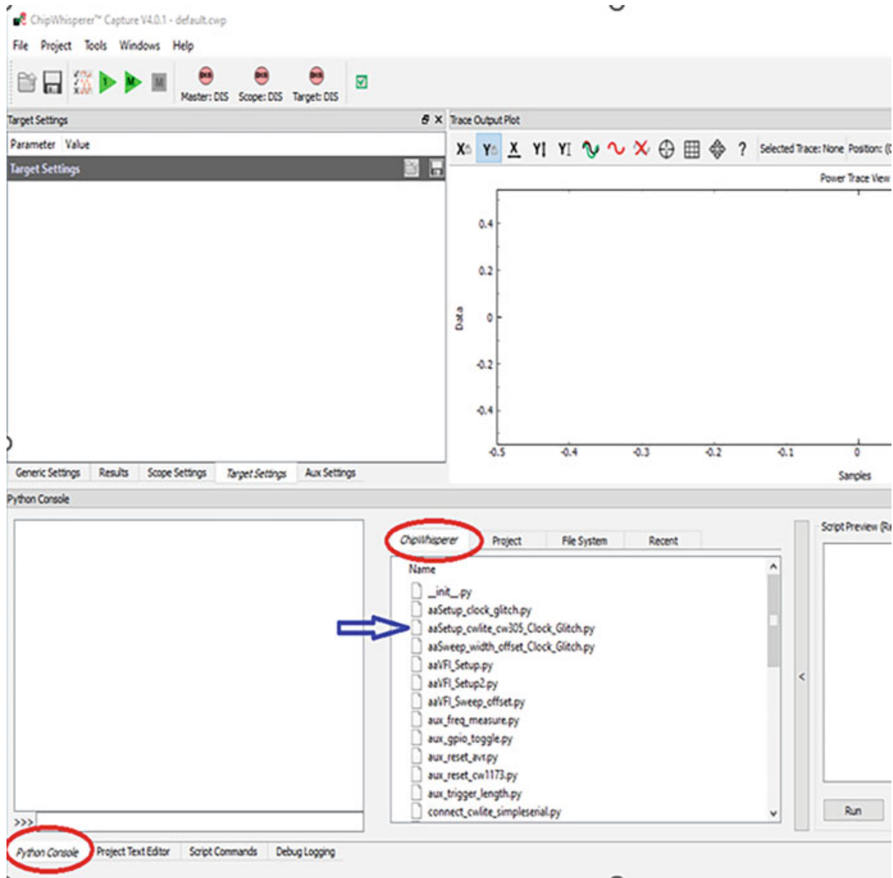


Fig. 11.10 ChipWhisperer GUI: Python console

2. Then select the file called aaSetup_cwlite_cw305_clock_glitch.py (or it might be called aaSetup_cwlite_cw305). Afterward, we are able to see the script preview in the left window. Click on the Run button and then we could see in the Python console if it ran correctly or not. In Fig. 11.11, we should see there is no problem with the Python code running. Then we can also check the log to see if ChipWhisperer and FPGA board are connected and if FPGA is programmed.
3. Next, we must click on the Mater Button as shown in Fig. 11.12. All three buttons should turn green at this point.
4. Then we want to do glitch setup using aaSetup_clock_glitch.py script. First, open the file and check the bitstream path. If it doesn't match the one that was generated previously, modify it. Then, select it from the middle window and run the script: aaSetup_clock_glitch.py. Now go to Scope setting and change the glitch trigger from continuous to Ext Trigger: Single-Shot as shown in Fig. 11.13

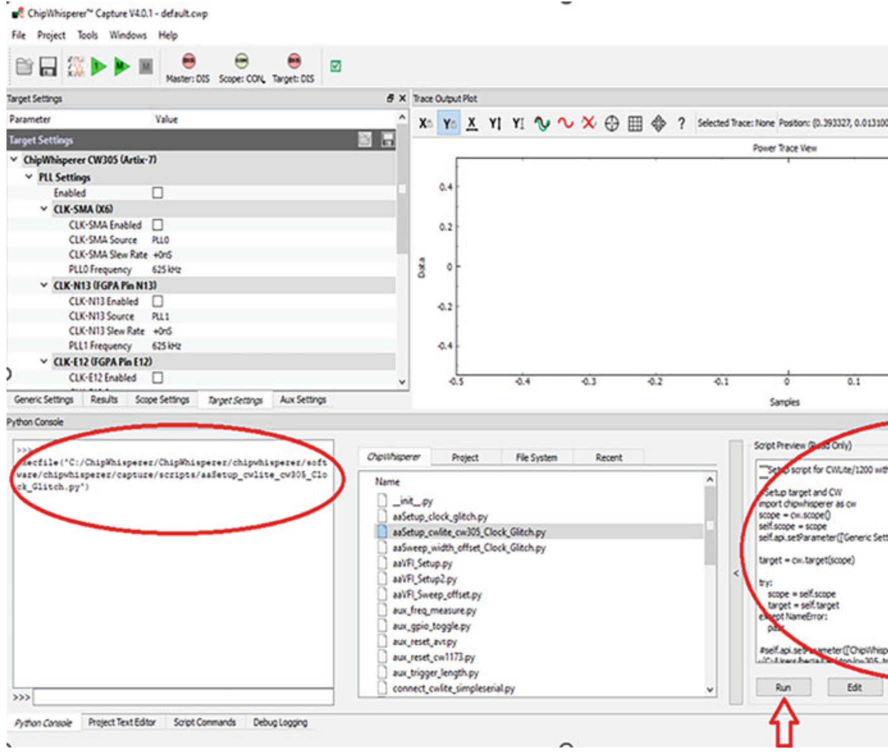


Fig. 11.11 Setup ChipWhisperer-Lite with cw305 FPGA board

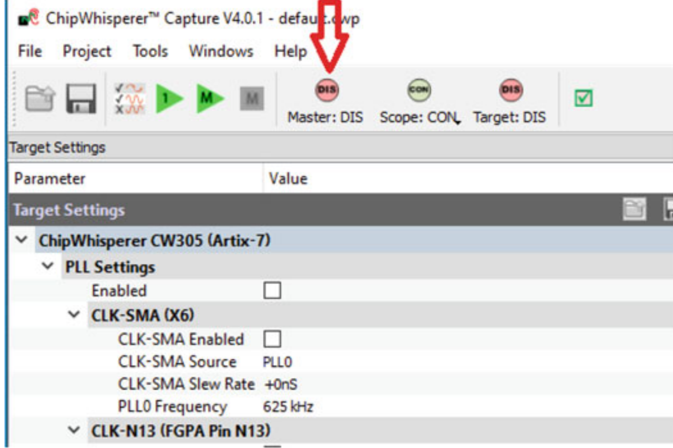


Fig. 11.12 Setup ChipWhisperer-Lite with cw305 FPGA board

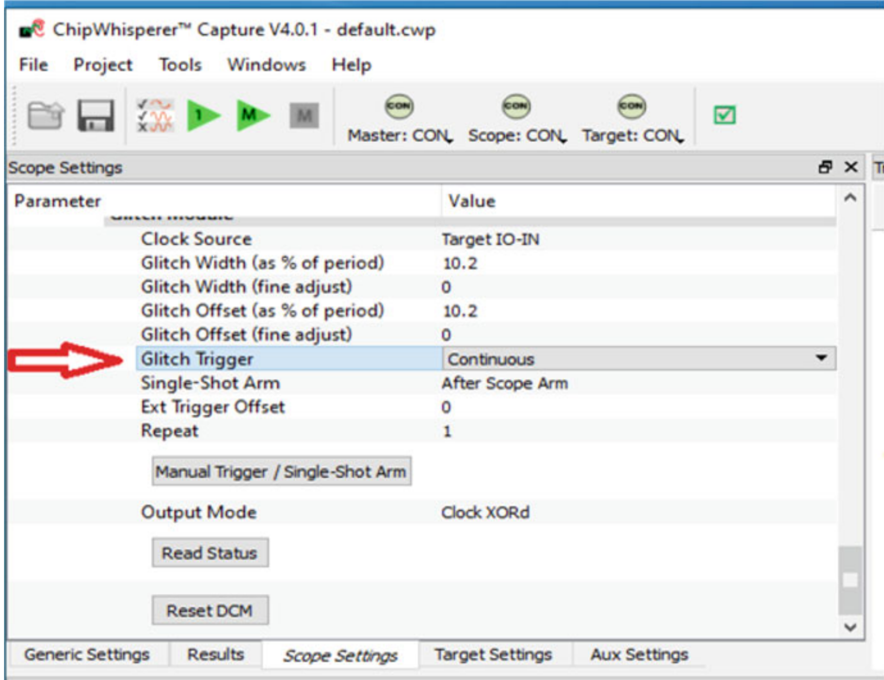


Fig. 11.13 Glitch trigger setup

5. Then, from the toolbar above, the ChipWhisperer GUI select Tools/Glitch Explorer. In the window that is open, edit the normal and successful response as shown in Fig. 11.14. In the example, if we receive the correct value, that is considered a normal response; otherwise, it means the fault attack was successful. Moreover, click on the Plot Widget button to plot the data vs. samples.
6. Next, click on Run button of the M glitches in the toolbar as shown in Fig. 11.15.
7. Now we should be able to see the power trace view and glitch map. If we don't see any glitches in this step, we can still do the glitch exploration part and there is a good chance to see glitches there if we change the value glitch and offset.
8. Finally, modify the Target HS IO-Out under CW Extra Settings so that it uses the output from the Glitch Module. Obtain a trace and confirm that we can see glitch in the power trace (see Fig. 11.16). Here is an example of a trace from samples 30–35 that clearly has a glitch. (We may need to adjust the offset and width of the glitch; this screenshot was taken with 30% and 10%.)

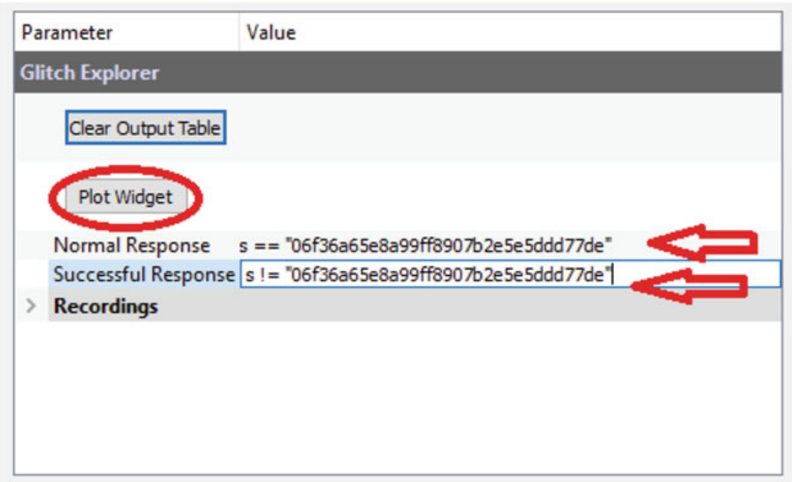
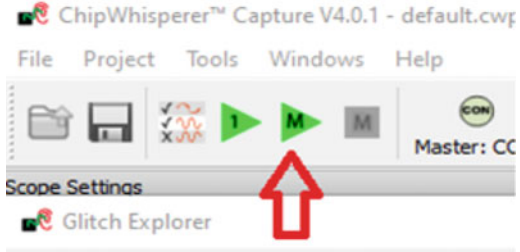


Fig. 11.14 Edit the normal and successful response

Fig. 11.15 Click on run button of the M glitches



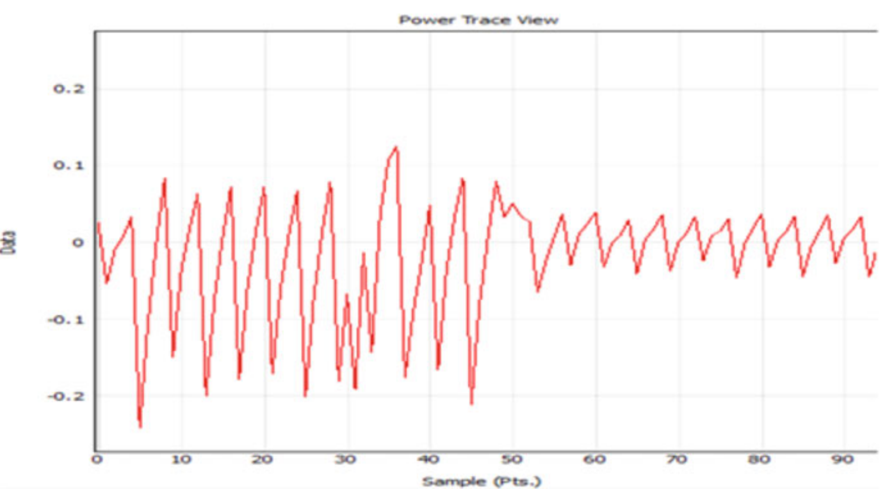


Fig. 11.16 Glitch in the AES-128 power trace

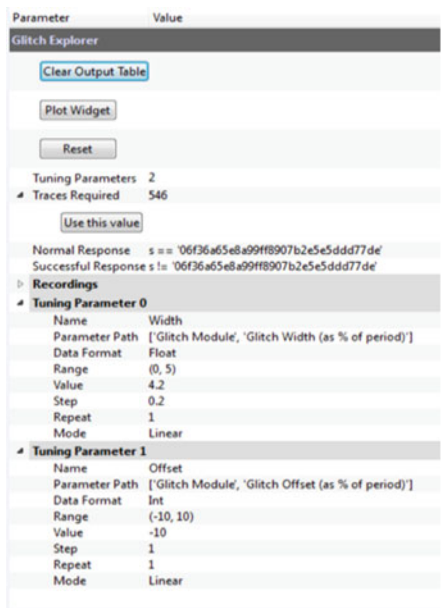


Fig. 11.17 Set up the glitch explorer

11.4.2 Glitch Explorer

The goal is to identify a set of clock glitch parameters that lead to the encryption process failing once we have successfully implemented clock faults. Let's set up the glitch explorer to look for glitches so that we can do this automatically (Fig. 11.17).

Our FPGA must first be configured to use a fixed plaintext and key. It will be harder to determine when a glitch was successful if we alter the inputs for every capture. Although we can use any key and plaintext, the remainder of this lesson will use the fixed key 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C and the fixed plaintext 5C 69 2F 91 03 B2 30 29 14 D7 E5 55 E4 DC EE 49. We can configure the glitch explorer to know when we've successfully glitched the FPGA using our fixed key and plaintext. To view the output format, open the glitch explorer and click Capture 1. The received output is "06f36a65e8a99ff8907b2e5ddd77de" for the aforementioned plaintext and key. Set this string to be checked for in the glitch explorer's normal/successful responses. The width and offset of the glitch module should then be swept using two tuning settings. When everything is configured, our glitch explorer should like Fig. 11.18. Important note: If our version of the ChipWhisperer GUI does not show tuning parameters, we must use the given Python code to sweep the glitch width and offset www.aaSweep_width_offset_clock_glitch.py and run it. This will be visible in the GUI under the Python coding examples and can be edited as needed to fit the described parameters. If you need help with what parameters to edit, this link is useful for the syntax to use to change parts of the GUI to fit our needs in Python: https://chipwhisperer.readthedocs.io/en/latest/api.

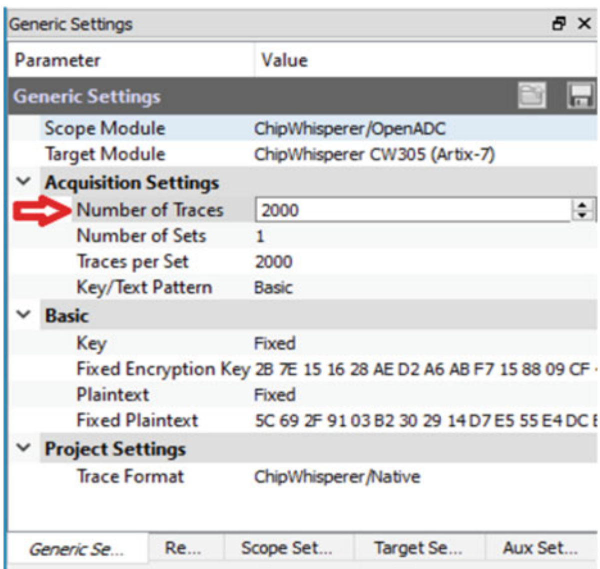


Fig. 11.18 Generic settings

html. Once everything is set up, either tuning parameters on the GUI or the relevant sweeping code have been run, we can now start capturing.

11.4.3 Results

There are two locations where the outcomes can be checked after our capture is complete. The glitch map first identifies the widths and offsets of successful glitches (see Fig. 11.19). The combinations of tuning parameters that led to the encryption failing are indicated by the green spots on this map. Our glitch map should, hopefully, like the one below (note that it might change every time we run it).

This map only provides a hint as to where to go in the glitch explorer; it doesn't reveal what the actual ciphertext was with any of these settings. The output of the glitch explorer is another thing we may examine. Looking at the glitched ciphertexts reveals a wide range of different outputs with various glitch settings (see Fig. 11.20). We have experimentally demonstrated the AES execution by applying clock glitches that cause it to produce erroneous results during the AES encryption process. Note: we must clarify in the glitch terminal what is deemed as "Normal" and a "Success." In our case, "Normal" can be equal to the expected ciphertext and "Success" can be anything not equal to that value.

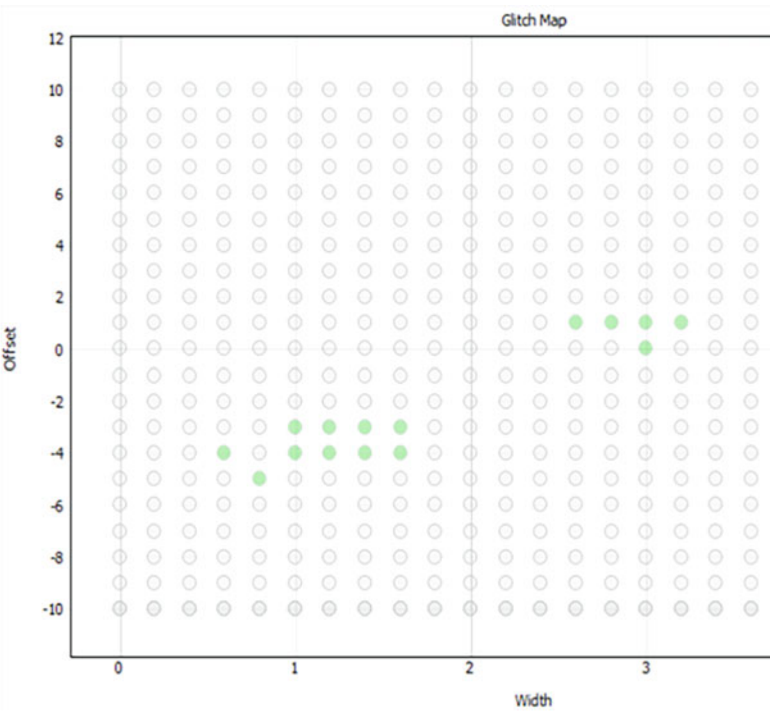


Fig. 11.19 Glitch widths and offsets caused successful glitches

	Status	Sent	Received	Date	Width	Offset
361	Normal		'06f36a65e8a99ff89...	16:03:27	0.4	-1
362	Normal		'06f36a65e8a99ff89...	16:03:27	0.2	-1
363	Normal		'06f36a65e8a99ff89...	16:03:26	0.0	-1
364	Success		'126685c180e68c41...	16:03:26	4.8	-2
365	Success		'126685c180e68c41...	16:03:26	4.6	-2
366	Success		'd8f1ce78e2c74895...	16:03:26	4.4	-2
367	Success		'a49528d42ac8afb6...	16:03:25	4.2	-2
368	Normal		'06f36a65e8a99ff89...	16:03:25	4.0	-2
369	Normal		'06f36a65e8a99ff89...	16:03:25	3.8	-2
370	Normal		'06f36a65e8a99ff89...	16:03:25	3.6	-2

Fig. 11.20 Glitches are changing the output

11.5 Conclusion

In this chapter, we performed the clock glitch fault injection attack on AES in FPGA. We injected the clock glitch in the FSM states of AES encryption. It gives concrete evidence to support the vulnerabilities of binary encoding schemes using AES. This configuration is applicable to a number of other targets, such as the implementation of additional hardware cores (ECC, SHA, SoC, etc.). It also supports the idea that a clock glitch is an easy-to-understand and implement adversarial device that can increase the vulnerability of any digital system that employs an FSM that has not been vetted for security. By end of this chapter, readers will understand how to perform practical fault attacks using clock glitches and exploit the FSM vulnerabilities.

References

1. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms (2022). arXiv preprint arXiv:2204.07909
2. Anandakumar, N.N., Dillibabu, S.: Correlation power analysis attack of AES on FPGA using customized communication protocol. In: Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology, CCSEIT '12, pp. 683–688 (2012)
3. Anandakumar, N.N., Das, M.P.L., Sanadhya, S.K., Hashmi, M.S.: Reconfigurable hardware architecture for authenticated key agreement protocol over binary edwards curve. ACM Trans. Reconfigurable Technol. Syst. **11**(2), 1–19 (2018)
4. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Depend. Secure Comput. **1**(1), 11–33 (2004)
5. Baksi, A., Bhasin, S., Breier, J., Jap, D., Saha, D.: A survey on fault attacks on symmetric key cryptosystems. ACM Comput. Surv. **55**(4), 1–3 (2022)
6. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: theory, practice, and countermeasures. Proc. IEEE **100**(11), 3056–3076 (2012)
7. Bhunia, S., Tehranipoor, M.: Hardware security: a hands-on learning approach. Morgan Kaufmann, Los Altos (2018)
8. Blömer, J., Seifert, J.P.: Fault based cryptanalysis of the advanced encryption standard (AES). In: International Conference on Financial Cryptography, pp. 162–181. Springer, Berlin (2003)
9. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. J. Cryptol. **14**(2), 101–119 (2001)
10. Dey, S., Park, J., Pundir, N., Saha, D., Shuvo, A.M., Mehta, D., Asadi, N., Rahman, F., Farahmandi, F., Tehranipoor, M.: Secure physical design (2022). Cryptology ePrint Archive
11. Dusart, P., Letourneux, G., Vivolo, O.: Differential Fault Analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) Applied Cryptography and Network Security, pp. 293–306. Springer, Berlin (2003)
12. Dworkin, M.J., Barker, E.B., Nechvatal, J.R., Foti, J., Bassham, L.E., Roback, E., Dray Jr, J.F.: Advanced Encryption Standard (AES). FIPS PUB 197 (2001). https://www.nist.gov/publications/advanced-encryption-standard-aes
13. Govindavajhala, S., Appel, A.W.: Using memory errors to attack a virtual machine. In: 2003 Symposium on Security and Privacy, 2003, pp. 154–165. IEEE, Piscataway (2003)

14. Hutter, M., Schmidt, J.M., Plos, T.: Contact-based fault injections and power analysis on RFID tags. In: 2009 European Conference on Circuit Theory and Design, pp. 409–412. IEEE, Piscataway (2009)
15. Karpovsky, M., Taubin, A.: New class of nonlinear systematic error detecting codes. IEEE Trans. Inform. Theory **50**(8), 1818–1819 (2004)
16. Kim, C.H.: Improved differential fault analysis on AES key schedule. IEEE Trans. Inform. Forensics Secur. **7**(1), 41–50 (2012). https://doi.org/10.1109/TIFS.2011.2161289
17. Kune, D.F., Backes, J., Clark, S.S., Kramer, D., Reynolds, M., Fu, K., Kim, Y., Xu, W.: Ghost talk: mitigating EMI signal injection attacks against analog sensors. In: 2013 IEEE Symposium on Security and Privacy, pp. 145–159. IEEE, Piscataway (2013)
18. Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E.: Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In: 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 77–88 (2013). https://doi.org/10.1109/FDTC.2013.9
19. Nahiyan, A., Farahmandi, F., Mishra, P., Forte, D., Tehranipoor, M.M.: Security-aware FSM design flow for identifying and mitigating vulnerabilities to fault attacks. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **38**(6), 1003–1016 (2019)
20. Shuvo, A.M., Pundir, N., Park, J., Farahmandi, F., Tehranipoor, M.: LDTFI: layout-aware timing fault-injection attack assessment against differential fault analysis. In: 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 134–139. IEEE, Piscataway (2022)
21. Song, L., Hu, L.: Differential fault attack on the PRINCE block cipher. In: International Workshop on Lightweight Cryptography for Security and Privacy, pp. 43–54. Springer, Berlin (2013)
22. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
23. Wang, H., Li, H., Rahman, F., Tehranipoor, M.M., Farahmandi, F.: Sofi: security property-driven vulnerability assessments of ICs against fault-injection attacks. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **41**(3), 452–465 (2021)

Chapter 12
Voltage Glitch Attack on an FPGA AES Implementation

12.1 Introduction

Side-channel attacks (SCAs) are considered among the most powerful physical attacks against embedded hardware devices like smartcards, FPGAs, and ASICs. There exist two classes of side-channel attacks, namely, active and passive [31]. Passive attacks exploit information that is spontaneously leaked by the device such as timing information [22], electromagnetic emissions [18], power consumption [13], or even acoustic emanations [5]. Active attacks, such as false injection attacks (FIAs), instead infect the system with an external or internal trigger. FIA has proven to be a powerful technique for uncovering confidential data with a limited number of experiments [25, 28]. A number of goals can be achieved by intruders using FIAs. There are two primary goals among these: (1) causing false outputs and in some cases, disrupting normal behavior by avoiding specific activities [34] and (2) extracting confidential information using faulty outputs [25].

Fault injection attacks can be further divided into three categories: noninvasive, invasive, and semi-invasive [9]. An integrated circuit (IC) is de-packaged during invasive attacks, and its physical characteristics are changed to enable specific examinations. Invasive attacks, such as laser, microprobing, and optical fault injection, are particularly effective at precisely positioning themselves in space time and can provide attackers access to a lot of sensitive information stored inside the device [1, 10]. These kinds of attacks are expensive and require sophisticated equipment and knowledgeable attackers. Noninvasive attacks are usually low cost and involve observations of the device's operation or manipulations of external signals. They require only moderately sophisticated equipment and knowledge to implement [33]. They do not physically harm the chip and often leave no trace. Altering the device clock signal and/or supply voltage, also known as clock and voltage glitch attacks [14, 20], is one of the most basic noninvasive fault attacks. For embedded devices in particular, noninvasive attacks pose a greater threat than invasive ones. This is due to three key factors: (1) the owner of the targeted

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_12

device might not be aware of the attack and continue to rely on its functionality and information security because this type of attack does not require any physical tampering, (2) even in a small laboratory, they can be updated and replicated using existing, affordable equipment [21, 24], and (3) they have demonstrated that a high success rate can be attained quickly [24]. According to Guillen, a sort of FIA that falls between invasive and noninvasive attacks is the semi-invasive attack. Between invasive and noninvasive attacks, the semi-invasive attack (FIA) occupies a middle position [19]. In this type of attack, the chip still needs to be de-packaged, but the internal structure remains intact. Although these attacks often leave traces, in most cases the chip remains fully operational.

Over the past decade, most of the research on fault attacks has concentrated on analyzing the fault effects and developing countermeasures. By inserting faults at precise points in the algorithms, the fault injection technique has shown how to defeat cryptographic algorithms including AES [7, 16], DES [11], and RSA [6, 30]. Among these, a practical demonstration of the fault injection technique is also given in [6, 7, 30]. All of these demos are carried out on a special board created to introduce faults into the integrated computer running the encryption algorithm. For a more comprehensive overview of attacks on AES and RSA, the reader is directed to [8]. We concentrate on voltage glitches in this chapter since they don't require package modification, are inexpensive, and are accurate in delivering the desired outcomes. Using a ChipWhisperer CW305 target board, we specifically show how to perform a voltage glitch in an AES block implemented on an FPGA and show how to fail an AES execution by applying voltage glitches that cause it to produce erroneous results during the AES encryption process. The remaining chapters are structured as follows. Section 12.2 briefly discusses basic information on voltage glitching, finite state machine (FSM), and AES. Section 12.3 briefly provides the experimental setup to perform fault attack using voltage glitch. A performing voltage glitch attack steps and glitch explorer and results are given in Sect. 12.4 discusses the. Finally, conclusions are presented in Sect. 12.5.

12.2 Background

12.2.1 Voltage Glitches

A cheap form of fault injection is to tamper with a device's power supply. A possible way to provoke faulty behavior is the underpowering of the device. Since there is no precise timing, the faults provoked by such a method tend to occur uniformly throughout the computations, and the attacker must be able to successfully discard the erroneous results caused by undesired faults. Another method to affect computations performed in the device is the induction of precise high variations in a power supply. Power spikes can cause a processor to skip or misinterpret an instruction and also induce memory faults. For instance, if a processor reads a memory location at the time of a voltage spike, the wrong data

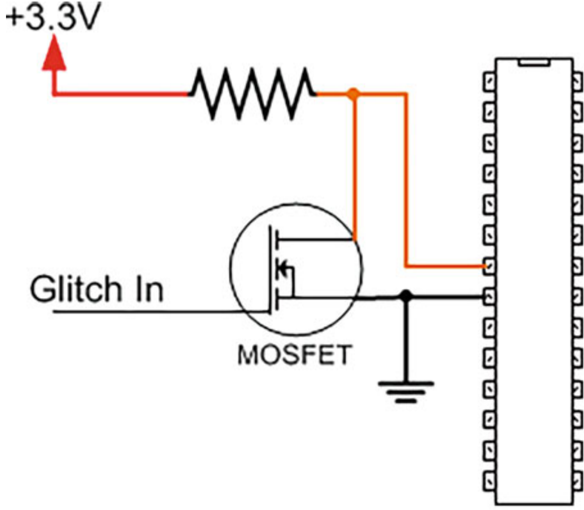


Fig. 12.1 Example of a voltage glitch implementation

may be gathered from the memory bus. Further, this fault injection technique is commonly exploited by attackers who aim to tamper with a program counter or a loop bound [4]. Both listed fault injection techniques are easy to implement and require an attacker to be able to access the power supply line of the device.

The idea behind a voltage glitch is to create a short disturbance in the power input of a certain device to affect it functionality. There are several ways to implement a voltage glitch, and the simplest and most common implementation is to connect a transistor between the positive power input (VCC) of the device and the ground pin (GND), as shown in Fig. 12.1.

The idea behind it is to decrease the voltage in the VCC line by withdrawing a considerable amount of current out of the pin. This current is going to decrease the charge present in the line for a small period of time causing a drop in the voltage. The reason why it works is that the power supplies that feed the FPGA tend to have a limited amount of current that they can provide to the power node. This power disturbance has to be short because most electronic systems have some sort of protection against voltage drops since they generally lead to unexpected outputs. If the drop in voltage is greater than a certain threshold of millivolts for a certain time, the device will trigger a reset and you will not be able to obtain information from it. These thresholds vary per device and per application. Another issue that can be encountered is disconnection, and a very low voltage can trigger a shutdown of the device too. When the voltage is lower than the usual operation range, the information will travel slower across the logic, therefore limiting the maximum frequency that the device can be operated, as shown in Fig. 12.2. At regular operation, the device will be in the “Safe Operating Area.” With a quick drop in voltage, for a short period of time, our device could be operating outside that area, therefore being susceptible to faults. With the voltage glitch, we will attempt to recreate a similar type of fault

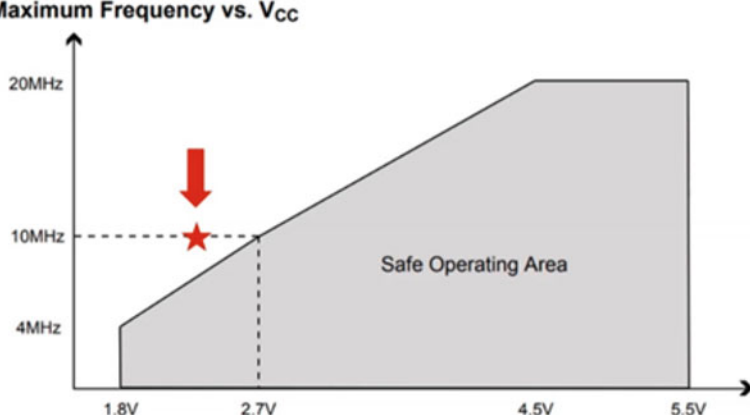


Fig. 12.2 Frequency vs. voltage safe operation area

to the one done with the clock glitch attack as discussed in Chap. 11. Ideally, we will be able to observe that some of the circuits compiled with the FSM one-hot encoding scheme will be more susceptible to faults than others. If a voltage glitch is effective, the output of the voltage glitch will be affected but will not be reset it.

12.2.2 Fault Models

Generally, the following categories of fault models are used in the analysis methods in the literature:

- **Bit flip** is the flipping of a bit, with the attacker having exact control over which bit is chosen. This category includes multiple-bit flips as long as the attacker chooses all of the target bits. For instance, the majority of fault attacks on neural networks make use of this concept [29].
- **Bit set/reset** is the process of changing a bit's value from '0' (reset) to '1' (set). This fault model can be used, for instance, in blind fault attacks and is quite strong. Once more, it is assumed that the attacker has the ability to choose which bit will be set or reset. This fault model is quite effective and can be used, for instance, in blind fault attacks [23].
- **Random byte** is a less accurate fault model in which the value of a specific byte changes to a random value. This is said to be the fault model that allows for the most successful DFA attack [26].
- **Instruction skip** ignores the currently being processed instruction's execution practically. Using this fault model, powerful attacks can be launched [12].
- **Execution faults** occur in FPGAs where the values being processed are impacted by setup violations [32].
- **Stuck-at faults** permanently transform the value of the stored data into another one. This fault model supports SIFA [15].

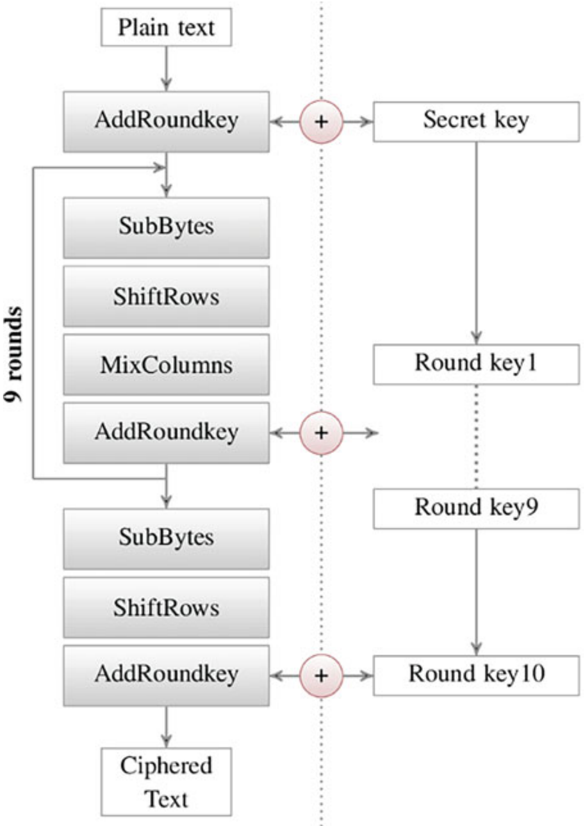


Fig. 12.3 AES encryption process

12.2.3 Brief Description of AES

AES is a standard established by the NIST [17] for symmetric key cryptography. It is a substitution and permutation network based on four transformations (i.e., SubBytes, ShiftRows, MixColumns, AddRoundKey) used iteratively in rounds as depicted in Fig. 12.3. The AES is defined for 128-bit blocks and key sizes of 128, 192, and 256 bits. It processes data blocks of 128 bits (usually represented as a 4×4 bytes matrix, called the AES state) in ten rounds (after round 0). The round keys (K1 to K10) used during every round are calculated on-the-fly by a key expansion module. Hence, full encryption is completed in 11 clock periods. In this chapter, we deal with the 128-bit AES due to its widespread usage. For a complete description and explanation of AES, please refer to [2, 17].

Fig. 12.4 AES finite state machine (FSM) with one-hot encoding

12.2.4 Voltage Glitch Attack on FSM in AES Controller

In the case of AES, this glitch could be used to cause its FSM to skip states and either provide an incorrect ciphertext or give an adversary access to the secret encryption key [27, 35]. When focusing on attacking an FSM, we must look at how that FSM is encoded to see what kind of vulnerabilities it may have. In our work, FSM for AES has the states WAIT KEY, WAIT DATA, INITIAL ROUND, DO ROUND, and FINAL ROUND and is one-hot encoding those states using the bit configurations of 00001, 00010, 00100, 01000, and 10000, respectively (see Fig. 12.4). When the FSM is moving from state WAIT KEY to WAIT DATA, or 00001 to 00010, both the most bit and least significant bit are flipping from 0 to 1 or 1 to 0. Because the flip flops for these bits may have different delays, if a voltage glitch occurs and the state is set too early within the flip flops, the user could get the state 10000 instead

of the expected 00100. This would mean the FSM was transferring directly from WAIT KEY to FINAL ROUND, theoretically giving the secret key as an output rather than the encrypted ciphertext. Therefore, the encoding of FSM states for encryption algorithms is critical in performing attacks on the FSM. In this chapter, we demonstrate how to perform fault attacks using voltage glitches and exploit the FSM vulnerabilities. In this chapter, we'll employ voltage glitches to try to break the AES execution. It is used to briefly short-circuit Vcc to ground. Then, the glitch is triggered by a field programmable gate array (FPGA) managing the attack timing which might be able to corrupt some of the bits of the FSM states and affect all 128 bits of the output.

12.2.5 ChipWhisperer CW305 Board

For the study of embedded hardware security, ChipWhisperer provides a collection of several helpful tools. There are ChipWhisperer hardware targets, ChipWhisperer target device firmware, target device FPGA blocks, ChipWhisperer analysis software and libraries, as well as ChipWhisperer-Capture devices (which sample power measurements). As a standalone target, the CW305 board enables the use of a larger FPGA target to implement cores like AES and ECC [3].

12.3 Experimental Setup

12.3.1 Hardware Setup

In this section, we will use the ChipWhisperer-Lite and ChipWhisperer CW305 board to perform fault attacks using voltage glitches. The setup for the experiment is shown in Fig. 12.5. To set up the hardware for voltage glitching, only one extra connection is required compared to the setup for a clock glitch attack (see Chap. 11). Note that the original SMA cable (connected to the ChipWhisperer's measure input) is not required for voltage glitching—if we only have one cable, we can just move it over. However, it is helpful to have power traces to see what effects the voltage glitches are having on the power rails, so if we can connect both of them.

To perform a voltage glitch attack, we need to have access to the pin that provides power to the FPGA. Luckily, the CW305 has an exposed connection on the top of the board where we can connect the SMA from the glitch port of the ChipWhisperer-Lite; that SMA port is identified with X3. To observe the effect of our glitch, we need to connect the SMA cable that comes from the MEASURE port to the port called X4 in the CW305 FPGA. That port will amplify the difference of voltage across a resistance that the power comes through. That difference in voltage, combined with the known value of the resistance, will allow us to observe the current that is going to the device. For the J16, K16, K15, and L14 switches, the configuration is 0, 1, 1, and

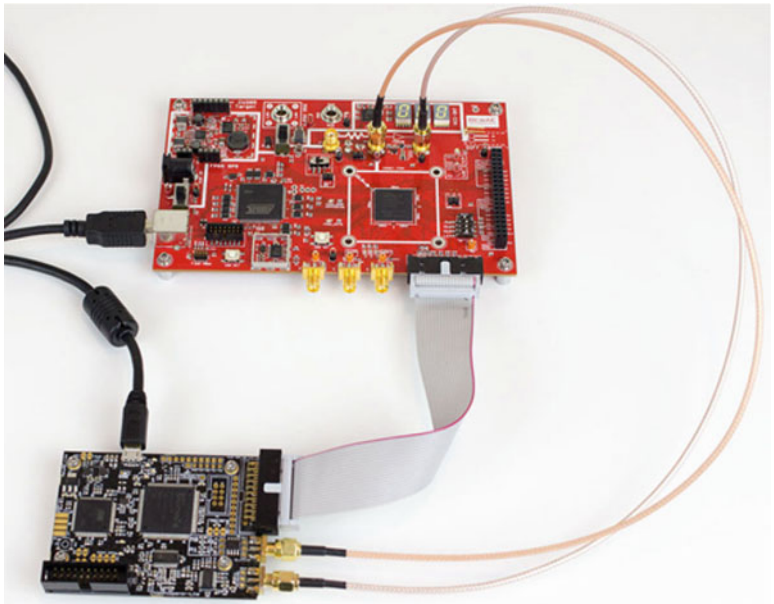


Fig. 12.5 The CW305 interconnected to the CW-Lite board (voltage glitch setup)

1, respectively. That will allow the FPGA to use the generated clock instead of the one that could be provided from the CLKIN (clock-in) port. The 20-pin connection from the ChipWhisperer Lite to the FPGA will be used as an interface to transmit the data between the AES implementation and the computer. The rest of the circuitry supports the objective of performing fault attacks on any target algorithm. Please refer to Chap. 8 for firmware setup and ChipWhisperer board configuration and how to connect a ChipWhisperer-Lite to the CW305 board.

12.3.2 Software Setup

First, we need to download the ChipWhisperer software V4.0.1 from the following link: https://github.com/newaetech/chipwhisperer/releases and install it on the control computer (PC). The ChipWhisperer Python library can be used to communicate with the NewAE Capture and target boards. Once the ChipWhisperer software is installed, we need to install the Xilinx Vivado software for AES bitstream generation. How to download and install the Xilinx Vivado Design Suite is discussed in Chap. 8. For this tutorial, we will provide a pre-existing AES-128 encryption example with a couple of project files to build a project using the Xilinx Vivado software. There are three steps that Vivado takes to turn our Verilog into a bitstream code: (1) synthesis in which the Verilog code is synthesized into a gate-level

representation, (2) implementation in which the synthesized logic is routed to fit onto the device, and (3) generate the bitstream. Once the bitstream is generated, we need to identify its path. It is usually stored in the project file folder. We need its directory to insert it in the code. The CW305 Artix FPGA target and ChipWhisperer-Lite were used to load the AES algorithm and inject clock glitches reliably. Moreover, all Verilog design files and all Python sources can be found at http://cad4security.org/index.php/trainings/hsl/ch12_voltage_glitch_fia/.

12.4 Performing Voltage Glitch Attacks

12.4.1 Steps in Performing Voltage Glitch Attacks

1. First, open the ChipWhisperer capture software from the toolbar below and choose the Python console. In the middle window, we can see the files in chipwhisperer/software/chipwhisperer/capture/scripts directory.
2. This implementation is quite similar to the clock glitch (see Chap. 8). First, we run aaVFI_Setup.py. Next, we must click on the mater button as shown in Fig. 12.6. All three buttons should turn green at this point.
3. Then, we run the aaVFI_Setup2.py and use the glitch explorer tool to see the outputs and plot them. Finally, we run aaVFI_Sweep_offset.py for the glitch exploration and run it for traces. If in the debug logging we receive a message that "FPGA Bitstream not configured or "not a file," browse the device to upload the bitstream file under FPGA bitstream and configure it by manually clicking on the program FPGA button as shown in Fig. 12.7. We should see there is no

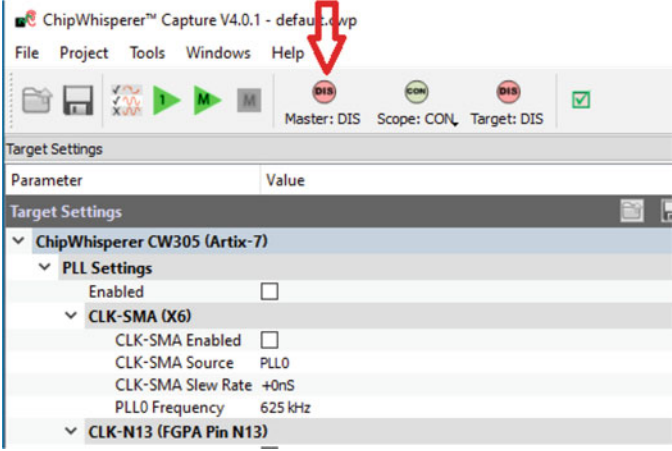


Fig. 12.6 Set up ChipWhisperer-Lite with cw305 FPGA board

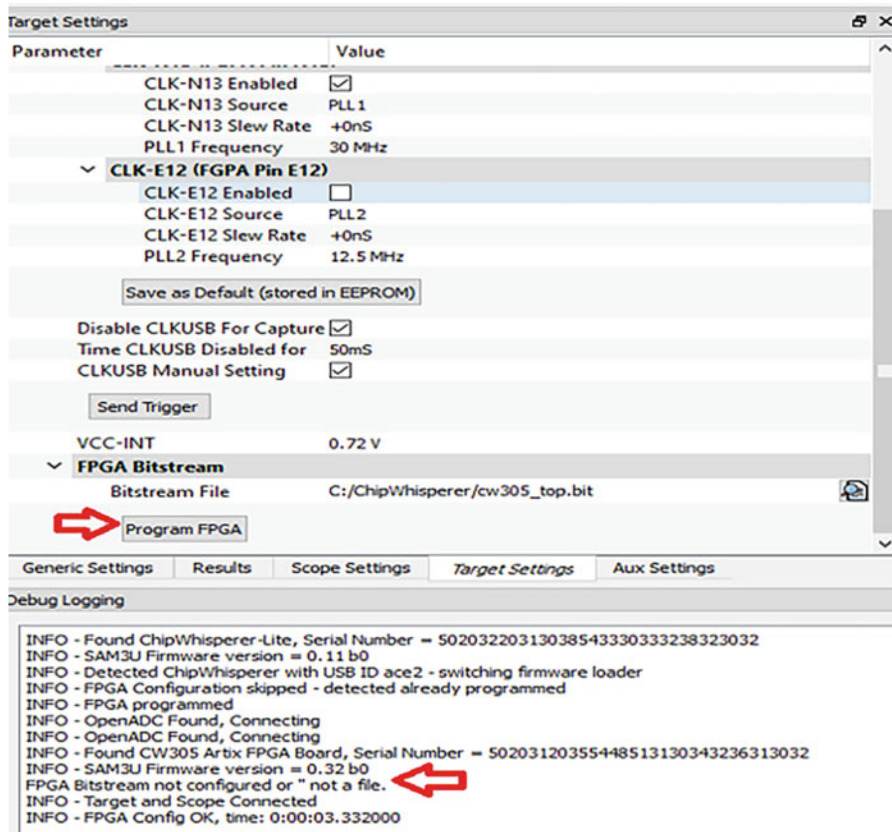


Fig. 12.7 FPGA bitstream configuration

problem with the Python code running. Then we can also check the log to see if ChipWhisperer and FPGA board are connected and if FPGA is programmed.

4. Once everything is set up, we will be able to observe a power waveform similar to Fig. 12.8. In it, we will find a series of peaks, each one related to one operation of the encryption process. Knowing the meaning of each peak will help us position our glitch in the correct area to extract information. In Fig. 12.8, the L is when the information is getting loaded to the FPGA, K is when the key start to arrive at the encryption, D is when the data arrives to the encryption, I is the initial round, 2–9 are the middle rounds, F the final, and E is the part where the data is exported out. Moreover, each peak only lasts one clock cycle.
5. Using that information, we can move the Ext. Offset parameter and target any of the rounds, depending on the attack that we want to execute. Other parameters that can be adjusted to make our glitch more effective are the glitch width, offset, and the number of repeats as shown in Fig. 12.9.

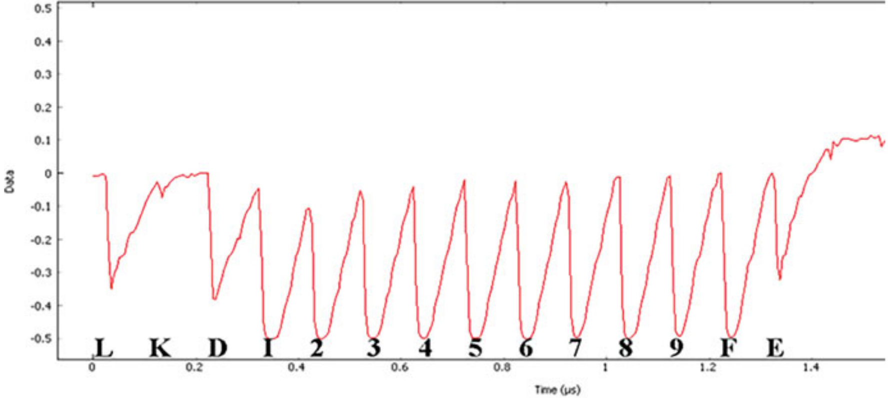


Fig. 12.8 Power waveform for AES encryption process

Glitch Module	
Clock Source	CLKGEN
Glitch Width (as % of period)	25
Glitch Width (fine adjust)	0
Glitch Offset (as % of period)	0
Glitch Offset (fine adjust)	0
Glitch Trigger	Ext Trigger:Single-Shot
Single-Shot Arm	After Scope Arm
Ext Trigger Offset	0
Repeat	1
Manual Trigger / Single-Shot Arm	
Output Mode	Glitch Only
Read Status	

Fig. 12.9 Set up the glitch module

6. The width will adjust the duration of our glitch, it can be adjusted up to 50% of a clock cycle, and there's also a fine adjustment in case it is needed. There are two kinds of offset, the Glitch Offset and the Ext Trigger Offset as shown in Fig. 12.9. The first one moves our glitch across one clock cycle and the latter moves our glitch across different clock cycles. The number of repeats changes the number of times the glitch gets activated after it is triggered, meaning that for the next number of cycles, a glitch is going to show up in the waveform.
7. Other parameters that can be adjusted are the ones that define the point of operation of the FPGA. These are FPGA frequency and FPGA voltage. They are marked in Fig. 12.10 with a red dot, and they can be found in the target settings tab.

Fig. 12.10 Target settings tab

8. Using the provided script: aaVFI_Sweep_offset.py, we will be able to sweep two of the parameters to obtain results from different parameters to evaluate our results. While sweeping the parameters, we will see the result showing up in the Glitch Map module as shown in Fig. 12.9.

12.4.2 Starting the Voltage Glitch Attack

The goal is to identify a set of voltage glitch parameters that result in the encryption process failing once we get voltage glitches operating. Let's configure the glitch explorer to automatically look for glitches so that we may accomplish this. Getting our FPGA to use a set plaintext and key is the first thing we must do. It will be more challenging to determine when a glitch was successful if we modify the inputs on every capture. The fixed key 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C and fixed plaintext 5C 69 2F 91 03 B2 30 29 14 D7 E5 55 E4 DC EE 49 will be used for this tutorial; however, we can use any key and plaintext. Because of the alterations to our AES implementation outputting the states rather than the encrypted output, the ideal output for the binary AES encoding is "1234056789abcdef0000000000000000". The portion of this output that is relevant is the first five bits: 12340. Set the normal/successful answers in the glitch explorer to look for this string. Then, as illustrated in Fig. 12.9, set up two tuning parameters to sweep the width and offset of the glitch module. Important note: If our version

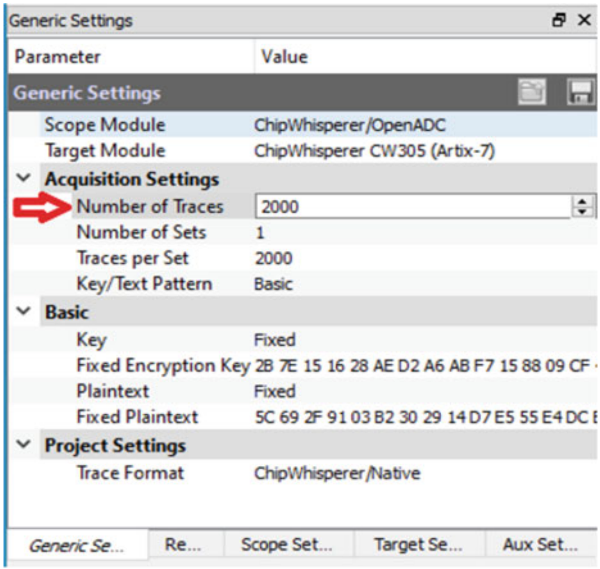


Fig. 12.11 Generic settings

of the ChipWhisperer GUI does not show tuning parameters, we must use the given Python code to sweep the glitch width and offset aaVFI_Sweep_offset.py and run it. This will be visible in the GUI under the Python coding examples and can be edited as needed to fit the described parameters. If you need help with what parameters to edit, this link is useful for the syntax to use to change parts of the GUI to fit our needs in Python: https://chipwhisperer.readthedocs.io/en/latest/api.html. Once everything is set up, either tuning parameters on the GUI or the relevant sweeping code has been run, we can now start capturing. Note that we need to set the number of traces to 2000 before capturing as shown in Fig. 12.11.

12.4.3 Results

Once our capture is complete, we can verify the output of the glitch explorer: examining the glitched ciphertexts reveals a wide range of various outputs with various glitch settings (see Fig. 12.12). We have experimentally demonstrated that the AES execution by applying voltage glitches caused it to produce erroneous results during the AES encryption process. Note: we must clarify in the glitch terminal what is deemed as "Normal" and a "Success". In our case, "Normal" can be equal to the expected ciphertext and "Success" can be anything not equal to that value. We have done our goal which was to set up voltage glitching the CW305 and determine exactly what effect our glitches had on this AES setup.

Glitch Explorer

	Status	Sent	Received	Date	Glitch Width	Glitch Offset
1080	Normal		'1234056789ab...	00:51:51	1.953125	-0.390625
1081	Normal		'1234056789ab...	00:51:51	1.953125	-0.78125
1082	Normal		'1234056789ab...	00:51:51	1.953125	-1.171875
1083	Success		'1034056789ab...	00:51:50	1.953125	-1.5625
1084	Normal		'1234056789ab...	00:51:50	1.953125	-1.953125
1085	Normal		'1234056789ab...	00:51:50	1.953125	-2.34375
1086	Normal		'1234056789ab...	00:51:49	1.953125	-2.734375
1087	Normal		'1234056789ab...	00:51:49	1.953125	-3.125
1088	Normal		'1234056789ab...	00:51:48	1.953125	-3.515625
1089	Normal		'1234056789ab...	00:51:48	1.953125	-3.90625

Parameter	Value
Glitch Explorer	
Clear Output Table	
Plot Widget	
Normal Response	s == '1234056789abcdef0000000000000000'
Successful Response	s != '1234056789abcdef0000000000000000'
Recordings	

Total 2000, Glitches Successful 5

Fig. 12.12 Glitches are changing the output

12.5 Conclusion

In this chapter, we performed the voltage glitch fault injection attack on AES in FPGA. We injected the voltage glitch in the FSM states of AES encryption. It gives concrete evidence to support the vulnerabilities of a one-hot encoding scheme on AES. By the end of this chapter, readers will understand how to set up voltage glitching the CW305 and determine exactly what effect our glitches had on AES and exploit the FSM vulnerabilities.

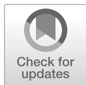
References

1. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms (2022). arXiv preprint arXiv:2204.07909

2. Anandakumar, N.N., Dillibabu, S.: Correlation power analysis attack of AES on FPGA using customized communication protocol. In: Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology, CCSEIT '12, pp. 683–688 (2012)
3. Anandakumar, N.N., Das, M.P.L., Sanadhya, S.K., Hashmi, M.S.: Reconfigurable hardware architecture for authenticated key agreement protocol over binary edwards curve. ACM Trans. Reconfigurable Technol. Syst. **11**(2), 1–19 (2018)
4. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.P.: Fault attacks on RSA with CRT: concrete results and practical countermeasures. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 260–275. Springer, Berlin (2002)
5. Backes, M., Dürmuth, M., Gerling, S., Pinkal, M., Sporleder, C., et al.: Acoustic {side-channel} attacks on printers. In: 19th USENIX Security Symposium (USENIX Security 10) (2010)
6. Barenghi, A., Bertoni, G., Parrinello, E., Pelosi, G.: Low voltage fault attacks on the RSA cryptosystem. In: 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 23–31. IEEE, Piscataway (2009)
7. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pellicioli, M., Pelosi, G.: Fault attack on AES with single-bit induced faults. In: 2010 Sixth International Conference on Information Assurance and Security, pp. 167–172. IEEE, Piscataway (2010)
8. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: theory, practice, and countermeasures. Proc. IEEE **100**(11), 3056–3076 (2012)
9. Beringuier-Boher, N., Gomina, K., Hely, D., Rigaud, J.B., Beroulle, V., Tria, A., Damiens, J., Gendrier, P., Candelier, P.: Voltage glitch attacks on mixed-signal systems. In: 2014 17th Euromicro Conference on Digital System Design, pp. 379–386. IEEE, Piscataway (2014)
10. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Los Altos (2018)
11. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Annual International Cryptology Conference, pp. 513–525. Springer, Berlin (1997)
12. Breier, J., Jap, D., Chen, C.N.: Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on AES. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, pp. 99–103 (2015)
13. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 16–29. Springer, Berlin (2004)
14. Dey, S., Park, J., Pundir, N., Saha, D., Shuvo, A.M., Mehta, D., Asadi, N., Rahman, F., Farahmandi, F., Tehranipoor, M.: Secure physical design (2022). Cryptology ePrint Archive
15. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embedded Syst. **2018**(3), 547–572 (2018)
16. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) Applied Cryptography and Network Security, pp. 293–306. Springer, Berlin (2003)
17. Dworkin, M.J., Barker, E.B., Nechvatal, J.R., Foti, J., Bassham, L.E., Roback, E., Dray Jr, J.F.: Advanced Encryption Standard (AES). FIPS PUB 197 (2001). https://www.nist.gov/publications/advanced-encryption-standard-aes
18. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 251–261. Springer, Berlin (2001)
19. Guillen, O.M., Gruber, M., Santis, F.D.: Low-cost setup for localized semi-invasive optical fault injection attacks. In: International Workshop on Constructive Side-Channel Analysis and Secure Design, pp. 207–222. Springer, Berlin (2017)
20. Karaklajić, D., Schmidt, J.M., Verbauwhede, I.: Hardware designer's guide to fault attacks. IEEE Trans. Very Large Scale Integr. Syst. **21**(12), 2295–2306 (2013)
21. Kazemi, Z., Papadimitriou, A., Souvatzoglou, I., Aerabi, E., Ahmed, M.M., Hely, D., Beroulle, V.: On a low cost fault injection framework for security assessment of cyber-physical systems:

clock glitch attacks. In: 2019 IEEE 4th International Verification and Security Workshop (IVSW), pp. 7–12. IEEE, Piscataway (2019)
22. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Annual International Cryptology Conference, pp. 104–113. Springer, Berlin (1996)
23. Korkikian, R., Pelissier, S., Naccache, D.: Blind fault attack against SPN ciphers. In: 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 94–103. IEEE, Piscataway (2014)
24. Li, Y., Chen, M., Wang, J.: Introduction to side-channel attacks and fault attacks. In: 2016 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC), vol. 1, pp. 573–575. IEEE, Piscataway (2016)
25. Liao, N., Cui, X., Liao, K., Wang, T., Yu, D., Cui, X.: Improving DFA attacks on AES with unknown and random faults. Sci. China Inform. Sci. **60**(4), 1–14 (2017)
26. Luo, P., Fei, Y., Zhang, L., Ding, A.A.: Differential fault analysis of SHA3-224 and SHA3-256. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 4–15. IEEE, Piscataway (2016)
27. Nahiyan, A., Xiao, K., Yang, K., Jin, Y., Forte, D., Tehranipoor, M.: AVFSM: a framework for identifying and mitigating vulnerabilities in FSMs. In: Proceedings of the 53rd Annual Design Automation Conference, pp. 1–6 (2016)
28. Piscitelli, R., Bhasin, S., Regazzoni, F.: Fault attacks, injection techniques and tools for simulation. In: Hardware Security and Trust, pp. 27–47. Springer, Berlin (2017)
29. Rakin, A.S., He, Z., Li, J., Yao, F., Chakrabarti, C., Fan, D.: T-BFA: targeted bit-flip adversarial weight attack. IEEE Trans. Pattern Anal. Mach. Intell. **44**(11), 7928–7939 (2021)
30. Schmidt, J.M., Herbst, C.: A practical fault attack on square and multiply. In: 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 53–58. IEEE, Piscataway (2008)
31. Spreitzer, R., Moonsamy, V., Korak, T., Mangard, S.: Systematic classification of side-channel attacks: a case study for mobile devices. IEEE Commun. Surv. Tutorials **20**(1), 465–488 (2017)
32. Tajik, S., Lohrke, H., Ganji, F., Seifert, J.P., Boit, C.: Laser fault attack on physically unclonable functions. In: 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 85–96. IEEE, Piscataway (2015)
33. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
34. Timmers, N., Mune, C.: Escalating privileges in linux using voltage fault injection. In: 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 1–8. IEEE, Piscataway (2017)
35. Wang, H., Li, H., Rahman, F., Tehranipoor, M.M., Farahmandi, F.: SOFI: security property-driven vulnerability assessments of ICs against fault-injection attacks. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **41**(3), 452–465 (2021)

Chapter 13
Laser Fault Injection Attack (FIA)



13.1 Introduction

Fault injection attacks (FIA) and respective countermeasures are one of the main research arenas in the hardware security domain. Fault injections are a set of active physical attacks that can be used by an adversary to make a target device not function correctly, have unauthorized access, leak assets, and sometimes, decipher the architecture of the whole device [1]. There are different types of fault injection attacks prevalent among attackers and the research community. Laser fault injection (LFI) is one of the types. Fault injection by laser has certain advantages over the other counterparts like timing and voltage fault injections. Timing fault injection and voltage fault injection usually affect the device globally. So, an attacker cannot achieve spatial precision with these attacks. On the other hand, a laser enables an attacker to inject faults into a very specific point of interest in the target device with very high precision. But such a laser setup is very costly and has a very exhaustive search space in the experiments. This chapter basically deals with a detailed experimental setup for laser fault injections from a practical perspective.

A system reacts uniquely to different fault injections. Fault propagation creates an individual response to a system. The reaction of a system is analyzed in the corresponding literature in detail [5, 16, 17, 28]. Fault injection works described in the literature deal with diverse faults, their effects, detailed injection techniques, and countermeasures. Timing, voltage, optical, and electromagnetic are some of the faults frequent among attackers and the research community [6, 22]. Fault injections in a device can create catastrophic consequences. These consequences range from denial of service (DOS) attacks to instruction skipping, authentication bypass, and breaking cryptographic algorithms [10, 23]. Countermeasures to these fault injections also have been described in corresponding literature including various security property-driven [18, 20, 26] and optical probing-based countermeasures [21, 25].

Fault injection attack requires test pattern generation. Test patterns can be generated electrically or externally. External stimuli can be generated by photons of

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_13

a laser beam or by electromagnetic stress. Laser beams or electromagnetic stress are applied on a device under test (DUT). As DUT, a rudimentary circuit implemented on a flip-chip FPGA is used. FPGAs are gaining vast popularity among designers, attackers, and researchers due to fast development, reasonable computing resource at a competitive price, fair controllability, and observability of these resources. In this book chapter, a laser beam is shot into an FPGA, and corresponding effects are observed.

The rest of the chapter is organized as follows: Sect. 13.2 provides background on fault injection, especially laser fault injection in detail. The complete setup is described in Sect. 13.3. Section 13.4 deals with data collection and the result analysis. Finally, Sect. 13.5 concludes this chapter.

13.2 Laser Fault Injection Attacks

This section deals with the theoretical background of laser fault injection and attack scenarios. Fault can be induced by invasive, noninvasive, and semi-invasive manners. Laser fault injection is a type of optical fault injection attack that falls under the semi-invasive or noninvasive categories. In a semi-invasive attack, we need to do sample preparation like decapsulation, circumvention of shielding, and passivation layer removal. If the target IC has a bare die packaging, then no sample preparation is required and, in that case, the attack is noninvasive.

13.2.1 Analysis of Laser Beams on MOSFETs

When a MOSFET is irradiated by a laser beam with sufficient energy, electron-hole pair is generated. These electron-hole pairs will recombine if there is no external force. When the MOSFET is biased with proper voltages on the gates, there will be a strong electric field in the MOSFET [19]. The electric field will prevent the electron-hole pair recombination and keep them moving. So transient current will start to flow [9], as shown in Fig. 13.1.

When the generated electron-hole pair propagates to the depletion region, the electron-hole pairs are rapidly collected by the electric field and create a current/voltage transient at that node. This collection is followed by the diffusion process [8], where additional charges are gathered by diffusion, as shown in Fig. 13.1c. The current induced by such carrier propagation mechanism is called "photocurrent" [15, 27], which is depicted in Fig. 13.1d.

As per our discussion, we see that a strong electric field is needed for maintaining electron-hole pair segregation [13]. The strongest electric field is created in case of a reverse bias. So areas with reverse bias are the most laser-sensitive part of a CMOS. For the nMOS transistor, the drain is reverse biased with $P_substrate$. This area is

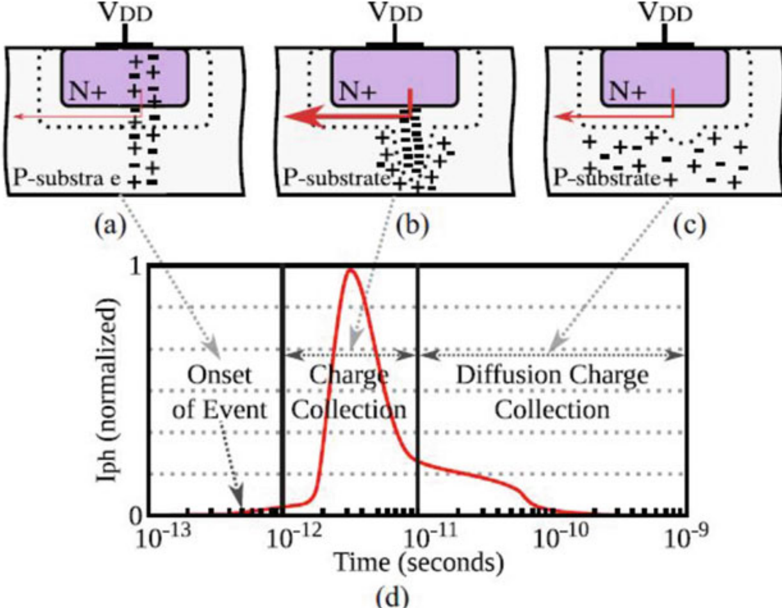


Fig. 13.1 Generation of transient photocurrent by laser beam [24]. (**a**) Generation of electron-hole pair, (**b**) propagation of generated carriers, (**c**) carrier diffusion, (**d**) transient photocurrent in different layers

the most vulnerable to laser attacks. The drain is also the most vulnerable part for laser attack for pMOS by the same reasoning.

The transient voltage, caused by the carrier propagation, can create a bit flip. Errors introduced by such a mechanism are called soft errors. Soft errors can be of two types: single-event upset (SEU) and single-event upset (SET). Single-event upset is created by a laser beam hitting on the sequential element, whereas single-event transient (SET) is created by a laser beam hitting on combinational circuits. As we can see in Fig. 13.2, the transient voltage can move through the circuit to a sequential element, and the wrong logical value can be captured by the sequential element.

13.2.2 Exploitation of Laser Attacks

Creating faults by the laser beam can have different effects in terms of security breaches. This section of the book chapter will feature a few of them. Password bypassing is one of them. When an attacker attacks authentication results by checking resources with a laser beam, the resource value can flip resulting in more privileged access. The adversary can have access to secured data and, sometimes,

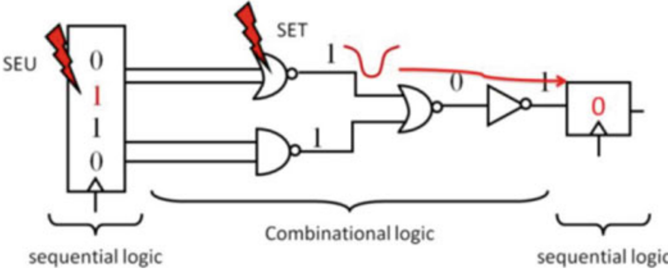


Fig. 13.2 Different types of soft errors and error propagation [7]

the architecture of the whole device. Memory dump is another kind of attack that is performed by the adversary. The adversary shoots the laser beam at the target device during data transmission. Because of this attack, data can be transmitted from the wrong location. Sometimes redundant data can be dumped. In the utmost case, the whole memory can be dumped.

Differential fault analysis (DFA) is used by the attacker for breaking the cryptographic algorithm. The objective of these attacks is to extract the key. The attacker attacks with a laser beam, and after a successful bit flip, they extract the wrong cipher text. The wrong cipher text then can be used to extract the key with mathematical reasoning [11]. Sometimes, the attackers use fault injection to bolster side-channel attacks by disabling the countermeasures [2]. If the countermeasure is configurable, it is possible for an attacker to disable them.

13.3 Device Under Test (DUT) Circuit on FPGA

This section describes the fundamentals of FPGAs and the type of FPGA which is usually used for laser fault injection. This section also characterizes the circuit that is used as the design under test (DUT) for this laser fault injection.

13.3.1 Field Programmable Logic Arrays

Application-specific integrated circuits (ASICs) are fixed hardware circuits and are very costly as well. FPGAs provide reconfigurability with decent performance. An FPGA fabric is composed of an array of configurable logic blocks, connected by programmable interconnections [3]. CLBs are the basic building blocks that can implement both combinational and sequential logic. The connection box and switch box are the hardware components that provide reconfigurable connections between fixed routing wires of the CLBs. For implementing memory elements, dedicated RAM components are used. Xilinx, one major vendor of FPGA, refer to these RAMs

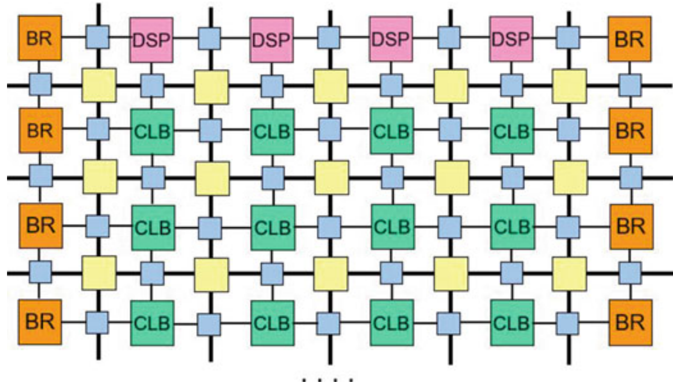


Fig. 13.3 FPGA fabric with two-dimensional arrays of CLBs, DSPs, block RAMs, and programmable interconnects

as block RAMs (BRAMs), whereas Intel, another major vendor of FPGAs, calls them memory blocks. Modern FPGAs also provide DSP slices that can implement multiplication, MAC circuits, Barrel shifter, FIR filters, etc. more efficiently. A typical modern FPGA contains thousands of CLBs, DSPs, BRAMs, and hundreds of I/O ports. A two-dimensional FPGA fabric is illustrated in Fig. 13.3.

Mostly, two types of packaging are available for FPGAs: wire bond packaging and flip chip packaging. In wire bond packaging, bonded wires are used to connect the metal layers of the IC with the PCB board, whereas in the flip chip, the IC is flipped and the metal layers are directly connected to the PCB. For laser fault injection attacks, flip-chip FPGAs are preferred. Because in modern technology nodes, there are as high as ten metal layers and these metal layers prohibit laser beams. In flip chip packaging, as the IC is flipped, silicon bulk is exposed and laser beams with certain wavelengths (1054 nm) can penetrate the bulk silicon and flip the bit value. For this experiment, the Artix-7 AC701 development board from Xilinx is used. This development board has a 7 series Artix FPGA with 28 nm technology node. This specific board is used for this experiment because the Artix-7 FPGA is a bare-die flip chip. So, no pre-processing like decapsulation is needed.

13.3.2 Device Under Test (DUT)

A very rudimentary circuit was designed to be used as the device under test (DUT). It is a simple finite state machine, as shown in Fig. 13.4. One thousand registers are set to zero value. In the Check state, the laser is fired until there is a non-zero value for the register. A non-zero value will imply that Once there is a non-zero value. Once the register has a non-zero value, the finite state machine will go into the done state, where this change in the register value will be observed.

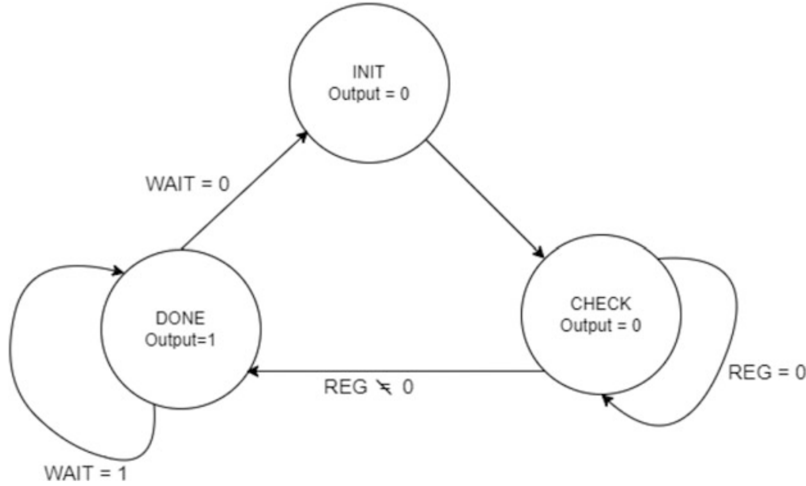


Fig. 13.4 Device under test for laser fault injection

13.4 Experimental Setup

In this section, we will discuss the hardware setup for laser fault injection. We will use 1064 nm diode laser in this experiment.

13.4.1 Hardware and Software

The following hardware and software setup was used for the implemented attack:

- Laptop/PC (i.e., installed Windows 10)
- Xilinx Vivado software: The design under test (DUT) device is developed using Xilinx Vivado 2021.2
- Laser: A 1064 nm diode laser is used for laser shooting. Laser power, pulse duration, and time between consecutive triggering can be controlled by a user application.
- XYZ precision stage: A XYZ precision stage is used to hold and place the target FPGA for precision laser shooting.
- Inspector software: The Inspector software includes a GUI interface for parameter tuning and Java scripts are run here to control the diode laser station, XYZ precision placement station, NIR camera.
- Artix-7 FPGA: The DUT is implemented in the AC-701 Artix-7 Xilinx FPGA with a 28 nm technology node which has flip-chip packaging.
- Oscilloscope: An oscilloscope is used to observe the bit flip caused by laser fault injections.

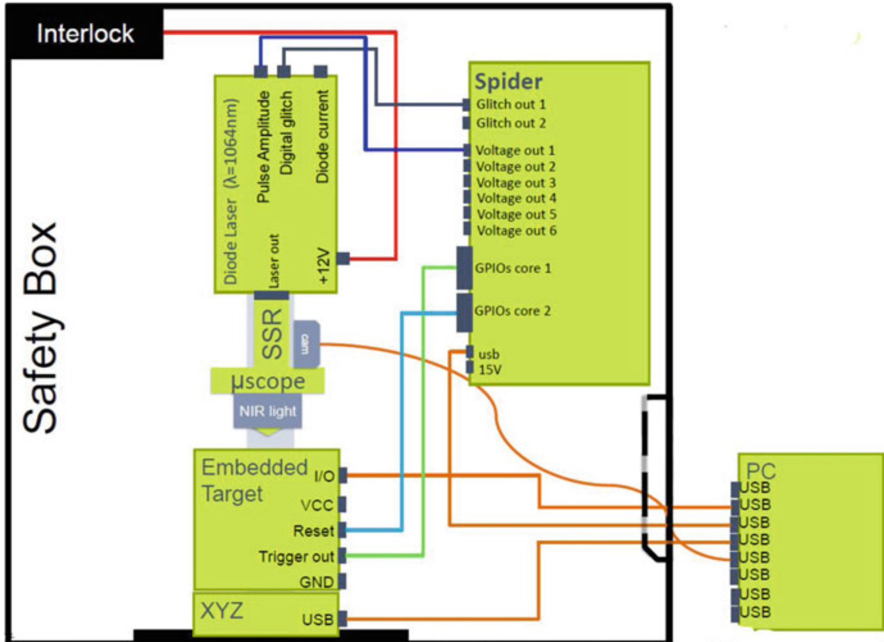


Fig. 13.5 Setup for laser fault injection [14]

13.4.2 Hardware Setup

We used Riscure laser station for housing the diode laser, NIR camera, and XYZ precision stage. A side-channel analysis workbench, Spider, is used for generating glitches and receiving trigger and reset signals from the embedded DUT. XYZ precision stage can be controlled by either the Inspector GUI or the attached wired controller. An oscilloscope will be used for observing the faults. The setup for the experiment is shown in Fig. 13.5.

The following subsections will briefly describe the main equipment for laser fault injection attack.

13.4.2.1 Diode Laser

For this experiment, a Riscure 1064 nm multimode laser diode [14] is used for fault injection. 1064 nm laser is used because we are using a flip chip as the target FPGA and 1064 nm laser can penetrate through bulk silicon and cause a bit flip. The laser can produce a maximum power of 30 W. In practice, up to 40–50% power is used so that it does not do any permanent damage. The laser diode is shown in Fig. 13.6.

Fig. 13.6 Riscure 1064nm diode laser [14]

Fig. 13.7 Riscure Spider tool [14]

13.4.2.2 Spider Tool

Spider tool [14], as shown in Fig. 13.7 from Riscure, is used for side-channel analysis and fault injections. It is used for the proper timing of the glitch generations. It also receives trigger signals from the target board. The spider tool can be controlled by inspector GUI.

13.4.2.3 AC701 Artix-7 Evaluation Board

The AC701 evaluation board [12] has flip-chip packaging of the Artix-7 FPGA, as shown in Fig. 13.8. Artix-7 is a 7 series Xilinx FPGA with 28 nm technology [4]. To configure correctly the switch, SW1 (marked with red box in Fig. 13.8) should have a value of 1, 0, 1.

Fig. 13.8 Target AC701 Artix-7 FPGA board with flip-chip packaging

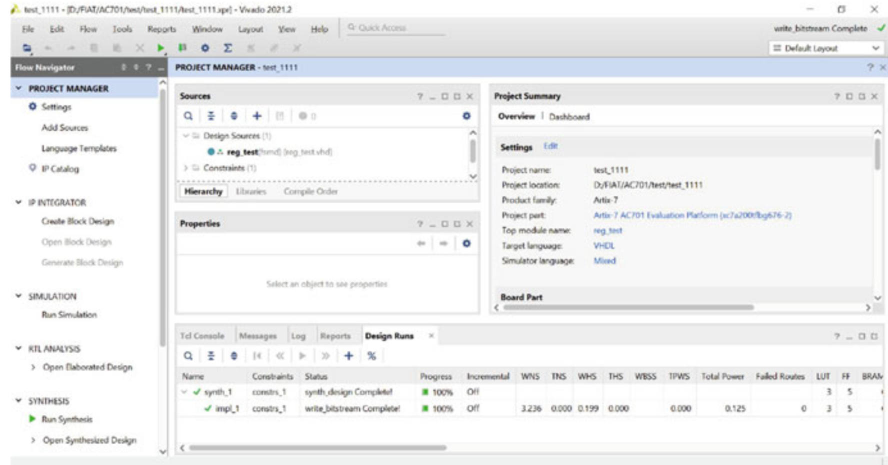


Fig. 13.9 Building the project using the Xilinx Vivado software

13.4.3 DUT Bitstream Generation

We will need Xilinx Vivado tool for DUT bitstream generation from the HDL description of the device. Moreover, all Verilog design files and sources can be found at http://cad4security.org/index.php/trainings/hsl/ch13_laser_fia/. In this experiment, we used Vivado 2021.2 version. We have to open a new project, as shown in Fig. 13.9. After that, bitstream generation is done by following the step-by-step process mentioned here.

1. Synthesis: the Verilog code is synthesized into a gate-level representation. During the synthesis stage, the HDL code composed at the design entry stage will be converted into a circuit in the form of netlist by the electronic design automation (EDA) tools. Our HDL code is going to be parsed to check syntax and then

optimized to reduce redundant logic according to the specified settings. The generated netlist will contain the needed logic elements and the connectivity among them as described by the HDL code.

2. Implementation: The synthesized logic is routed to fit onto the device. The implementation phase will then technology map the logic elements in the netlist to the primitives available in the selected FPGA model so that the design could be implemented on our physical chip. In addition, this step will place and route the primitives on the FPGA layout virtually per the constraints from designers and physical aspects to make the final design meet the power, performance, and area requirements.
3. Bitstream generation: Finally, the placed and routed netlist will be translated to the binary configuration data, called bitstream with the vendor-specific tool. The bitstream is stored in the Desktop directory.
4. Then, download it to the target device to fulfill the functionality. Bitstreams to be stored in the FPGA, but they will be volatile meaning that lost once the FPGA loses power. Persistent storage is available on the Artix-7 in the form of an SPI flash chip.

   For this experiment, the pre-generated bit file "Laser_FSM.bit" can be used.

13.4.4 Hardware Connection

Now we have to make sure all the electrical wires are properly connected, as shown in Fig. 13.10. Once all the connections of the diode laser, spider, NIR camera, and joystick controller for the XYZ board are thoroughly checked, and then we will power the laser station. Spider, target FPGA board, camera, and joystick controller for XYZ stage will have their separate COM ports in the connected PC. Then we have to connect an oscilloscope to the output pin of the FPGA to observe the bitflip, as shown in Fig. 13.11. In this case, the output pin is T22 of the AC701 evaluation board.

13.4.5 Placement of the FPGA

The Artix-7 FPGA will be placed on the XYZ precision board, as shown in Fig. 13.12. To perfectly place the FPGA right under the laser, we will use the camera. In our setup, we have three objective lenses with magnifications of 5×, 20×, and 50×, respectively. We will take picture of the IC to find our area of interest, as shown in Fig. 13.13.

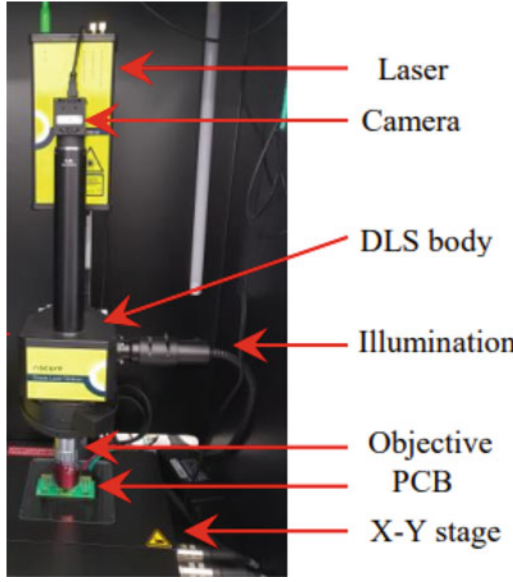


Fig. 13.10 Mounting of diode laser, camera, objective lenses, and XYZ stage

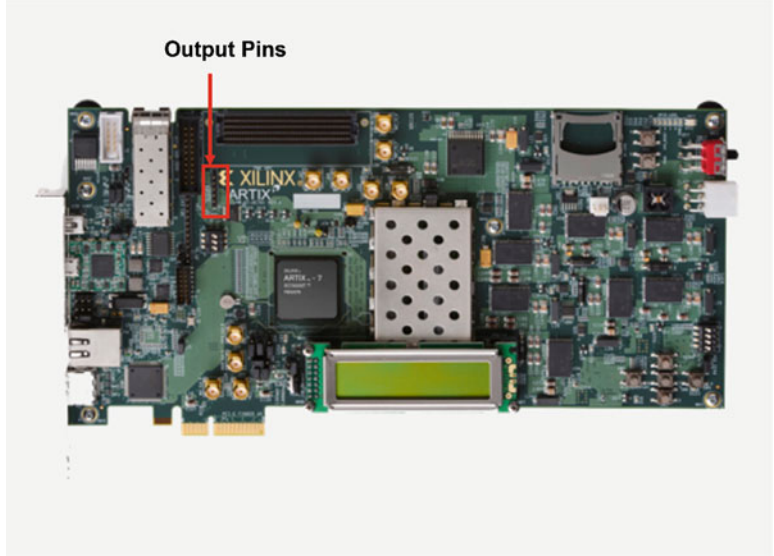


Fig. 13.11 Output T22 pin of AC701 for oscilloscope connection

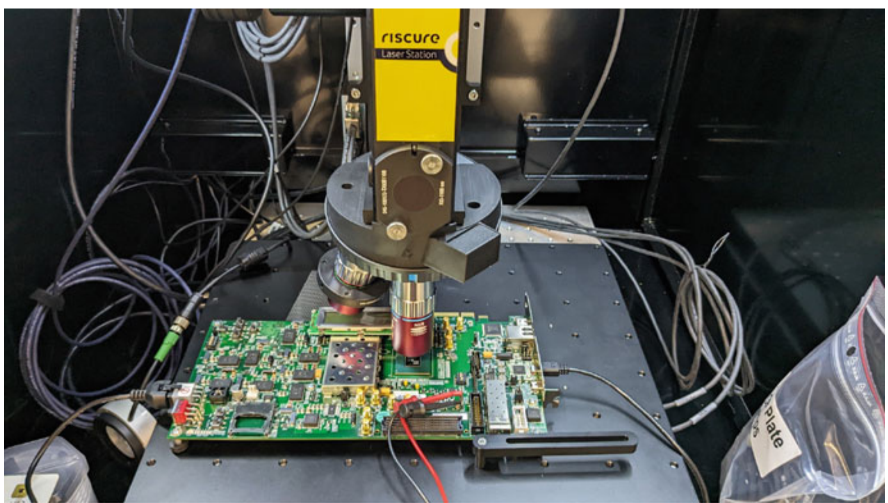


Fig. 13.12 Placement of the Artix-7 FPGA under objective lenses

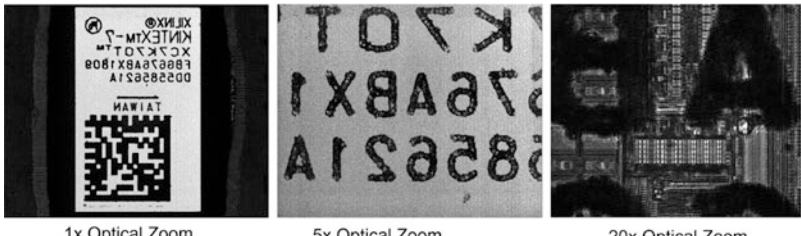


Fig. 13.13 Target IC with different magnifications

13.4.6 Fault Injection Attack

We can carry out fault injection attacks in Inspector software. In this subsection, we will describe how we can use Inspector software. The inspector tool is developed by Riscure [14].

13.4.6.1 Fault Injection by Inspector

After opening inspector Gui [14], we will open the optical perturbation sequence from the project list. The following Python code will be executed.

```
import random
import os
import serial
```

```
from time import time, sleep
from pathlib import Path

from fipy.parameters import *
from fipy.scriptutils import ResultColor, fipy_script
from spidersdk.chronology import Chronology
from spidersdk.spider import Spider
from fipy.transformutil import TransformUtil


tu = TransformUtil()

PARAMETERS = Parameters(
('scans', AttemptsParameter('Scans')),
('xyz_scanner', MaskedXYZScanParameter('XYZ scanner',
    transformutil=tu)),
('z_offset', IntParameter('Z Offset', unit='um')),
('pulse_power', IntParameter('Pulse Power', unit='%')),
('pulse_length', IntParameter('Pulse Length', unit='ns')),
('pulse_delay', IntParameter('Pulse Delay', unit='ns')),
('spider_com_port', SerialPortParameter('Spider COM')),
('serial_com_port', SerialPortParameter('Pinata COM')),
('serial_baudrate', IntParameter('Pinata Baudrate')),
('serial_timeout', FloatParameter('Pinata read timeout')),
('normal_voltage', FloatParameter('Pinata normal voltage',
unit='V')),)

TRIGGER_IN = 0
RESET_OUT = 2
TRIGGER_OUT = 8
TARGET_POWER = Spider.GLITCH_OUT1
TRIGGER_EDGE = Spider.RISING_EDGE
DLS_PULSE_AMPLITUDE = Spider.VOLTAGE_OUT1
DLS_DIGITAL_GLITCH = Spider.GLITCH_OUT2
DLS_GLITCH_VOLTAGE = 3.3


@fipy_script
def execute_script(util):
    util.set_termination_timeout(5)
    util.parameter_init(PARAMETERS)

    script_name = Path(__file__).stem
    db = util.create_database_table('logs/{}.sqlite'.
    format(script_name), script_name)
```

```
util.add_to_cleanup(util.close_database)

xyz_interface = util.get_xyz()
tu.add_system('table',
xyz_interface.get_reference_points())

# # Hardware initialization (Spider)
spider_com_port = serial.Serial()
spider_com_port.port = str(PARAMETERS['spider_com_port'])
spider_com_port.open()
spider_core1 = Spider(Spider.CORE1, spider_com_port)
spider_core1.reset_settings()
util.add_to_cleanup(spider_com_port.close)

# Hardware initialization (Pinata)
serial_target = serial.Serial()
serial_target.baudrate =
int(PARAMETERS['serial_baudrate'])
serial_target.timeout =
float(PARAMETERS['serial_timeout'])
serial_target.port =
str(PARAMETERS['serial_com_port'])
serial_target.open()
serial_target.reset_input_buffer()
serial_target.reset_output_buffer()
util.add_to_cleanup(serial_target.close)

try:
    glitcher = Chronology(spider_core1)
except IndexError as e:
    raise Exception(str(e) +
        "\n\nDid you select the right COM port for
        Spider?
        Is it powered on?")

glitcher.forget_events() # Forget any previous
added events

normal_vcc = float(PARAMETERS['normal_voltage'])

counter = 0
do_reset = True
expected_response = bytes.fromhex('6986')
success_response = bytes.fromhex('9000')
```

```
# Initialize target VCC
glitcher.set_vcc_now(TARGET_POWER, 0)
# Initialize the digital glitch output
glitcher.set_vcc_now(DLS_DIGITAL_GLITCH, 0)
# Initialize the pulse amplitude output
glitcher.set_power_now(DLS_PULSE_AMPLITUDE, 0)

# Transform to chip coordinates using the warping
tool.
# This tools will use chip warping when it is
enabled in the project settings,
# and otherwise perform regular transformations
between table and chip coordinates

transform = util.get_warping_tool()

for p in PARAMETERS:
    t = time()
    if not util.process_commands():
        break

    glitcher.set_power_now(DLS_PULSE_AMPLITUDE,
    p['pulse_power'])

    chip_pos = p['xyz_scanner']
    table_pos = transform.from_chip(chip_pos)
    # By default, tango controls are reversed,
    meaning
    that adding positive
    # numbers to the z position will move the
    mounted lens up, increasing
    # the distance between the lens and the table
    platform.
    z = table_pos.z + p['z_offset']
    xyz_interface.move_abs(table_pos.x, table_pos.y, z)

    glitcher.forget_events()

    if do_reset:
        # Sleeps might require manual tuning, based
        on device
        glitcher.set_vcc_now(TARGET_POWER, 0)
        glitcher.set_gpio_now(RESET_OUT, 0)
        sleep(1e-3)
        glitcher.set_vcc_now(TARGET_POWER, normal_vcc)
```

```
        glitcher.set_gpio_now(RESET_OUT, 1)
        sleep(100e-3)
        do_reset = False

    glitcher.set_gpio(TRIGGER_OUT, 1)
    glitcher.wait_trigger(TRIGGER_IN, TRIGGER_EDGE,
    count=1)
    glitcher.set_gpio(TRIGGER_OUT, 0)
    glitcher.glitch(
        DLS_DIGITAL_GLITCH,
        DLS_GLITCH_VOLTAGE,
        p['pulse_delay'] / 1e9,
        p['pulse_length'] / 1e9)

    glitcher.start()

    pin_guess = os.urandom(4)

    serial_target.write(b'\xA2' + pin_guess)
    pin_response = serial_target.read(2)

    spider_timeout = glitcher.wait_until_finish(1000)

    if spider_timeout:
        color = ResultColor.PINK  # no trigger,
        check setup
    elif pin_response == expected_response:
        color = ResultColor.GREEN
    else:
        # Check if there are more bytes
        pin_response += serial_target.read(1024)
        if pin_response == expected_response:
            color = ResultColor.GREEN
        elif pin_response == success_response:
            color = ResultColor.RED
        elif len(pin_response) == 0:
            color = ResultColor.YELLOW
        else:
            color = ResultColor.ORANGE  # some error

    if color != ResultColor.GREEN:
        # Force TRIGGER_OUT to 0 in case there was
        a problem
        glitcher.set_gpio_now(TRIGGER_OUT, 0)
        do_reset = True
```

```
result = Parameters(
    ("id", counter),
    ("timestamp", int(t)),
    ("iter_t (ms)", int((time() - t) * 1000)),
    ("scan", p['scans']),
    ("x", chip_pos.x),
    ("y", chip_pos.y),
    ("z", p['z_offset']),
    ("pulse_power", p['pulse_power']),
    ("pulse_delay", p['pulse_delay']),
    ("pulse_length", p['pulse_length']),
    ("normal_voltage", p['normal_voltage']),
    ("spider_timeout", spider_timeout),
    ("pin_guess", pin_guess),
    ("do_reset", do_reset),
    ("Data", pin_response),
    ("Color", int(color))
)

util.monitor(result)

counter += 1
db.add(result)
```

In Inspector GUI, a dialogue box will open. In the General tab, we will select "accepts measurements with error," as shown in Fig. 13.14.

In the camera tab, we will select NIR camera, and open the live feed. By observing the live feed, we will adjust the brightness, contrast, and sharpness values. This picture is depicted in Fig. 13.15. In the XYZ device tab, the coordination of the IC will be given input by adjusting the XYZ board using the joystick controller.

In the perturbation tab, we will use 2k for number of measurements, 40% as laser pulse power, 20 ns for pulse delay, and 20 ns as laser pulse duration, as shown in Fig. 13.16.

In the target tab, we will use Spider laser Fi as sequence, 115,200 as target baud rate, voltage out1 as pulse amplitude port, and Glitch out1 as digital glitch port. We will also select 1064 nm wavelength as laser, as depicted in Fig. 13.17. Now, we will now run the script and the laser will be shot to the predefined area of the IC.

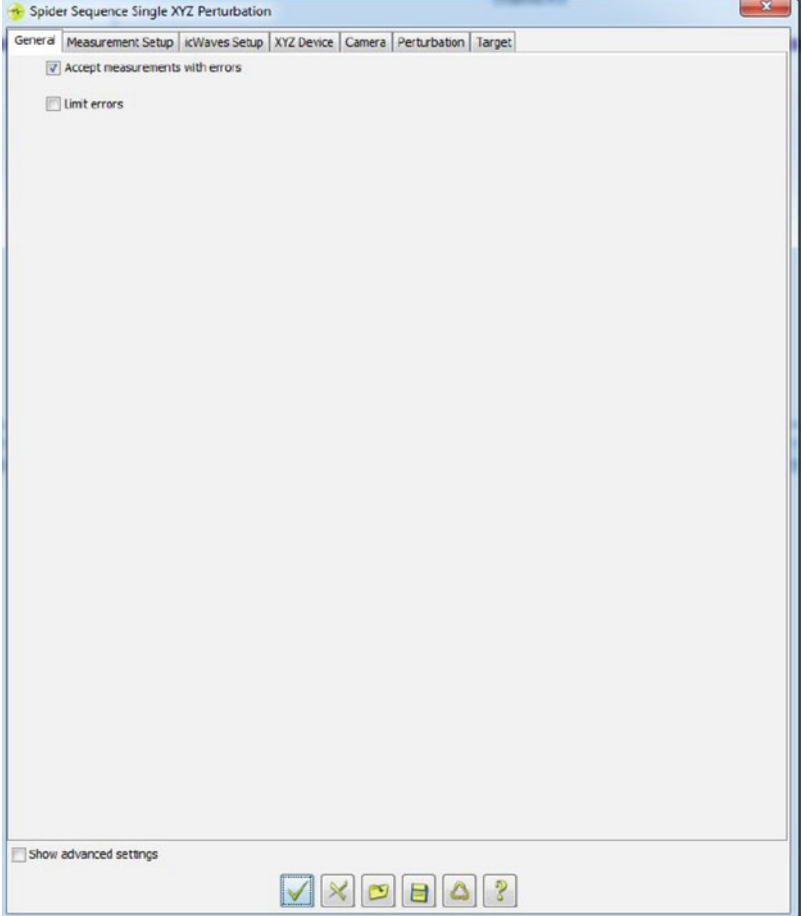


Fig. 13.14 General tab setting in Inspector GUI

13.4.7 *Bitflip Observation*

In this section, we will describe the way by which we can observe the bit flip and analyze the results. We are continually observing the value of the output pin, as shown in Fig. 13.18. As per our finite state machine, all the register values are initialized to zero values. The FSM will only go to Done state only after any of the registers have at least one non-zero value. At Done state, the output will be one. So if we observe a level shift in the oscilloscope, we will know that there has been at least one bit flip. In Fig. 13.18, if the output is zero, then no fault has been injected. If we observe the output level as one in the oscilloscope, it means the FSM is in Done state and at least one fault has been injected

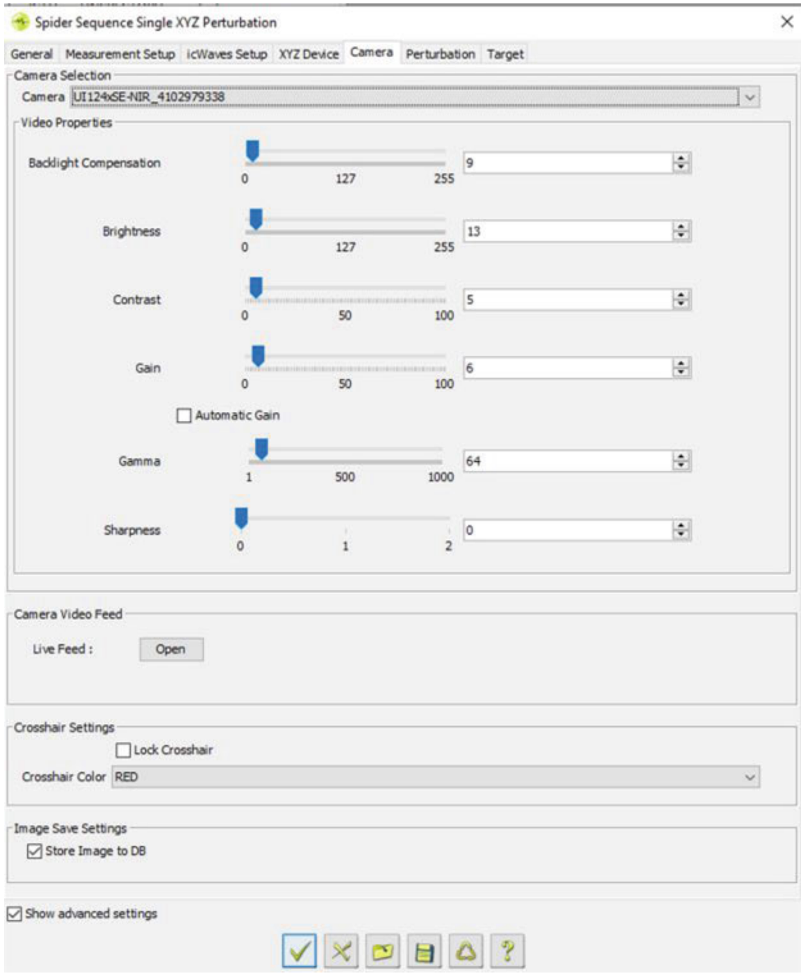


Fig. 13.15 Camera tab setting in Inspector GUI

13.5 Conclusion

In this chapter, the practical aspects of laser fault injections are described in detail. In this experiment, we injected faults in a very basic circuit and tried to observe the faults. However, there are several limitations with this experiment which reduce the probability of bit flips. The spatial and temporal search space is huge for perfectly injecting and observing faults. As of now for a single-bit flip observation, millions of laser injections would be necessary.

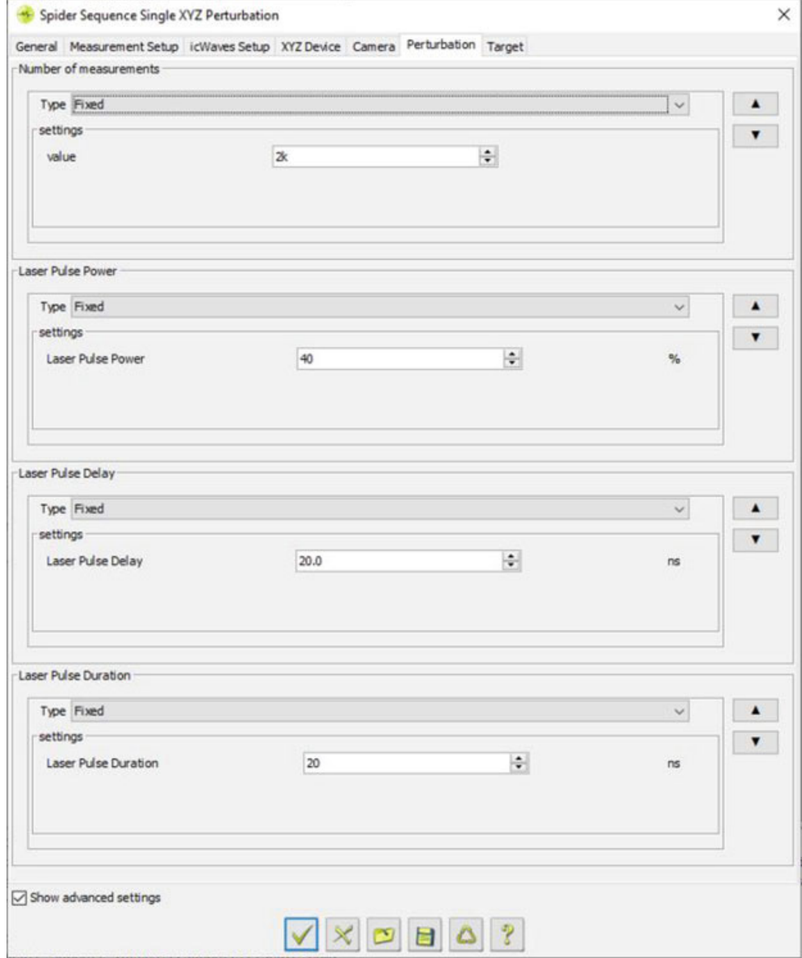


Fig. 13.16 Perturbation setup tab setting in Inspector GUI

Fig. 13.17 Target tab setting in Inspector GUI

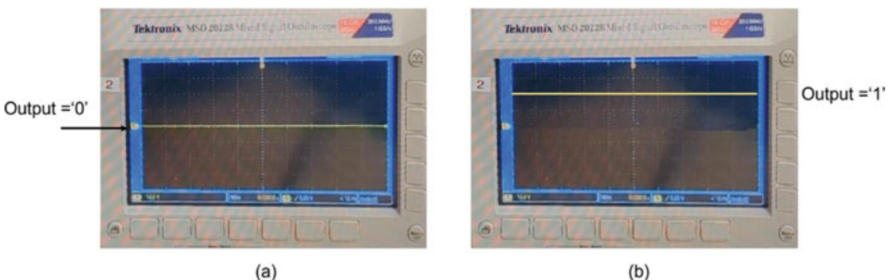


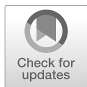
Fig. 13.18 Observation of level shift in the oscilloscope. (**a**) When output is zero, (**b**) when the output is one

References

1. Ahmed, B., Bepary, M.K., Pundir, N., Borza, M., Raikhman, O., Garg, A., Donchin, D., Cron, A., Abdel-moneum, M.A., Farahmandi, F., et al.: Quantifiable assurance: from IPs to platforms (2022). arXiv preprint arXiv:2204.07909
2. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and active combined attacks: combining fault attacks and side channel analysis. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), pp. 92–102. IEEE, Piscataway (2007)
3. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Design and analysis of FPGA based PUFs with enhanced performance for hardware-oriented security. ACM J. Emer. Technol. Comput. Syst. **18**(4), 1–26 (2022)
4. Anandakumar, N.N., Hashmi, M.S., Sanadhya, S.K.: Field programmable gate array based elliptic curve Menezes-Qu-Vanstone key agreement protocol realization using physical unclonable function and true random number generator primitives. IET Circuits Devices Syst. **16**(5), 382–398 (2022)
5. Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.C., Laprie, J.C., Martins, E., Powell, D.: Fault injection for dependability validation: a methodology and some applications. IEEE Trans. Softw. Eng. **16**(2), 166–182 (1990)
6. Asadizanjani, N., Rahman, M.T., Tehranipoor, M.: Physical Assurance. Springer, Cham (2021)
7. Azambuja, J.R., Kastensmidt, F., Becker, J.: Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors. Springer, Berlin (2014)
8. Baumann, R.C.: Radiation-induced soft errors in advanced semiconductor technologies. IEEE Trans. Device Mater. Reliab. **5**(3), 305–316 (2005)
9. Biswas, L.K., Lavdas, L., Rahman, M.T., Tehranipoor, M., Asadizanjani, N.: On backside probing techniques and their emerging security threats. IEEE Design Test **39**(6), 172–179 (2022)
10. Breier, J., Jap, D.: Testing feasibility of back-side laser fault injection on a microcontroller. In: Proceedings of the WESS'15: Workshop on Embedded Systems Security, pp. 1–6 (2015)
11. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on AES. In: International Conference on Applied Cryptography and Network Security, pp. 293–306. Springer, Berlin (2003)
12. https://www.xilinx.com/products/boards-and-kits/ek-a7-ac701g.html: Xilinx Artix-7 FPGA AC701 Evaluation Kit
13. Hsieh, C.M., Murley, P.C., O'Brien, R.R.: Collection of charge from alpha-particle tracks in silicon devices. IEEE Trans. Electron Devices **30**(6), 686–693 (1983)
14. https://www.riscure.com: Excellent insight driving more secure device software and hardware
15. Jordan, A., Milnes, A.: Photoeffect on diffused PN junctions with integral field gradients. IRE Trans. Electron Devices **7**(4), 242–251 (1960)
16. Muttaki, M.R., Barker, B.T., Tehranipoor, M., Farahmandi, F.: FTC—a universal low-overhead fault injection attack detection solution. In: ISTFA 2022, pp. 386–391. ASM International, Detroit (2022)
17. Muttaki, M.R., Zhang, T., Tehranipoor, M., Farahmandi, F.: FTC: a universal sensor for fault injection attack detection. In: 2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 117–120 (2022). https://doi.org/10.1109/HOST54066.2022.9840177
18. Pundir, N., Li, H., Lin, L., Chang, N., Farahmandi, F., Tehranipoor, M.: Security properties driven pre-silicon laser fault injection assessment. In: 2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 9–12. IEEE, Piscataway (2022)
19. Schellenberg, F., Finkeldey, M., Richter, B., Schäpers, M., Gerhardt, N., Hofmann, M., Paar, C.: On the complexity reduction of laser fault injection campaigns using obic measurements. In: 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 14–27. IEEE, Piscataway (2015)

20. Shuvo, A.M., Pundir, N., Park, J., Farahmandi, F., Tehranipoor, M.: LDTFI: layout-aware timing fault-injection attack assessment against differential fault analysis. In: 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 134–139. IEEE, Piscataway (2022)
21. Stern, A., Mehta, D., Tajik, S., Guin, U., Farahmandi, F., Tehranipoor, M.: Sparta-cots: a laser probing approach for sequential trojan detection in cots integrated circuits. In: 2020 IEEE Physical Assurance and Inspection of Electronics (PAINE), pp. 1–6. IEEE, Piscataway (2020)
22. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
23. Van Woudenberg, J.G., Witteman, M.F., Menarini, F.: Practical optical fault injection on secure microcontrollers. In: 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 91–99. IEEE, Piscataway (2011)
24. Viera, R.A.C., Maurine, P., Dutertre, J.M., Bastos, R.P.: Simulation and experimental demonstration of the importance of IR-drops during laser fault injection. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **39**(6), 1231–1244 (2019)
25. Wang, H., Forte, D., Tehranipoor, M.M., Shi, Q.: Probing attacks on integrated circuits: challenges and research opportunities. IEEE Design Test **34**(5), 63–71 (2017)
26. Wang, H., Li, H., Rahman, F., Tehranipoor, M.M., Farahmandi, F.: Sofi: security property-driven vulnerability assessments of ICs against fault-injection attacks. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **41**(3), 452–465 (2021)
27. Wirth, J., Rogers, S.: The transient response of transistors and diodes to ionizing radiation. IEEE Trans. Nucl. Sci. **11**(5), 24–38 (1964)
28. Xu, X., Li, M.L.: Understanding soft error propagation using efficient vulnerability-driven fault injection. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1–12 (2012). https://doi.org/10.1109/DSN.2012.6263923

Chapter 14
Optical Probing Attack on Logic Locking



14.1 Introduction

The increasing use of digital devices in the modern world calls for an extra layer of protection schemes against hardware attacks. Hardware attacks range from extracting sensitive information such as secret keys from FPGAs or ASICs and/or changing and manipulating memory contents from these devices [4, 8]. Several incidents, such as the big hack [15, 23], have shown the possibility of a global-scale attack on integrated circuits facilitated by large entities or countries. The rising concerns about this scenario have increased the necessity of a detailed study of possible attack strategies and countermeasures in this domain. The study would be helpful in staying ahead of any possible attack scenario and increasing the security depth of any electronic device [19, 27].

The attack approaches had been designed as a means of stealing intellectual property and sensitive data by exposing security protocols. This was accomplished by exposing the protocols. Side-channel attacks are a method for getting around the cryptographic security protocols of a software program. These procedures are based on mathematical problems that are thought to be too difficult for anyone who does not possess the key to be able to solve them. Instead of trying to crack them, the attacker examines how the hardware performs, such as its power consumption or the amount of calculation time it takes when these algorithms are being executed, in an effort to discover what their secrets are [21]. Differential fault analysis is the other important category, and it is the one that blocks the computer system by intentionally causing it to overheat or by inducing faults within the hardware [3]. This can be done, for example, by causing the system to overheat. It is more common for the goal in either scenario to be the recovery of information rather than the destruction of a device. These attacks were first aimed to steal the banking data on chip cards [12, 14]. These methods are currently being used on mobile phones, which have circuits that are not well protected. The situation is considerably more precarious for the Internet of Things, which is characterized by pervasive gadgets

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_14

that have inadequate or no security at all. Optical probing has enabled failure analysis by measuring the optical activity within registers of ASIC or FPGA. These techniques can well be used for extracting cryptographic keys. An adversary can easily rent an optical FA tool with an estimate of $300/h rate for this purpose and might be able to locate and extract key bits [20]. Cyber-attacks, whether they are directed at hardware or software, take use of vulnerabilities. Researchers in the field of cybersecurity work to patch these holes in order to prevent malicious actors from finding and using them.

The logic-locking technique is a popular method deployed in ASICs and FPGAs for hiding and encrypting IP [24, 29]. The method requires the insertion of additional inputs known as "key bit inputs" into a circuit. The key bit inputs are either XORed or XNORed with the original inputs to the IP. Without knowing the key bits, it is not possible to get the desired output from the logic-locked circuit. The logic-locking technique serves as a good security measure against exposing the true netlist. This is why logic-locking schemes have become a prime target for attackers. Finding a way to unlock the logic-locked circuit without knowing the key would risk exposing the true nature and functionalities of an IP [22, 30].

Optical probing techniques such as electro-optical probing (EOP) and electro-optical frequency mapping (EOFM) are intended to be used for gathering information about circuit-level activities and failure analysis [26]. These techniques can be used for exposing sensitive information such as the position of memory registers, flip flops (FF), or look-up tables (LUT) in an FPGA and their contents either in the time or frequency domain [5]. This may as well be used for key-value extraction for a logic-locked circuit. Several studies have shown well-developed pipelines for using optical probing to localize and identify key bits in a logic-locked IP. This shows how vulnerable and outdated logic locking is, and it requires additional prevention measures [6, 9, 10, 22].

However, developing prevention measures requires a well-understanding of the attack schemes in the first place. This book chapter is devoted to explaining possible attack scenarios against logic locking in the optical probing domain. A device under test (DUT) requires sample preparation for optical probing to work. The next phase requires localizing the region where key bits are securely stored. This requires a thorough understanding of the device architecture. Later, optical inspection is required to localize the key bit position. After that, electro-optical probing is performed to find out the state of the key bit registers [17, 20]. An elaborate pipeline of this method is well demonstrated in this chapter with proper background, visuals, and examples. It is expected that the reader will have a fine understanding of the concepts described in this chapter after studying them, and this will give them insight into how to use the dedicated devices for this sort of experimentation, such as the PHEMOS installed in the FICS lab [1]. Once the problem is well understood, it will be easier for a researcher to spot the vulnerabilities and the parameters that play a crucial role in various attack scenarios such as the dimension of the DUT, laser resolution and spot size, spacing between gates, and how a system can be developed from hardware perspective [18] or at the simulation end. In addition, it will help them develop prevention schemes for logic locking against optical attacks.

In this chapter, we focus on the optical probing attack on logic locking. In particular, we demonstrate how to extract security-critical information, like a locking key, from hardware systems that incorporate logic locking circuitry [2]. We use electro-optical probing (EOP) for time-domain analysis and electro-optical frequency mapping (EOFM) for frequency-domain analysis, both of which can be used on the PHEMOS-1000 machine [1]. The rest of the chapter is organized as follows: Sect. 14.2 discusses on optical probing techniques. Section 14.3 briefly provides the experimental setup to perform an optical probing attack. Results of the performed optical probing attack steps are given in Sect. 14.4. Finally, conclusions are presented in Sect. 14.5.

14.2 Background

14.2.1 Optical Probing Overview

Two recent trends in microelectronics fabrication serve as the foundation of the attack discussed in this module. The first trend is the rise of new failure analysis (FA) techniques to localize chip defects in a rapid, semi-invasive manner. Optical methods like photon emission analysis (PEA), electro-optical probing (EOP), and optical beam-induced resistance change (OBIRCH), to name a few, take advantage of the transparency of silicon to near-infrared (NIR) photons to gather information about the activity of circuit elements in the IC. The second trend is the rise of flip-chip packaging. Due to increasingly compact designs and higher pin counts, it has become common to insert the die into its package upside-down as shown in Fig. 14.1. This leaves the substrate of the chip exposed as a target platform for attackers to launch a probing attack, with the aforementioned FA tools [28].

It is necessary to understand two optical probing techniques, in particular, to execute this module successfully. The first technique is electro-optical probing (EOP). EOP is based on the probing of transistor signals with an incoherent light source. This laser stimulus passes through the silicon substrate and gets reflected and modulated from different device features (like the active region or metal region on a MOSFET). The reflected laser gets converted into an electrical signal and analyzed on the FA machine side, creating a time-dependent waveform that correlates roughly to the voltage of the target region. This is useful for time-domain analysis on how a specific transistor element switches over time as shown in Fig. 14.2.

The other important optical probing technique is electro-optical frequency mapping (EOFM) as shown in Fig. 14.3. EOFM can be thought of as the complement to EOP since it is used for analysis in the frequency domain instead of the time domain. In EOFM, a larger region of interest (RoI) is scanned with the laser, and the modulated reflection is continuously assessed by a preconfigured frequency filter [25]. Once the RoI has been fully swept by the laser, a complete 2D image is shown by the associated software, representing the spatial activity of every node in

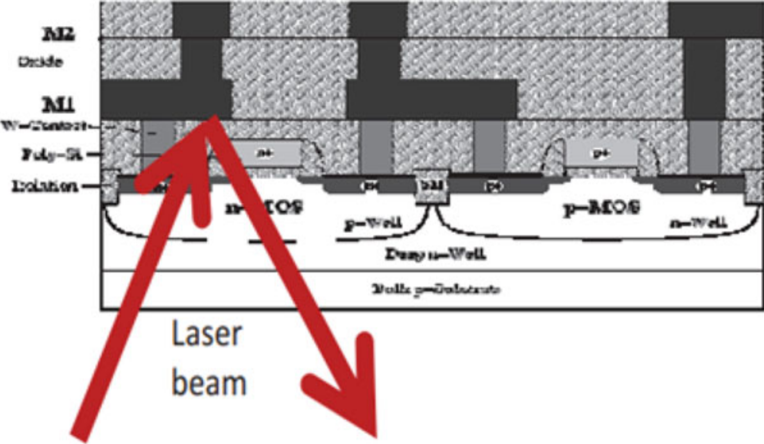


Fig. 14.1 Silicon backside and optical probing path in CMOS [16]

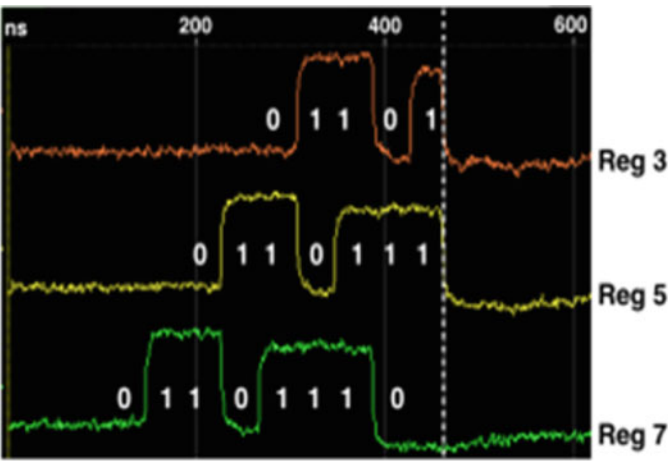


Fig. 14.2 Example optical probing results in the time domain, with values [13]

the region as it pertains to the configured frequency. This is vital in many frequency-dependent applications, like honing in on where a clock signal connects to by observing which registers light up with activity at the clock frequency.

Clearly, EOP and EOFM together form a coherent suite of tools for semi-invasively analyzing circuit activity in both the time and frequency domains. Regarding the FICS lab, these techniques are performed with the PHEMOS-1000 machine as shown in Fig. 14.4. The electronics, power, communication, etc. have been set up in advance, and a set of computers with the appropriate software have been placed adjacent to the machine to serve as the user interface.

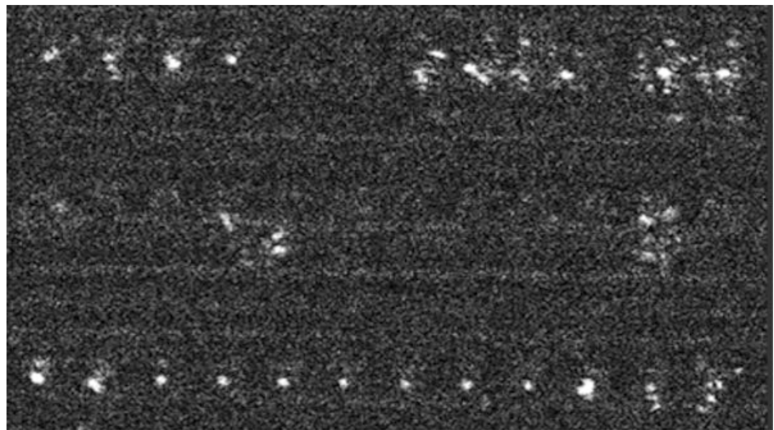


Fig. 14.3 Sample EOFM results from PHEMOS-1000 machine

Fig. 14.4 Hamamatsu photonics PHEMOS-1000 machine [7]

14.2.2 Logic Locking

Logic locking is a defensive scheme that obfuscates the functionality and implementation of a design at the gate level. In theory, a logic-locked circuit prevents attackers from reverse engineering attempts. In practice, this module will challenge this assumption. There are two forms of logic locking: combinational logic locking

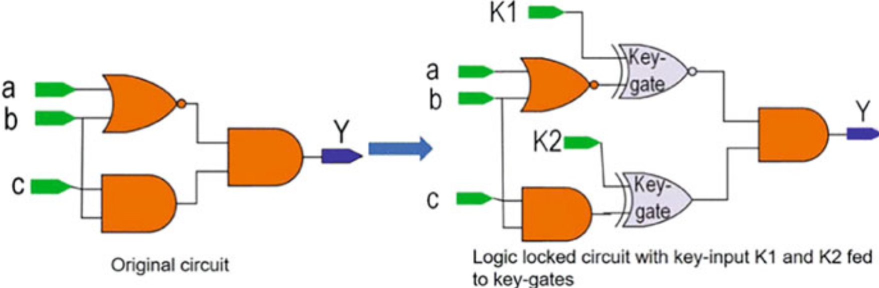


Fig. 14.5 Combinational logic locking circuitry [11]

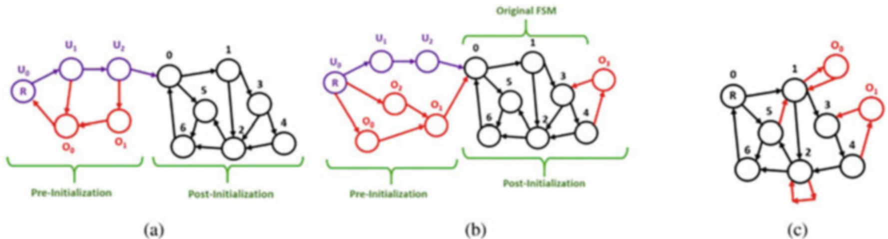


Fig. 14.6 State space of sequential logic locking schemes: (**a**) HARPOON type, (**b**) interlocking type, (**c**) entangled type [11]

and sequential logic locking. In combinational logic locking, the IP is obscured by inserting additional logic gates called "key gates" as shown in Fig. 14.5. The key gates each receive an additional input that collectively comes from a "locking key." The locking key is responsible for de-obfuscating/unlocking the design, so it gets placed in secure nonvolatile memory (NVM) after fabrication. The key's vulnerability comes from the fact that it propagates from NVM through additional, probing-susceptible registers before reaching the key gates. These registers, known as "key registers," will be a major focus in the optical attack.

On the other hand, sequential logic locking obfuscates IP on a more abstract level, by creating additional states for the finite state machine (FSM) in the design as shown in Fig. 14.6. For this reason, it is often referred to as FSM locking or state-space obfuscation. FSM locking can come in a wide variety of implementations. The main ones to acknowledge are HARPOON type, interlocking type, and entangled type. In HARPOON type, pre-initialization states are added such that the original FSM states can only be accessed after a proper, key-based traversal through a specific sequence of states in pre-initialization. Interlocking type also inserts a pre-initialization space, but even if the wrong key is used, the FSM eventually reaches its original state. However, proper key/traversal is required to preemptively unlock the correct FSM functionality for later on in the original state space. Finally, entangled-type locking chooses to keep one main state space region without adding a pre-initialization space. However, additional obfuscation states are directly scattered throughout the FSM's original state space.

Ultimately, the goal of an optical attack on logic locking is to extract the key, whether it be for combinational or sequential locking. Combinational locking attacks have been verified experimentally with optical probing, mainly by taking advantage of the key registers. Sequential locking attacks have presented a tougher challenge, as the algorithms developed to iteratively extract the key bits have proven to be ineffective. Still, current research aims to continue tear apart the overly confident security assumptions of sequential locking.

14.3 Experiment Setup

Once the device under test is selected, the sample needs to be prepared for optical probing, and the equipment needs to be properly set up. The following steps are to be followed.

14.3.1 Programming the Sample

At this stage, we will program the FPGA to implement a logic-locked circuit as a proof of concept (PoC).

The XOR/XNOR gates connected to K1 and K2 inputs in this implementation obfuscated the circuit (see Fig. 14.7). When K1 = 1 and K2 = 0, the proper input combination, the circuit generates the proper output Y. The circuit's inputs are denoted here by the letters a, b, and c. Four of the logic-locked circuits are

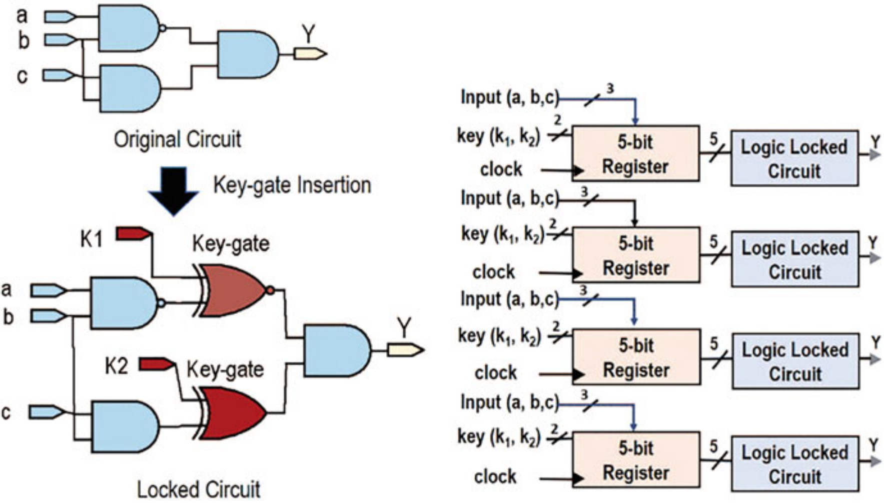


Fig. 14.7 Example of a possible PoC circuit [20]

also implemented in a bigger circuit block as shown in Fig. 14.7. Each logic-locked circuit will be coupled to three parallel input registers that correspond to the a, b, and c ports. Our design uses a key that is 8 bits long for each logic-locked circuit. In order to input the key to key gates, 8-bit parallel registers will also be used. A reset signal is used in the design to replicate the chip's reset procedure. Connecting a PC to the board will allow you to program the FPGA once the HDL code is available. Moreover, design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch14_probing_on_logic_locking/. The FTDI chip on USB is responsible for handling the FPGA's programming. The given development board supply should be used to power the board. Don't make any additional electrical adjustments to the board.

14.3.2 Sample Preparation

The attack surface, our target site for probing, is the backside of the FPGA chip die. X-ray imaging is used to localize the die under the heat sink if not spotted by visual inspection. The sample preparation method depends on whether the chip is packaged as a flip chip or a non-flip chip. Like most modern chips, the sample selected for this module has a flip-chip ball grid array (BGA) package. Use a hotplate and lab knife to remove the heat sink over the chip. This exposes the backside of the chip. Further selective polishing can be performed to increase the resolution of the laser-scanning image.

14.3.3 Measurement Setup

A Hamamatsu PHEMOS-1000 FA microscope provides the optical contactless probing setup, as seen in Fig. 14.8. The equipment comprises of an optical probing preamplifier (Hamamatsu C12323) and a suitable probing light source (Hamamatsu C13193). Connect the FPGA board to the PHEMOS platform, and then turn it on.

14.4 Performing the Attack

14.4.1 Attack on Combinational Logic Locking

The methodology to extract the key from combinational locking scheme is as shown in Fig. 14.9. In order to eventually locate the key registers, the first step is to determine the primary clock frequency of the chip. This can be done in several ways,

Fig. 14.8 Measurement setup

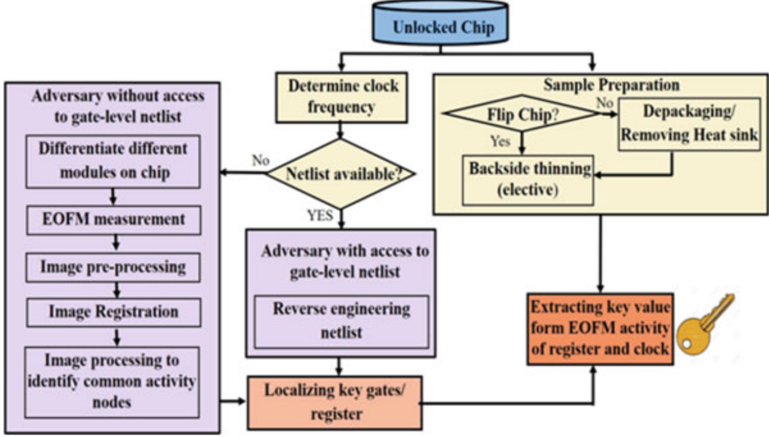


Fig. 14.9 Overview of methodology to extract key from combinational locking scheme [20]

but the most straightforward approach is simply to read the chip's documentation, which commonly includes the clock specifications.

Using the determined clock frequency as the target frequency, launch an EOFM session and analyze which regions on the die are active. These are the sequential elements of the design. Additionally, consider which areas are active specifically during the bootup process of the chip. By forcing a repeating reset loop of the chip and stimulating different inputs to the chip, EOFM analysis during bootup will show the functionally critical sequential logic elements that are consistently

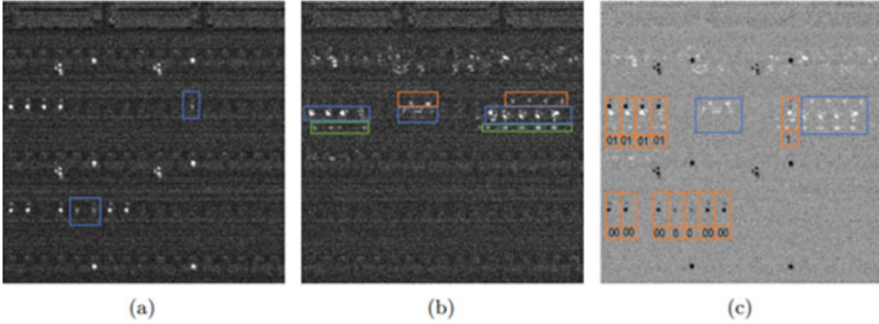


Fig. 14.10 Example of EOFM results for a logic-locked design. (**a**) Target frequency = clock frequency. The activity shows which elements are sequential. The spots in the blue rectangles are slightly dimmer which means there is only one flip flop. The brighter spots, therefore, have two flip flops. This can be confirmed by analyzing the structure of the chip. (**b**) Target frequency = reset frequency. The image shows critical sequential and combinational logic that undergoes activity every time the chip restarts. (**c**) The subtracted image of (**a**) and (**b**). The orange boxes are likely the key registers, since they are both active as sequential/clocked elements and also every time the chip starts up [20]

active. This sequential logic should correlate to the key registers. To confirm this, consider performing additional partial-reverse engineering, like by analyzing the documentation and differentiating the various logic regions as shown in Fig. 14.10.

Once the key registers' locations are known, the value of the key can be extracted from them. This is done with a new EOFM analysis on the key registers, this time at a different target frequency. The new frequency should be set to a target value that is slower than the clock frequency and is equal to the frequency at which the chip will be continuously reset (decide what this value will be and ensure it matches the reset stimulus frequency to the chip). After performing the EOFM sweep, the resulting image will show which key bits undergo switching activity from their reset value every time the chip restarts—these are bits representing a logical 1—and which key bits simply remain inactive at their reset value: these are bits representing a logical 0.

Depending on the architecture of the chip, it can be a bit tricky to resolve the logical values of flip flops that are adjacent to each other, like with the dual-FF scheme in some FPGAs. Distinguishing the values can be done by looking at the overall shape of the activity of the dual-FFs. For instance, if only one FF is active, the combined spot may be less bright and tilted toward the active FF as shown in Fig. 14.11.

After analyzing the activity in each of the key registers, you should now have a complete bit vector. Congratulations! You have successfully extracted the value of the "secure" key that controls the logic locking. This is a powerful move in dismantling a widely used and respected defense scheme. For more detailed information on this attack, please refer to [20].

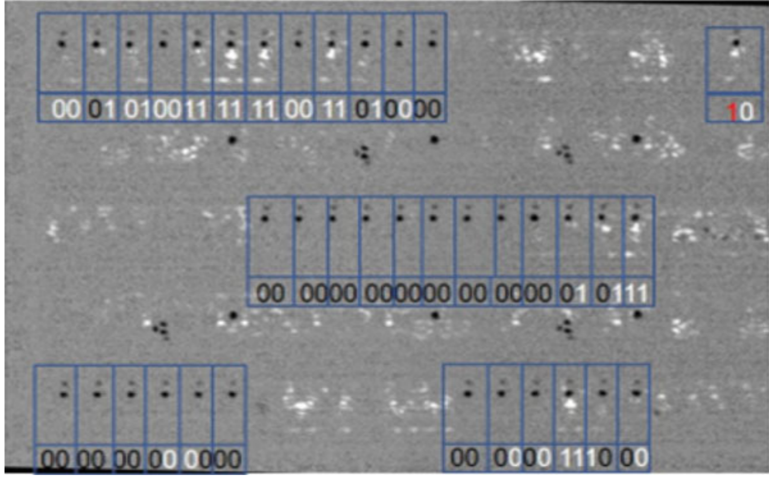


Fig. 14.11 EOFM analysis and results for c1355-CS320 benchmark [20]

14.4.2 Attack on Sequential Logic Locking

Extending the work done against combinational locking to sequential locking has proven to be a formidable task. A comprehensive attack against an established sequential locking benchmark is yet to be done successfully, but current research aims to continue diving deeper into the optical probing technique of extracting a sequential locking key. The challenge in sequential logic locking comes from the fact that the key, which guides the FSM through the proper sequence to unlock the obfuscated circuit, is loaded sequentially in portions depending on the current FSM state. While it still propagates through key registers in a similar manner, the additional time element makes an exclusive EOFM analysis tricky. A tool called SWAG (state-space-obfuscation waveform attack generator) was developed in FICS in 2020, to automatically generate a reset stimulus for the chip. The reset signal would allow the chip to boot up for longer and longer each time in order to observe new frequencies of the sequential key portions passing through the registers. With the EOP approach, the key registers should first be localized in the same EOFM-based manner as done for combinational logic locking. Once these targets are located, EOP should be performed on every register to get the time-domain information of the key bit value. Since the key is loaded sequentially in portions, this will provide information on multiple-bit values (one-bit value for each key portion loaded). After doing this for every register, observe each portion in time across all key bits, then move on to the next time portion, and repeat until the bit vector for every key portion is noted. Again, the successful application of optical attacks on sequential locking is still a frontier area of current research, so do not worry if certain aspects of the attack appear difficult or yield poor results. Valuable, hands-on attack experience can be gained by attempting these procedures.

14.5 Conclusion

In this chapter, we showed that the key being stored on the same chip makes the entire obfuscation open to attack by attackers with a variety of skills, regardless of how secure the locking techniques may be. Unfortunately, up until now, researchers have concentrated on adding additional gates to the IP, sacrificing space and power overhead, in the mistaken belief that the key is safe beneath the cover of tamper-/read-proof memories. This chapter demonstrates that even if a chip has secure or tamper-proof memories, there is still a vulnerability created by the key movement between the memory and key gates of the locked circuit during chip bootup that can be used by an attacker to obtain the key. We draw the conclusion that there is no definite, all-encompassing approach for defending chip assets against optical backside attacks, given the variety of responses researchers have suggested. Consequently, it is essential to create an attack vs. countermeasure matrix to help IC designers include more strong IC security measures without affecting the price, applicability, and dependability of the device.

References

1. Asadizanjani, N., Rahman, M.T., Tehranipoor, M.: Physical assurance for electronic devices and systems. Springer, Cham (2021)
2. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Los Altos (2018)
3. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Annual International Cryptology Conference, pp. 513–525. Springer, Berlin (1997)
4. Birleanu, F.G., Bizon, N.: Reconfigurable computing in hardware security. A brief review and application. J. Electr. Eng. Electron. Control Comput. Sci. **2**(1), 1–12 (2016)
5. Biswas, L.K., Lavdas, L., Rahman, M.T., Tehranipoor, M., Asadizanjani, N.: On backside probing techniques and their emerging security threats. IEEE Design Test **39**(6), 172–179 (2022). https://doi.org/10.1109/MDAT.2022.3185797
6. Biswas, L.K., Shafkat, M., Khan, M., Lavdas, L., Asadizanjani, N.: Emerging nonvolatile memories—an assessment of vulnerability to probing attacks. In: ISTFA 2022, pp. 217–224. ASM International, Detroit (2022)
7. Hamamatsu: Phemos-1000 Emission microscope. https://www.hamamatsu.com/eu/en/product/semiconductor-manufacturing-support-systems/failure-analysis-system/C11222-16.html. Accessed September 2022
8. Japa, A., Majumder, M.K., Sahoo, S.K., Vaddi, R., Kaushik, B.K.: Hardware security exploiting post-CMOS devices: fundamental device characteristics, state-of-the-art countermeasures, challenges and roadmap. IEEE Circuits Syst. Mag. **21**(3), 4–30 (2021)
9. Kamali, H.M., Azar, K.Z., Farahmandi, F., Tehranipoor, M.: Advances in logic locking: past, present, and prospects (2022). Cryptology ePrint Archive
10. Kamali, H.M., Azar, K.Z., Homayoun, H., Sasan, A.: On designing secure and robust scan chain for protecting obfuscated logic (2020). arXiv preprint arXiv:2005.04262
11. Lavdas, L., Rahman, M.T., Tehranipoor, M., Asadizanjani, N.: On optical attacks making logic obfuscation fragile. In: 2020 IEEE International Test Conference in Asia (ITC-Asia), pp. 71–76 (2020). https://doi.org/10.1109/ITC-Asia51099.2020.00024

12. Li, Y., Zhang, X.: Securing credit card transactions with one-time payment scheme. Electron. Commerce Res. Appl. **4**(4), 413–426 (2005)
13. Lohrke, H., Tajik, S., Boit, C., Seifert, J.P.: No place to hide: contactless probing of secret data on FPGAs. In: International Conference on Cryptographic Hardware and Embedded Systems, pp. 147–167. Springer, Berlin (2016)
14. Markantonakis, K., Tunstall, M., Hancke, G., Askoxylakis, I., Mayes, K.: Attacking smart card systems: theory and practice. Inform. Secur. Tech. Rep. **14**(2), 46–56 (2009)
15. Mehta, D., Lu, H., Paradis, O.P., MS, M.A., Rahman, M.T., Iskander, Y., Chawla, P., Woodard, D.L., Tehranipoor, M., Asadizanjani, N.: The big hack explained: detection and prevention of PCB supply chain implants. ACM J. Emer. Technol. Comput. Syst. **16**(4), 1–25 (2020)
16. Perdu, P., Bascoul, G., Chef, S., Celi, G., Sanchez, K.: Optical probing (EOFM/TRI): A large set of complementary applications for ultimate VLSI. In: Proceedings of the 20th IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA), pp. 119–126 (2013). https://doi.org/10.1109/IPFA.2013.6599138
17. Rahman, M.T., Asadizanjani, N.: Backside security assessment of modern SOCs. In: 2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV), pp. 18–24. IEEE (2019)
18. Rahman, M.T., Dipu, N.F., Mehta, D., Tajik, S., Tehranipoor, M., Asadizanjani, N.: Concealing-gate: optical contactless probing resilient design. ACM J. Emerging Technol. Comput. Syst. **17**(3), 1–25 (2021)
19. Rahman, M.T., Rahman, M.S., Wang, H., Tajik, S., Khalil, W., Farahmandi, F., Forte, D., Asadizanjani, N., Tehranipoor, M.: Defense-in-depth: a recipe for logic locking to prevail. Integration **72**, 39–57 (2020)
20. Rahman, M.T., Tajik, S., Rahman, M.S., Tehranipoor, M., Asadizanjani, N.: The key is left under the mat: on the inappropriate security assumption of logic locking schemes. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 262–272 (2020). https://doi.org/10.1109/HOST45689.2020.9300258
21. Randolph, M., Diehl, W.: Power side-channel attack analysis: a review of 20 years of study for the layman. Cryptography **4**(2), 15 (2020)
22. Rinsy, J.J., Sivamangai, N., Naveenkumar, R., Napolean, A., Puviarasu, A., Janani, V.: Review on logic locking attacks in hardware security. In: 2022 6th International Conference on Devices, Circuits and Systems (ICDCS), pp. 342–347. IEEE (2022)
23. Robertson, J., Riley, M.: The big hack: How China used a tiny chip to infiltrate us companies. Bloomberg Businessweek **4**(2018) (2018)
24. Shamsi, K., Li, M., Plaks, K., Fazzari, S., Pan, D.Z., Jin, Y.: Ip protection and supply chain security through logic obfuscation: A systematic overview. ACM Trans. Design Autom. Electron. Syst. **24**(6), 1–36 (2019)
25. Stern, A., Mehta, D., Tajik, S., Guin, U., Farahmandi, F., Tehranipoor, M.: Sparta-cots: a laser probing approach for sequential trojan detection in cots integrated circuits. In: 2020 IEEE Physical Assurance and Inspection of Electronics (PAINE), pp. 1–6. IEEE (2020)
26. Tehranipoor, M.: Emerging Topics in Hardware Security. Springer, Berlin (2021)
27. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer, Berlin (2011)
28. Wang, H., Forte, D., Tehranipoor, M.M., Shi, Q.: Probing attacks on integrated circuits: challenges and research opportunities. IEEE Design Test **34**(5), 63–71 (2017)
29. Yasin, M., Sinanoglu, O.: Evolution of logic locking. In: 2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 1–6. IEEE (2017)
30. Zamiri Azar, K., Mardani Kamali, H., Homayoun, H., Sasan, A.: Threats on logic locking: a decade later. In: Proceedings of the 2019 on Great Lakes Symposium on VLSI, pp. 471–476 (2019)

Chapter 15
Universal Fault Sensor

15.1 Introduction

The modern Internet of Things (IoT) era has seen the evolution of applications not only for general mass involving smart home security, tracking goods, connected appliances, and autonomous vehicles but also for government organizations working with space equipment, healthcare, and financial systems [23]. The emerging security threats against these applications are based on the embedded devices, especially the field-programmable gate array (FPGA) [3, 20] and microprocessors. These devices have been increasingly susceptible to hardware-based attacks [2, 6]. One such attack is the fault injection attack (FIA) which has become one of the leading hardware attacks in recent times. This form of attack is a lucrative option for adversaries for several reasons. They are accessing secret information [9], causing a denial of service [14], and violating data integrity [24].

FIAs can be carried out in different ways, i.e., voltage/clock glitch [6, 9], electromagnetic emanation [8], optical fault injection (OFI) [26], and laser fault injection (LFI) [21]. Figure 15.1 shows an overview of these FIAs. The voltage glitch and electromagnetic fault injection (EMFI) impact the victim device's power line, which in turn creates delay variation through the interconnects. A clock glitch interrupts the original clock signal instantaneously by changing its frequency for a specific cycle causing data corruption through the setup and hold-time violations. Lastly, OFI/LFI impacts systems in a localized way by changing the transistor states. This change causes current flow variation through the transistors and induces voltage variation. To ensure a system's security, it is essential to recognize the traits of FIAs once they are carried out. However, their covert and momentary nature makes it very hard to effectively detect them instantaneously [5, 7, 23].

The research community has proposed different on-chip solutions to address emerging FIAs. However, the solutions were mostly directed toward a specific fault attack. For instance, RC circuit-based detection techniques have been presented in [15, 16] to detect voltage glitch attacks. Similarly, techniques such as monitoring

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_15

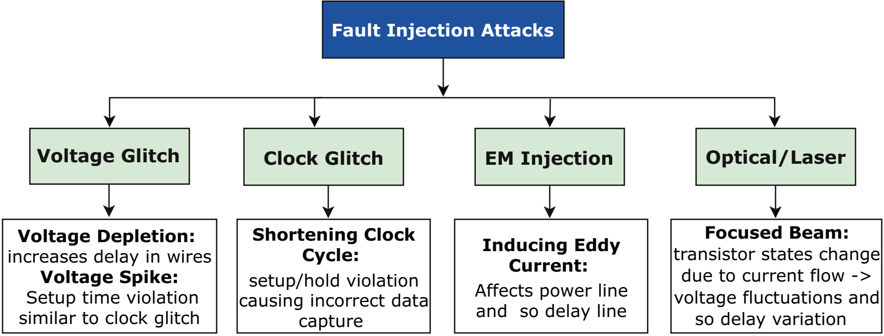


Fig. 15.1 Fault injection attack (FIA) overview [10]

system clocks and phase-frequency detection circuitry have been presented as the solution to detect clock glitch attacks in [13, 18]. Other techniques are proposed for detecting EMFI in [12, 19] and detecting optical fault detection in [11, 17]. For addressing different FIAs, one can think of combining multiple techniques into a device under test (DUT). However, this is not a feasible solution due to the area/power overheads from multiple techniques and affecting their detection capabilities in the presence of other detection hardware in close proximity. In this regard, there is a need for a lightweight, universal solution that can efficiently detect different FIAs.

The rest of the chapter is organized as follows: Sect. 15.2 provides the background of the FIAs with the proposed solution, Sect. 15.3 presents sensor architecture and how it works, Sect. 15.4 discusses the implementation process of the sensor in the FPGA platform, and Sect. 15.5 provides the results from the sensor under different FIAs. Finally, Sect. 15.6 concludes the paper.

15.2 Background

For developing a universal solution to detect different FIAs, it is imperative to understand the impact of each FIA on the DUTs. In this section, we discuss the prominent FIAs and their impact.

- **Voltage glitch:** Voltage glitch attacks can arise from voltage overshoots or undershoots. Both of them can cause timing faults in the design. For instance, considering voltage undershoot, the timing constraint equation can be defined as below [27]:

$$t_{ck} > d_{clk2q} + d_{pMax} + t_{stp} - t_{skew} \tag{15.1}$$

where t_{ck}, d_{ck2q}, d_{pMax}, t_{stp}, t_{skew} define the clock period, register (clock-to-q) delay, data propagation delay, setup time, and clock skew delay, respectively. If the source voltage gets lowered, that will increase the d_{pMax} and create timing faults.

- **Clock glitch:** Clock glitch attack impacts the design clock tree by corrupting one or more cycle(s) of the clock. This change causes setup/hold-time violations while latching the data. This, in turn, leads to incorrect data acquisition by the capturing registers.
- **EM Fault Injection:** During the EM injection attack, an additional magnetic field perturbs the magnetic field generated during running the victim device. The injected EM wave causes eddy current flow in the internal closed wire loop. This eddy current flow creates a potential difference, resulting in delay variation. Thus, the circuit timing is impacted.
- **Optical fault injection:** Light/laser can be used as a medium to emit high-intensity waves to the DUT cells (transistors) in a focused way. This high-intensity emission creates electron-hole (e/h) pairs at the transistor drain, resulting in a current pulse [4]. Finally, a potential difference is built in the presence of load capacitance which can affect the delays of nearby transistors/components, thus impacting the overall timing of the circuit.

We can see from the above discussion that the considered FIA mostly affects the timing of the circuits. If a sensor can be developed that converts and quantifies the FIA impact into timing changes, it will effectively detect multiple FIAs with the same structure. This will solve the problem regarding incurred overheads and performance impacts from implementing multiple hardware for detecting different FIAs. Such a unified sensor named fault-to-time converter (FTC) is presented in the next section.

15.3 FTC Sensor

Figure 15.2 shows the building blocks of the proposed FTC sensor [10]. At the top left of the figure, the system clock signal goes into the *sampling clock generator* block to be scaled to a specific frequency the system will run in. The resultant sampled clock signal drives the FTC block to perceive the delay variations. The major difference between this opposed FTC structure and the time-to-digital converter (TDC) [22] is the use of HVT and LVT cells for creating the buffer delay lines. Using two separate delay lines with two types of V_t cells increases the sensor's effective range, as the LVT and HVT lines are more sensitive to voltage and delay variations. Inside FTC, the sampled clock feeds into the HVT and LVT delay lines and to the clock (enable) ports of the latches. The buffer outputs are *XORed* to detect the mismatch of transmitted values and stored using the latches when the enable pins are de-asserted. As the routing paths for these signals probably be uneven due to constrained hardware resources, the stored values may show irregular 1s and 0s.

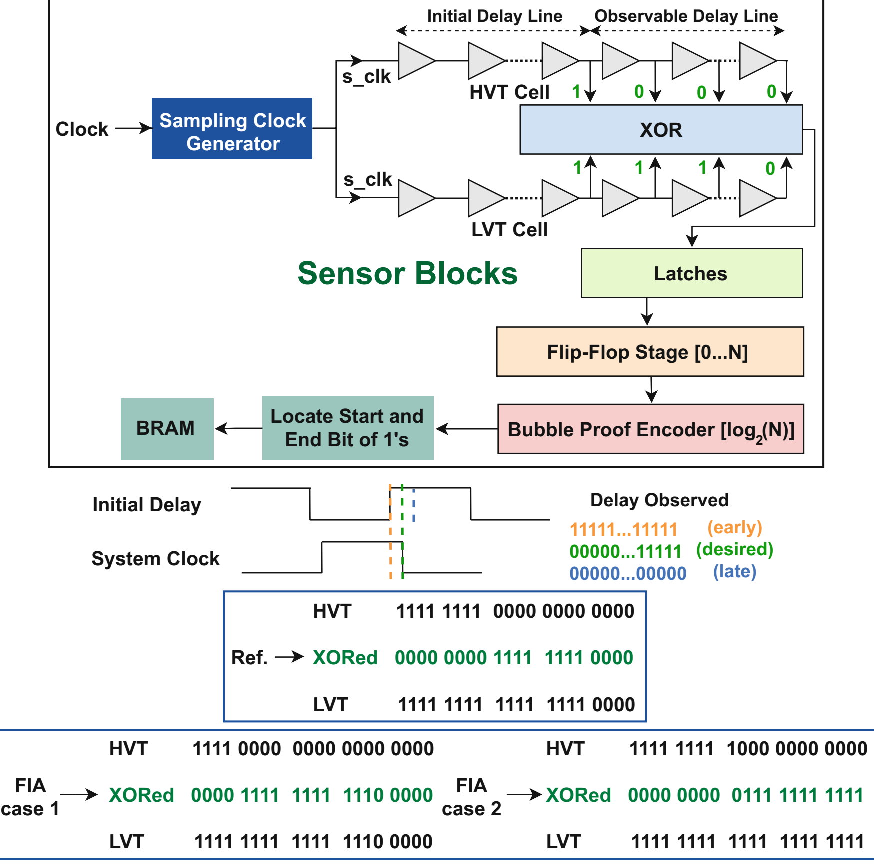


Fig. 15.2 The FTC sensor block diagram [10]

The *bubble-proof encoder* filters out the unexpected 0s (bubbles) and encodes the information of occurring *XORed* 1s into the decimal values. The output of the *XOR* stage in a properly calibrated sensor will be an array of 1s surrounded by 0s on either side. The length of the observable buffer and the initial buffer delay lines can be adjusted to calibrate the sensor. To properly represent this in decimal form, the encoding stage provides two outputs – the bit locations of the least and most significant 1s from the flip-flop stage.

We provide two examples to illustrate the efficacy of the HVT/LVT delay line for generating high-resolution outputs. Suppose there are *20* buffers in both delay lines for the observable delay part. In normal condition (without FIAs), the LVT/HVT buffer values and subsequent *XOR*ed latched outputs are marked as *Ref.* in Fig. 15.2). For this case, the encoded output will be the location of the start and end position of 1s from the flip-flop stage. In example 1, let us consider a delay increase by a fault injection attack due to voltage undershoot. Here, as the HVT cell transistors

require a high threshold voltage to be turned "ON," they will be the most impacted due to the delay increase. As a result, a certain number of 0s will be introduced in the HVT line compared to the LVT. Here, the total number of 1s introduced at the *XOR*ed output is 11. In example 2, we consider a decrease in delay due to the voltage overshoot attack. In this case, as the threshold voltage requirement is less for the LVT cell, they will propagate a higher number of 1s compared to HVT cells. Again, the number of 1s introduced at the *XOR* is 11. When comparing these *XOR*ed outputs to the reference case, we can find a 3-bit flip in the later cases.

We can see a considerable improvement in detectable resolution change when comparing the 1s appearing in *Ref.* LVT to *FIA case 1* LVT and in *Ref.* HVT to *FIA case 2* HVT (only a 1 introduced). These test cases also signify that only a single delay line (LVT/HVT) usage may perform better than *XOR*ed results under one FIA condition (i.e., *Ref.* HVT to case 1 HVT $\Rightarrow$ 4 1s introduced). However, the resolution difference will be minimal for longer delay lines meaning that the *XOR* configuration will perform very close to the best-case scenario for either FIA condition. When we consider the standard threshold voltage cells (SVT), the resolution expected from these cells will be minor compared to the best-case scenarios of either HVT/LVT configurations due to the V_t condition $LVT < SVT < HVT$.

15.4 Hardware Implementation Setup

This section presents the setup and overview of implementing the FTC sensor design in an FPGA environment. The design can also be implemented effortlessly in an application-specific integrated circuit (ASIC) environment. From the design perspective, in FPGA platforms, cells with varying V_t are not readily accessible. Rather FPGAs are composed of SRAM cells with standard V_t. For hardware prototyping purposes in the FPGAs, some adjustments must be made to imitate the behavior of varying V_t cells.

The initial and observable delay lines for (LVT and HVT) can be built using available transparent lookup tables (LUTs) for the specific FPGA model. For modeling the LVT and HVT cells, the number of LUTs can be varied for these two types of cells, for example, 2-LUTs for modeling an LVT buffer vs. 5- LUTs for an HVT buffer. This change in LUT number creates unequal inter-cell delays and more delay in the HVT path compared to LVT. The accumulated delay difference will increase as the signal traverses through more buffers, similar to actual LVT/HVT cells. For detecting FIAs, the sensor is first run under nominal conditions to capture the golden traces. These traces then need to be averaged to minimize measurement noise. Later, different FIAs can be implemented, and the sensor response traces will be recorded. Finally, the FIA instances can be differentiated from nominal cases by comparing the two sets of traces. The sensor sensitivity will depend on calibrating the delays and the spatial location of the sensor instance.

15.4.1 Hardware and Software

For the hardware implementation, we have used Zybo Z7 board with Zynq 7010 FPGA, ensuring it is correctly set up, a p/laptop, and a USB interface for the connection. For the software, we use Xilinx Vivado, where VHDL/Verilog codes of the design modules can be added, simulated, and synthesized, and later the bitstream can be generated. Moreover, all Verilog design files and source codes can be found at http://cad4security.org/index.php/trainings/hsl/ch15_universal_fault_sensor/. After that, the ".bit" file (contains bitstream) is loaded into the user-specified FPGA board. Finally, the integrated logic analyzer (ILA) is used to observe the data from the FPGA run. The sensor consists of five modules, namely, $buffer_LVT$, $buffer_HVT$ (single buffer cell), $buffer_chain_LVT$, $buffer_chain_HVT$ (buffer chain by cascading cells), and $encoder$ (to encode latched output) and the top module, that is, top module. Additionally, there can be other program modules, i.e., the AES encryption module, in the design source section. The design hierarchy can be found in Fig. 15.8.

The code snippet for module $buffer_LVT$ is provided in Fig. 15.3. The HVT cells can be designed similarly by changing the "size" parameter in Fig. 15.3. The code snippet for module $buffer_chain_LVT$ is provided in Fig. 15.4. The HVT chain code can be written in a similar way.

We also provide the code snippet for the encoder design and top module design in Figs. 15.5, 15.6, and 15.7.

```
299  module buffer_LVT # (parameter SIZE = 120)
300   ( vout, vin );
301    input vin;
302    output vout;
303
304    (* DONT_TOUCH= "TRUE" *) wire [SIZE-2:0] inter;
305
306    LUT1 # (.INIT(2'b10)) LUT1_inst1 (.I0(vin), .O(inter[0]));
307    LUT1 # (.INIT(2'b10)) LUT1_inst_final (.I0(inter[SIZE-2]), .O(vout));
308
309    genvar i;
310      generate
311          for(i = 1; i < SIZE-1; i=i+1) begin:gen
312              LUT1 # (.INIT(2'b10)) LUT1_inst2 (.I0(inter[i-1]), .O(inter[i]));
313          end
314      endgenerate
315
316  endmodule
```

Fig. 15.3 Code snippet for LVT cell

```
338 module buffer_chain_LVT #(parameter INIT_DELAY_CELL_SIZE = 50, parameter CELL_SIZE = 50, parameter LUT_COUNT = 2)
339     (vin, vout);
340
341     input vin;
342     output [CELL_SIZE-1:0] vout;
343     (* DONT_TOUCH= "TRUE" *) wire [INIT_DELAY_CELL_SIZE+CELL_SIZE-2:0] inter;
344     genvar i;
345     generate
346         for(i = 0; i < INIT_DELAY_CELL_SIZE+CELL_SIZE; i=i+1) begin:gen
347             // Initial Delay
348             if (i < INIT_DELAY_CELL_SIZE) begin:init_delay
349                 if (i == 0) begin:first_init_delay
350                     buffer_LVT #(LUT_COUNT) b0(.vout(inter[0]), .vin(vin));
351                 end else begin:subsequent_init_delays
352                     buffer_LVT #(LUT_COUNT) b1(.vin(inter[i-1]),  .vout(inter[i]));
353                 end
354             end else begin:buffers // Buffers
355                 if (i == 0) begin:first_buffer
356                     buffer_LVT #(LUT_COUNT) b0(.vout(inter[0]), .vin(vin));
357                     assign vout[0] = inter[0];
358                 end else begin:subsequent_buffers
359                     buffer_LVT #(LUT_COUNT) u1(.vin(inter[i-1]), .vout(vout[i-INIT_DELAY_CELL_SIZE]));
360                     assign inter[i] = vout[i-INIT_DELAY_CELL_SIZE];
361                 end
362             end
363         end
364     endgenerate
365 endmodule
```

Fig. 15.4 Code snippet for LVT cell chain

```
403 module encoder #(parameter SIZE = 64)
404     (capture, num);
405
406     input [SIZE-1:0] capture;
407     output reg [$clog2(SIZE)-1:0] num;
408     (* DONT_TOUCH= "TRUE" *) wire [SIZE-1:0] smoothed_capture;
409     // First AND
410     assign smoothed_capture[0] = capture[0] && capture[1];
411     // Last AND
412     assign smoothed_capture[SIZE-1] = capture[SIZE-1] && capture[SIZE-2];
413
414     genvar i;
415     generate
416         for (i = 1; i < SIZE-2; i=i+1) begin:encoder_smoothing_gen
417             assign smoothed_capture[i] = capture[i] && (capture[i-1] || capture[i+1]);
418         end
419     endgenerate
420
421     always @(smoothed_capture) begin:encoder_gen
422         (* DONT_TOUCH= "TRUE" *) reg [$clog2(SIZE)-1:0] j;
423         (* DONT_TOUCH= "TRUE" *) reg Break;
424         Break = 1'b0;
425         num = 0;
426         for (j = $unsigned(SIZE-1); j > 0; j=$unsigned(j-1)) begin
427             if (!Break && smoothed_capture[j] == 1'b1) begin
428                 num = j;
429                 Break = 1'b1; //Take MSB
430             end
431         end
432     end
433 endmodule
```

Fig. 15.5 Code snippet for encoder design

```
//Buffer Lines
buffers_LVT #(INIT_DELAY_CELL_SIZE, CELL_SIZE, LVT_LUT_COUNT) buffer_line_lvt
( .vin(tdc_clk), .vout(inter_lvt_xor) );
buffers_HVT #(INIT_DELAY_CELL_SIZE, CELL_SIZE, HVT_LUT_COUNT) buffer_line_hvt
( .vin(tdc_clk), .vout(inter_hvt_xor) );


wire [CELL_SIZE-1:0] inter_latch_ff;
(* DONT_TOUCH= "TRUE" *) wire [CELL_SIZE-1:0] inter_ff_encoder;
(* DONT_TOUCH= "TRUE" *) wire [CELL_SIZE-1:0] rev_inter_ff_encoder;


genvar i;
generate
    for (i=0; i < CELL_SIZE; i=i+1) begin:top_gen
        //XOR
        assign inter_xor_latch[i] = inter_lvt_xor[i] ^ inter_hvt_xor[i];
        //LATCH
        LD latch_inst (.G(tdc_clk), .D(inter_xor_latch[i]), .Q(inter_latch_ff[i]));
        //Flip Flop
        FD ff_inst (.D(inter_latch_ff[i]), .C(tdc_clk), .Q(inter_ff_encoder[i]));
    end
endgenerate

// Set rev_inter_ff_encoder = bit reversed inter_ff_encoder
genvar j;
generate
    for (j=0; j < CELL_SIZE; j=j+1) begin:rev
        assign rev_inter_ff_encoder[j] = inter_ff_encoder[CELL_SIZE-1-j];
    end
endgenerate
```

Fig. 15.6 Code snippet for top module design (part 1)

15.4.2 Bitstream Generation

The main steps for generating bitstream are as follows:

- Firstly, the Xilinx Vivado software version (20.1 or newer) needs to be downloaded and installed. The WEBPACK version is free for our target Zybo Z7 FPGA.
- Then, for creating new projects, the "new project" option should be selected. After that, the Vivado GUI will ask for the project name, board name (where it would run), the relevant source, testbench, and constraint files. Assuming that all the relevant files are created beforehand, the project window will look similar to Fig. 15.8.
- In addition to user-created design files, Vivado provides an option to use custom IPs from its "IP catalog." For instance, in this project, we use the multi-mode clock manager (MMCM) to configure a fast PLL clock to observe the system clock (slower) transitions and the dependent buffer output values. Figure 15.11 shows such an interface.

```
//Encoders
encoder #(CELL_SIZE) e1 ( .capture(inter_ff_encoder), .num(vout1) );
encoder #(CELL_SIZE) e2 ( .capture(rev_inter_ff_encoder), .num(vout2) );

(* DONT_TOUCH= "TRUE" *) wire dummyout;
input dummyrst;
AES_top AES_top(.clk(clk), .rst_2(dummyrst),  .dummyout(dummyout));

//Time Multiplexed FF Output///////////////////////////////////////////////////////
input run_ffout;
input ffout_rst;
output [FFOUT_SIZE-1:0] ffout;

reg [$clog2((CELL_SIZE-1)/FFOUT_SIZE + 1):0] ff_ctr;
reg ffout_done;
reg [FFOUT_SIZE-1:0] ffout_reg;

assign ffout = ffout_reg;

always @(posedge probe_clk) begin:ffout_gen
    reg [$clog2(FFOUT_SIZE)+1:0] ffout_index;
    if (ffout_rst) begin
        ffout_done = 1;
    end else if (!ffout_rst && run_ffout) begin //Trigger ffout
        ffout_done = 1'b0;
    end

    if (ffout_done) begin
        ff_ctr <= 0;
        ffout_reg <= 0;
    end else begin
        for (ffout_index = 0; (ffout_index < FFOUT_SIZE)
        && (ffout_index + ff_ctr * FFOUT_SIZE < CELL_SIZE);
        ffout_index=ffout_index+1) begin
            ffout_reg[ffout_index] <= inter_latch_ff[ffout_index + ff_ctr * FFOUT_SIZE];
        end
        ff_ctr <= ff_ctr + 1; //Iterate to next bus of ff's
    end

    if (ff_ctr == $unsigned((CELL_SIZE-1)/FFOUT_SIZE+1)) begin
        ffout_done = 1'b1;
    end
end
```

Fig. 15.7 Code snippet for top module design (part 2)

- Finally, for observing the data after the FPGA run, we use the ILA block that can also be found in the “IP catalog” and configured as per design requirements.
- There are three steps to transform the VHDL/Verilog code into the bitstream in the form of a “.bit” file. They are:

  1. Synthesis: The VHDL/Verilog codes are synthesized into a gate-level representation. In this step, an RTL schematic is created. The user can view the schematic by accessing the “Open Synthesized Design” in the Vivado “Flow Navigator” and then clicking the “Schematic” option. An example schematic view of the FTC sensor design can be seen in Fig. 15.9.

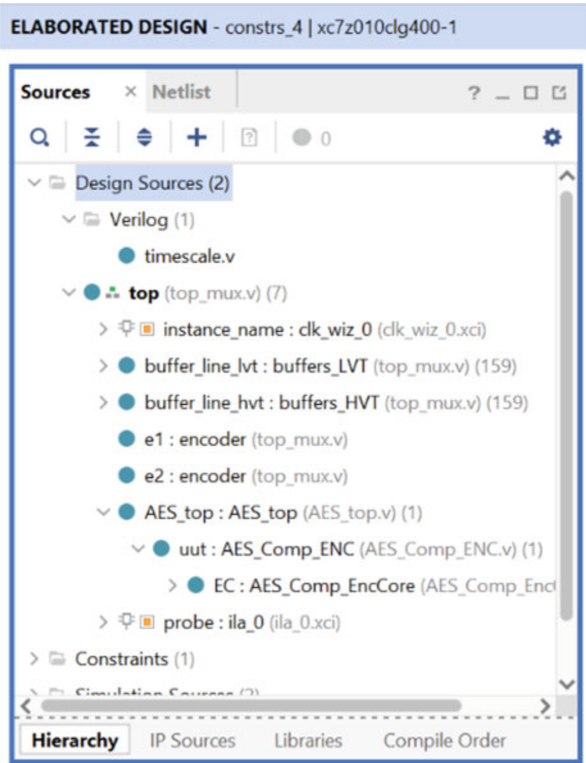


Fig. 15.8 Design hierarchy of the FTC sensor implementation

2. Implementation: In this step, the synthesized logic will be placed and routed according to our user-defined constraint file to fit onto the device.
3. Bitstream generation: After the successful synthesis and implementation, the bitstream is generated. A ".bit" file will be generated, which we will load into the Zybo 7 FPGA board.

- After generating the ".bit" file, the FPGA is required to be attached to a laptop/pc via a USB port. The ".bit" file is loaded into the FPGA by opening "Hardware Manager" and clicking the "program device" option. Then by selecting the generated ".bit" file in the "impl_1" folder under the project directory and clicking the "program" option, the FPGA board can be programmed as shown in Fig. 15.10.

15.4.3 Capturing Output

To observe the output, we have used the ILA IP from the Vivado "IP catalog" (see Fig. 15.12). It can be configured based on the user's requirement with sufficient

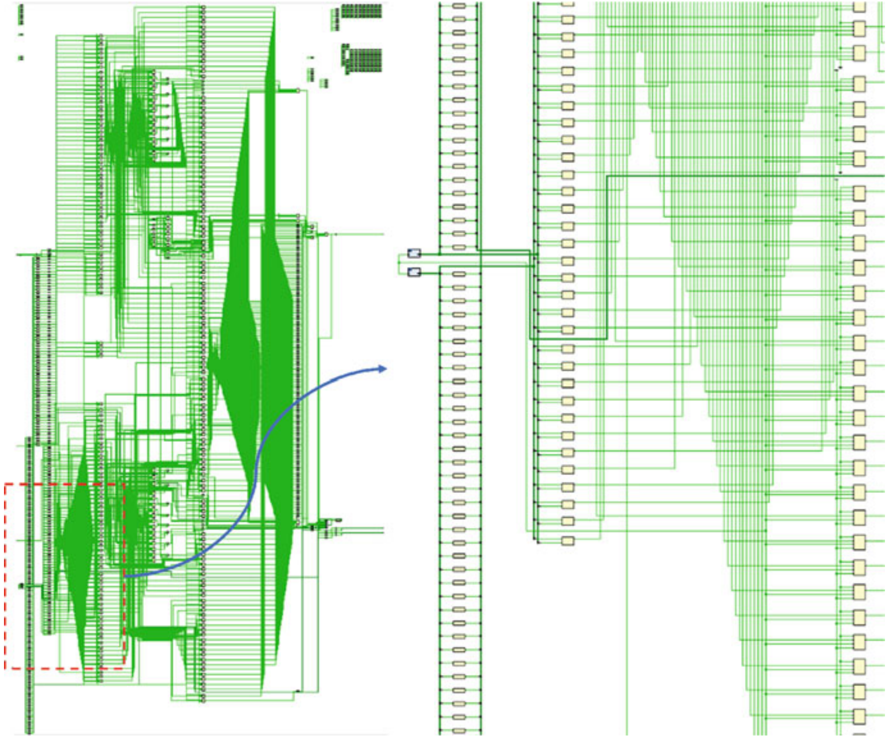


Fig. 15.9 Schematic view for the FTC sensor design

trigger conditions to efficiently debug and calibrate the design. Later, the data file can be post-processed in MATLAB to compare results under nominal and attack conditions.

15.5 Results and Analysis

We have used the FPGA platform Zybo Z7 board having Zynq 7010 FPGA to illustrate the hardware acquired results. We used 128 (N) buffers for the observable delay length for this specific implementation. The encoder output range is from *0* to *127* for the specific implementation. Here, the encoded output *127* signifies that the $0 \rightarrow 1$ transition pulse from the source has reached the final buffer (most significant bit (MSB)) of the observable delay line. We elaborate on three separate setups to facilitate three FIA experiments. These are EMFI, voltage glitch, and clock glitch attacks. For the EMFI attack, we use E1 Immunity Development System [1] from the EMV-Langer to produce a wide span of intensity and waveform options for the generated EM signal. For the voltage glitch, we apply a brief short circuit at

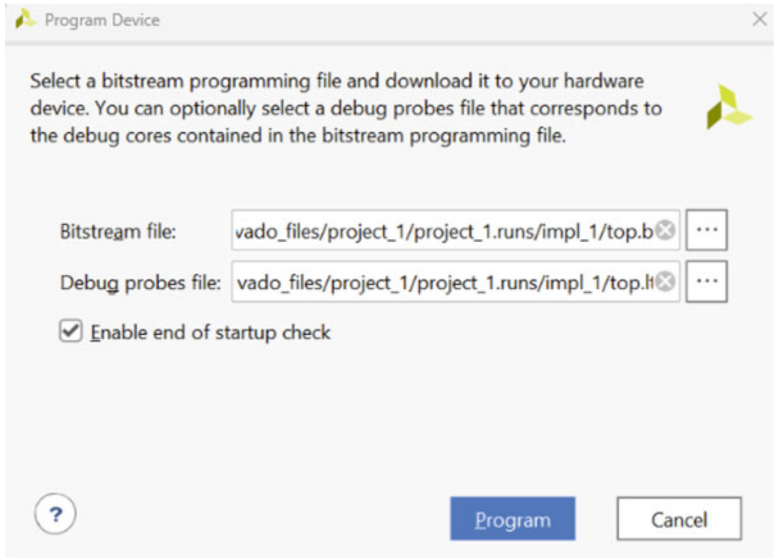


Fig. 15.10 Program device GUI to upload generated bitstream

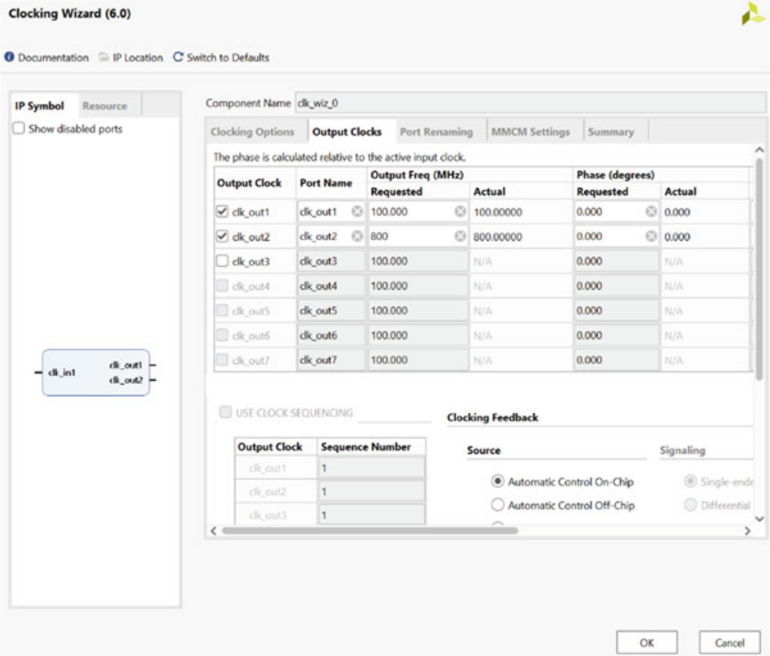


Fig. 15.11 Clocking wizard configuration interface in Vivado

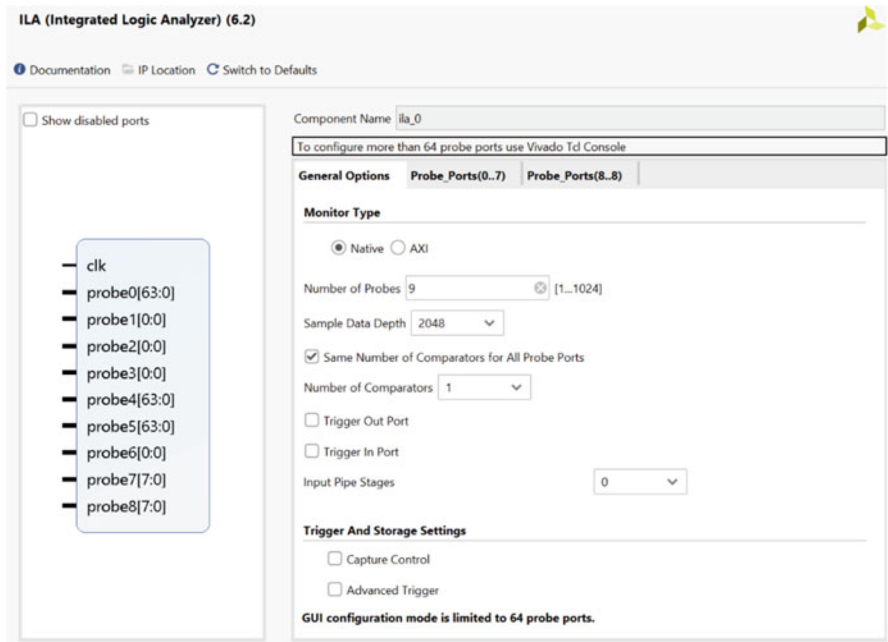


Fig. 15.12 Integrated logic analyzer configuration interface in Vivado

a selected capacitor (part of the reconfigurable unit) from the Zynq 7010 FPGA. Finally, we use the Xilinx multi-mode clock manager (MMCM) that enables us to create a high-frequency clock pulse from the system clock to implement a clock glitch. The sensor was put at close proximity [25] to a running program, i.e., AES encryption as the target of the potential fault injection attacks to diligently observe the change in delays under these FIAs after post-processing the sensor outputs for nominal/attack cases.

15.5.1 EM Attack Analysis

The EM generator specification includes an input voltage range of 500–1500 V, pulse duration (flat/steep), and pulse frequency range of 125–200 MHz. Figure 15.13 shows a setup for the EMFI attack experiment and monitoring process. Figure 15.14 illustrates the sensor outputs with the AES running in normal and EMFI attack conditions on the victim device (FPGA). The solid lines define the traversal of *XOR*ed 1s (from LVT and HVT cells) ranging from observable buffer elements *45–46* to *77–78* under nominal conditions. With the sinusoidal nature of the EM pulse having a varying frequency, the impact is likely to be both a voltage overshoot and undershoot on the delay line. As explained in Sect. 15.3, the HVT

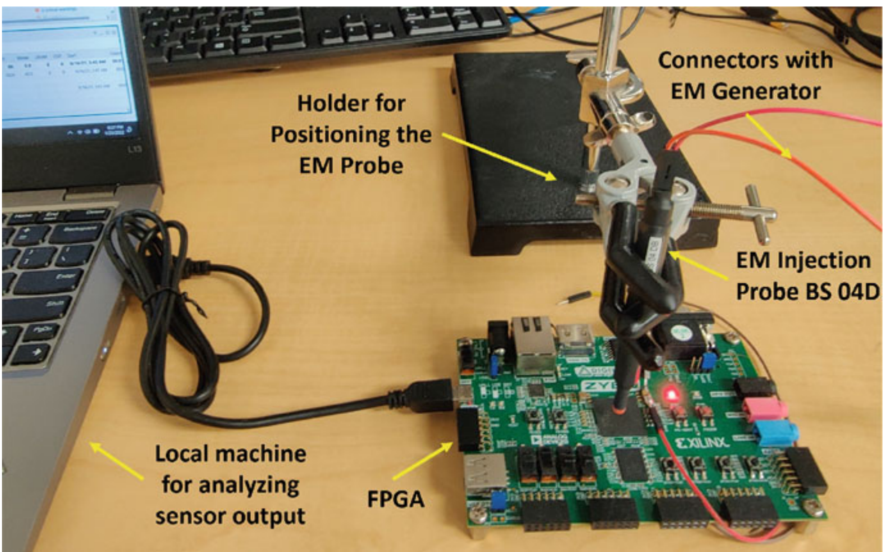


Fig. 15.13 Experimental setup for EMFI attack

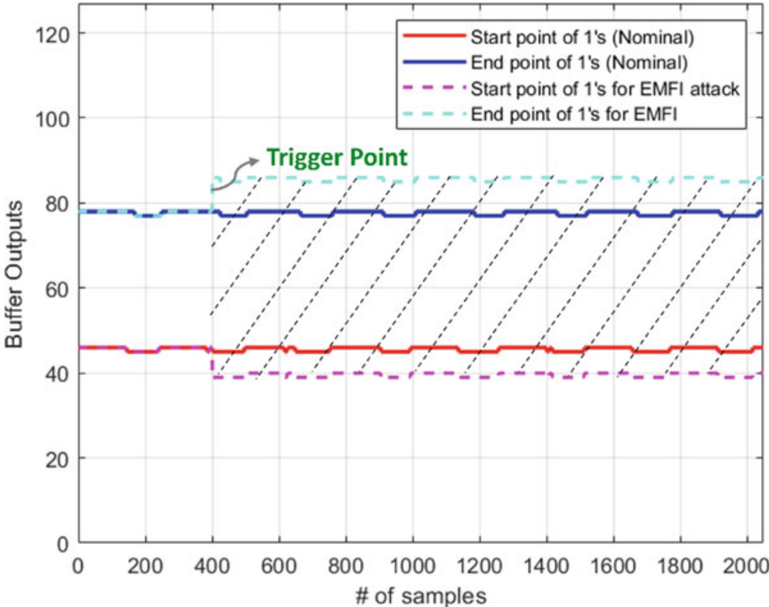


Fig. 15.14 Sensor results for EMFI attack [10]

cells will be impacted more in case of voltage undershoot and introduce *XOR*ed 1s in the initial part of the delay line. This impact can be found in Fig. 15.14 where we can see the decreased dotted line lower limit. However, the upper dotted line jumps to higher values than the solid *blue* line, indicating that the buffers at the farthest distance from the input signal have received 1s (impact of a voltage overshoot). The dashed area shows the updated range for *XOR*ed 1s introduced under EMFI attack.

15.5.2 Voltage Glitch Attack Analysis

To implement this attack, we first reviewed the FPGA power connection schematic to identify the *C108* 100 μF capacitor (reconfigurable unit) for performing short circuit. The drop in voltage between the two terminals was 1.5 V under normal conditions. Figure 15.15 illustrates the encoded results for the voltage glitch attack and the normal run. The short circuit caused a voltage decrease for a short time which essentially is an undervoltage attack. During this attack, the HVT cells are impacted more than LVT cells due to the increased delay through interconnects. Therefore, more HVT cells get 0s, flipping the initial *XOR* outputs to 1s. This effect is seen as the *XOR*ed 1s start from earlier buffers (dotted line under the solid *red* line) at the attack trigger point. Additionally, we find a minimal decrease at the dotted line near the upper limit due to the lesser impact of delay increase on LVT cells. As a result, a small number of *XOR* bits close to the MSB are flipped. Here, the dashed area shows the updated range for *XOR*ed 1s added under this attack.

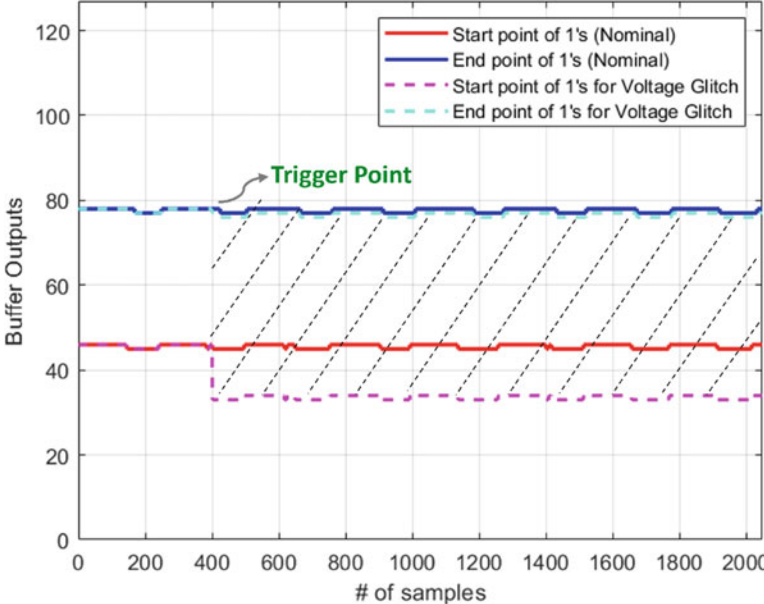


Fig. 15.15 Sensor results for voltage undershoot attack [10]

15.5.3 Clock Glitch Attack Analysis

To implement this attack, a 400 MHz glitch pulse is applied to the AES encryption program running at a slower 100 MHz frequency. An external switch acts as the trigger condition having a counter to implement the glitch and restrict its span to one clock period. Figure 15.16 illustrates the result comparisons for clock glitch (triggered at 400th sample) and the nominal response. We can find the clock glitch impact at the trigger point with it only staying around the clock cycle of the trigger point. As the attack ends and the original clock takes over, the sensor response follows very closely to the nominal trace. The clock glitch attack may not directly impact the delay through the interconnects, but it affects the sampling time of the storage cells after the *XOR* stage. As we have used a high-speed ILA with an 800 MHz sampling frequency clock, anomalies during storing/sampling data based on clock variation could be easily detected in this case. From this analysis, we can assume that clock glitch attacks may not have a consistent pattern based on the impact on delay lines. However, if any anomaly impacts the design during sampling/storing values, the sensor output will deviate from the nominal response to alert the user about the attack condition. While setting a threshold for the sensor through calibration for detecting EMFI and voltage glitch attacks, the same value can be applied to detect this attack.

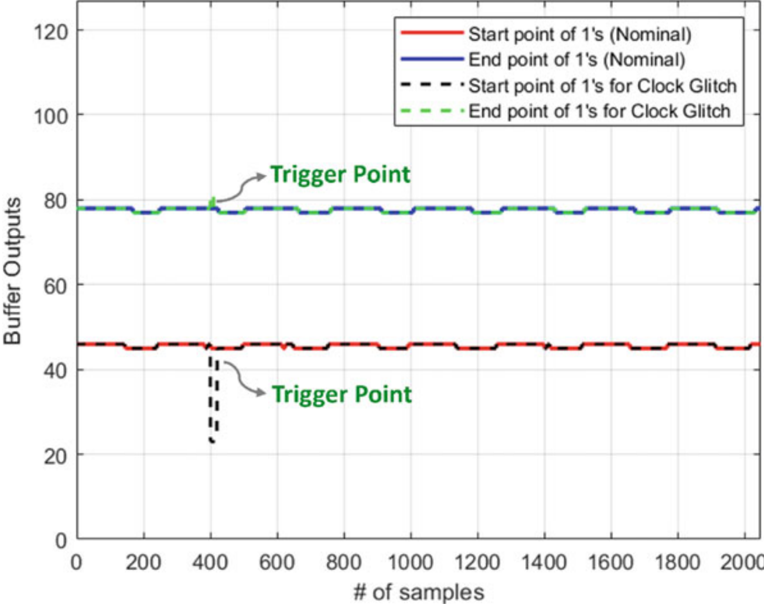


Fig. 15.16 Sensor output for clock glitch attack [10]

15.5.4 Proximity Analysis

The sensor output was also analyzed by varying the position in the FPGA floorplan. The analysis found that the sensor output changes from changing the sensor's location. Additionally, the sensor's sensitivity is impacted by changing from the prior location for which sensor parameters were calculated. This observation calls for further calibration of initial and observable delays for the new location. In Fig. 15.17, the FTC sensor block is placed far away from the AES block, and Fig. 15.18 illustrates the sensor results with no calibration for the later location. As can be seen, the starting and end point differences of *XOR*ed 1s are low. This suggests fewer discrepancies between the stored values of the HVT and LVT cells. This response can be explained as the sensor being distant from the AES block; with the prior choice of initial/observable delays, the variation in delay observed by the sensor becomes less evident. For this reason, some initial (close to LSB) and later (close to MSB) *XOR* 1s are overturned to 0s, causing a reduced dashed area. Lastly, with the new calibration of the delay parameters for the updated location, in Fig. 15.19, a change in *XOR*ed 1s can be seen with improved sensitivity (more *XOR*ed 1s at the starting and ending buffers).

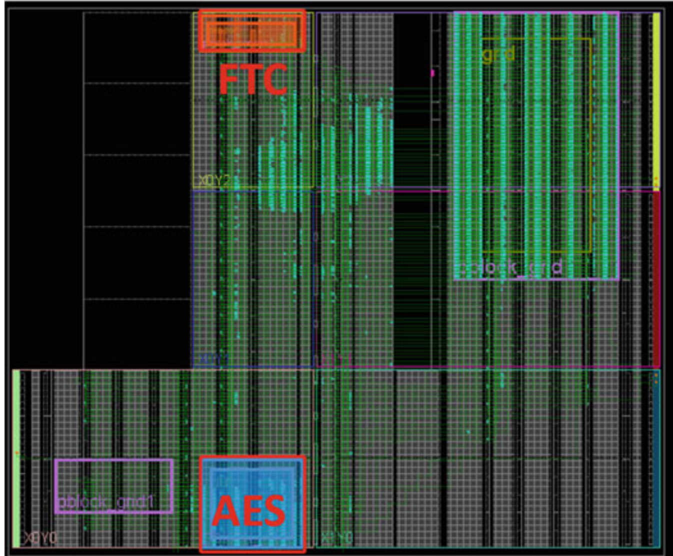


Fig. 15.17 FPGA floor plan showing AES encryption and FTC blocks far away [10]

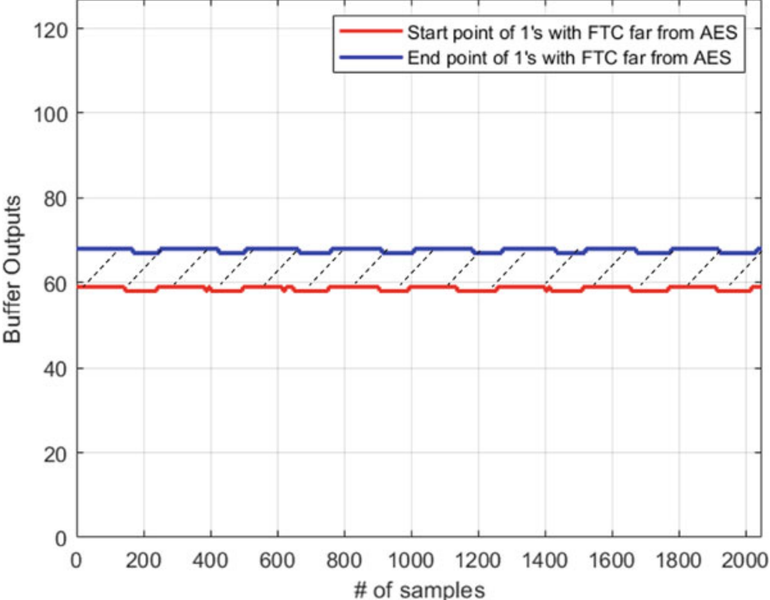


Fig. 15.18 FTC sensor output with no calibration [10]

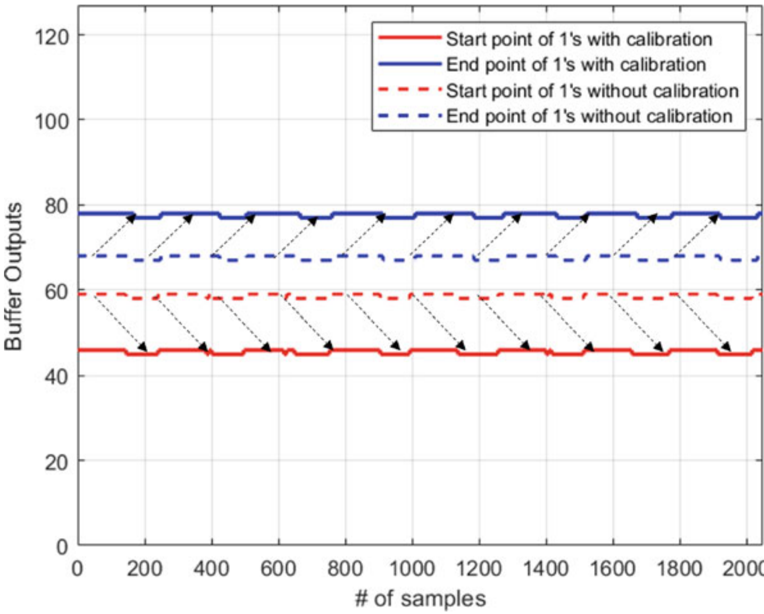


Fig. 15.19 FTC sensor output with calibration [10]

15.6 Conclusion

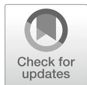
This chapter aims to help readers learn about a unique, unified, on-chip solution against FIAs. Moreover, the chapter provides a direction toward implementing the sensor design at the hardware level with the help of reconfigurable FPGA platforms. To emphasize the quality of the solution, we have added some hardware-generated results collected from the sensor implementation under three different FIAs. Additionally, the overhead incurred due to the additional cells will be minimal due to the very simple structure and usage of small logic cells. In addition, the sensor can be further calibrated and utilized to detect other FIAs that affect the timing parameter of a design. As a result, the proposed solution can go a long way in providing an effective, lightweight on-chip solution for detecting the prominent FIAs.

References

1. 2015, L.E.T.: E1 set: Immunity development system. https://www.langer-emv.de/en/product/immunity-development-system/68/e1-set-immunity-development-system/54
2. Anandakumar, N.N., Das, M.P.L., Sanadhya, S.K., Hashmi, M.S.: Reconfigurable hardware architecture for authenticated key agreement protocol over binary edwards curve. ACM Trans. Reconfig. Technol. Syst. **11**(2), 1–19 (2018)
3. Anandakumar, N.N., Peyrin, T., Poschmann, A.: A very compact FPGA implementation of LED and PHOTON. In: International Conference on Cryptology in India, pp. 304–321. Springer, Berlin (2014)
4. Asadizanjani, N., Rahman, M.T., Tehranipoor, M.: Physical Assurance for Electronic Devices and Systems. Springer Nature Switzerland AG, Cham (2021)
5. Bhunia, S., Tehranipoor, M.: Hardware Security: A Hands-on Learning Approach. Morgan Kaufmann, Burlington (2018)
6. Chen, Z., Vasilakis, G., Murdock, K., Dean, E., Oswald, D., Garcia, F.D.: VoltPillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 699–716. USENIX Association, Berkeley (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai
7. Dey, S., Park, J., Pundir, N., Saha, D., Shuvo, A.M., Mehta, D., Asadi, N., Rahman, F., Farahmandi, F., Tehranipoor, M.: Secure physical design. Cryptology ePrint Archive (2022)
8. Dumont, M., Lisart, M., Maurine, P.: Electromagnetic fault injection: How faults occur. In: 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 9–16 (2019). https://doi.org/10.1109/FDTC.2019.00010
9. Endo, S., Sugawara, T., Homma, N., Aoki, T., Satoh, A.: An on-chip glitchy-clock generator for testing fault injection attacks. J. Cryptograph. Eng. **1**(4), 265 (2011)
10. FTC—A Universal Low-Overhead Fault Injection Attack Detection Solution, International Symposium for Testing and Failure Analysis, vol. ISTFA 2022: Conference Proceedings from the 48th International Symposium for Testing and Failure Analysis (2022). https://doi.org/10.31399/asm.cp.istfa2022p0386
11. He, W., Breier, J., Bhasin, S.: Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering, pp. 27–46. Springer International Publishing, Cham (2016)

12. Homma, N., Hayashi, Y.i., Miura, N., Fujimoto, D., Tanaka, D., Nagata, M., Aoki, T.: Em attack is non-invasive?-design methodology and validity verification of em attack sensor. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2014, pp. 1–16. Springer Berlin Heidelberg, Berlin (2014)
13. Igarashi, H., Shi, Y., Yanagisawa, M., Togawa, N.: Concurrent faulty clock detection for crypto circuits against clock glitch based dfa. In: 2013 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1432–1435 (2013). https://doi.org/10.1109/ISCAS.2013.6572125
14. Khan, M.N.I., Ghosh, S.: Fault injection attacks on emerging non-volatile memory and countermeasures. In: Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, pp. 1–8 (2018)
15. Kim, E.S., Kim, J.H.: Voltage glitch detection circuits and methods thereof. US Patent 7,483,328 (2009)
16. Kim, C.Y., Jun, S.J., Kim, E.S.: Voltage-glitch detection device and method for securing integrated circuit device from voltage glitch attack. US Patent 7,085,979 (2006)
17. Lee, D.G., Choi, D., Seo, J., Kim, H.: Reset tree-based optical fault detection. Sensors **13**(5), 6713–6729 (2013)
18. Luo, P., Fei, Y.: Faulty clock detection for crypto circuits against differential fault analysis attack. Cryptology ePrint Archive (2014)
19. Miura, N., Najm, Z., He, W., Bhasin, S., Ngo, X.T., Nagata, M., Danger, J.L.: Pll to the rescue: A novel em fault countermeasure. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2016). https://doi.org/10.1145/2897937.2898065
20. Park, J., Anandakumar, N.N., Saha, D., Mehta, D., Pundir, N., Rahman, F., Farahmandi, F., Tehranipoor, M.M.: PQC-SEP: Power Side-channel Evaluation Platform for Post-Quantum Cryptography Algorithms. Cryptology ePrint Archive, Paper 2022/527 (2022)
21. Rodriguez, J., Baldomero, A., Montilla, V., Mujal, J.: Llfi: Lateral laser fault injection attack. In: 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 41–47. IEEE, Piscataway (2019)
22. Schellenberg, F., Gnad, D.R., Moradi, A., Tahoori, M.B.: An inside job: Remote power analysis attacks on fpgas. In: 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1111–1116 (2018). https://doi.org/10.23919/DATE.2018.8342177
23. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer Science & Business Media, Berlin (2011)
24. Timmers, N., Spruyt, A., Witteman, M.: Controlling pc on arm using fault injection. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 25–35 (2016). https://doi.org/10.1109/FDTC.2016.18
25. Wang, H., Li, H., Rahman, F., Tehranipoor, M.M., Farahmandi, F.: Sofi: Security property-driven vulnerability assessments of ics against fault-injection attacks. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., 1–1 (2021). https://doi.org/10.1109/TCAD.2021.3063998
26. van Woudenberg, J.G., Witteman, M.F., Menarini, F.: Practical optical fault injection on secure microcontrollers. In: 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 91–99 (2011). https://doi.org/10.1109/FDTC.2011.12
27. Zussa, L., Dutertre, J.M., Clédière, J., Tria, A.: Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In: 2013 IEEE 19th International On-Line Testing Symposium (IOLTS), pp. 110–115 (2013). https://doi.org/10.1109/IOLTS.2013.6604060

Chapter 16
Scanning Electron Microscope Training



16.1 Introduction

The scanning electron microscope (SEM) is a microscope that uses electrons instead of light to form an image. The electronic console and the electron column are the two primary components of the SEM instrument. The electronic console has control knobs and switches that let you change the instrument's focus, magnification, brightness, contrast, and filament current. The electronic console, which houses the control knobs, CRTs, and an image capturing device, is not essential with the state-of-the-art electron microscope because it works in tandem with a computer system. The computer system's mouse and keyboard are used to access all of the main controls. Instead of the traditional control knobs and switches seen on older-style scanning electron microscopes, the operator simply has to be familiar with the graphical user interface (GUI) or software that operates the instrument. The SEM picture is typically displayed on CRTs positioned on the electronic console. Captured images can be saved in digital format or printed right away. Compared to conventional microscopes, the scanning electron microscope has many benefits. The SEM's broad depth of field makes it possible to focus on more of a specimen at once. Closely spaced specimens can be enlarged at much greater levels, thanks to the SEM's significantly superior resolution. The SEM gives the researcher much greater control over the level of magnification because it doesn't use lenses but rather electromagnets. The scanning electron microscope is one of the most valuable tools in research today owing to all of these benefits and the images' genuine remarkable clarity.

The learning objective of this chapter is for readers to learn the fundamentals of scanning electron microscopy and understand how to operate SEM and then how to gain hand experience in IC-level hardware Trojan detection by using nano-image analysis and artificial intelligence. Readers will learn step by step how to use physical inspection methods such as scanning electron microscopy (SEM) to detect any malicious changes from the backside of an IC. Integrated circuits (ICs)

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8_16

can have malicious modifications made to their design at several points during the design and production processes. These are commonly known as hardware Trojans. By assigning a distinct descriptor for each type of logic cells or gates, sophisticated computer vision algorithms are utilized in conjunction with neural network models to categorize authentic and malicious cells from an integrated circuit (IC) under authentication. To identify any slight changes in the active region that would reveal the presence of a potential hardware Trojan, these descriptors are compared to a gold standard.

The rest of the chapter is organized as follows: Sect. 16.2 briefly provides generalized construction of SEM and specimen preparation. Section 16.3 discusses the steps of setting up the experiment for image acquisition with the SEM. Section 16.4 discusses in detail how to detect malicious modifications by untrusted foundry inside an IC using Xilinx FPGA. Finally, conclusions are presented in Sect. 16.5.

16.2 Background

16.2.1 Scanning Electron Microscopy

Scanning electron microscopy is a nondestructive method used for multiple applications such as composition analysis, surface morphology, crystallography, etc. For this chapter first, we will discuss the SEM in relation to the failure analysis of semiconductor structures. The scanning electron microscope (SEM) itself is a powerful tool that uses the mechanisms of different types of electrons (backscattering, secondary, etc.), all of which help create images with nano-scale resolutions. The way that an SEM can do so is by first utilizing an electron source that provides electrons for the beam. This beam of energetic electrons is then directed with an anode and focused by an electromagnetic lens(es) that allow probing of a surface.

From this process, as shown in Fig. 16.1, signals are generated from the interaction of excited electrons and the sample. These signals are then collected through detectors, which are consequently digitized by having the image pixel value synchronized with intensity proportional to the collected signal. There can also be changes in magnification by altering the ratio of the length of the line on the display device to the length scanned on the real sample. Overall, when trying to acquire the best image, it is important to know about the four main probe beam parameters that are used: probe diameter, probe current, probe convergence angle, and accelerating voltage (kV). As the operator, it is also important to keep in mind resolution, depth of focus, image quality (S/N ratio), and analytical performance. As most have heard, the process of getting an image is more of an art than science, but knowing these details will be helpful in setting the foundation in our experimentation.

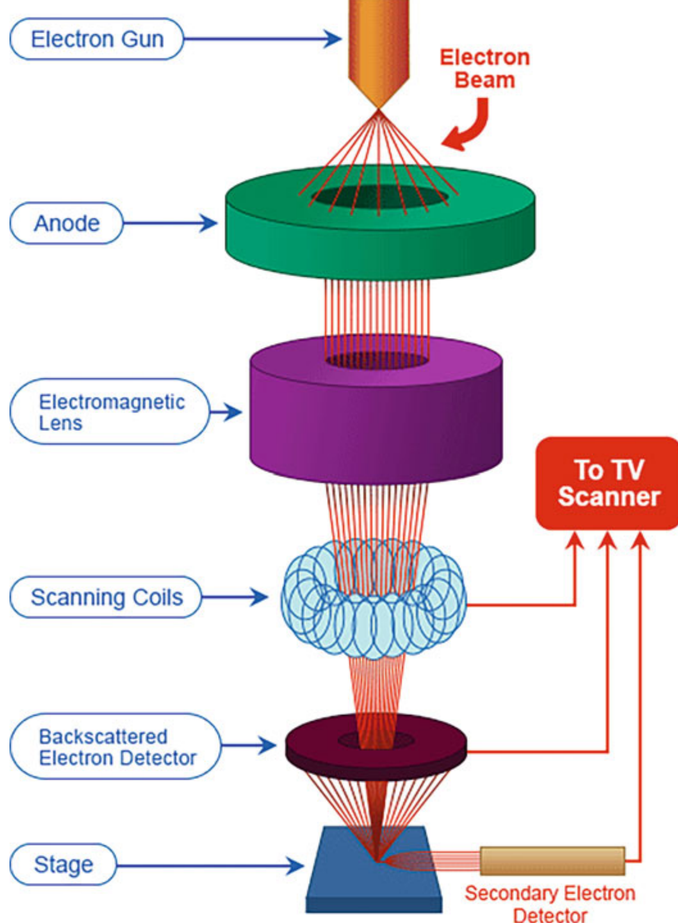


Fig. 16.1 Generalized construction of SEM

16.2.2 Beam Interaction

When a primary electron, or small electron beam, enters a specimen surface, signal detection takes place. After entering the specimen, the primary electron will likely travel a fair distance into the surface. The primary electron will transmit some of its momentum upon impact with another electron, a nucleus, etc. before continuing on a new trajectory. The term "scattering" refers to this inelastic collision. These scattering events are the most intriguing since it is possible to identify the scattering event's components (not all events contain electrons). There are different events which occur inside the SEM chamber as shown in Fig. 16.2 once the electron beam enters the specimen. Some of the most important ones we will focus on include:

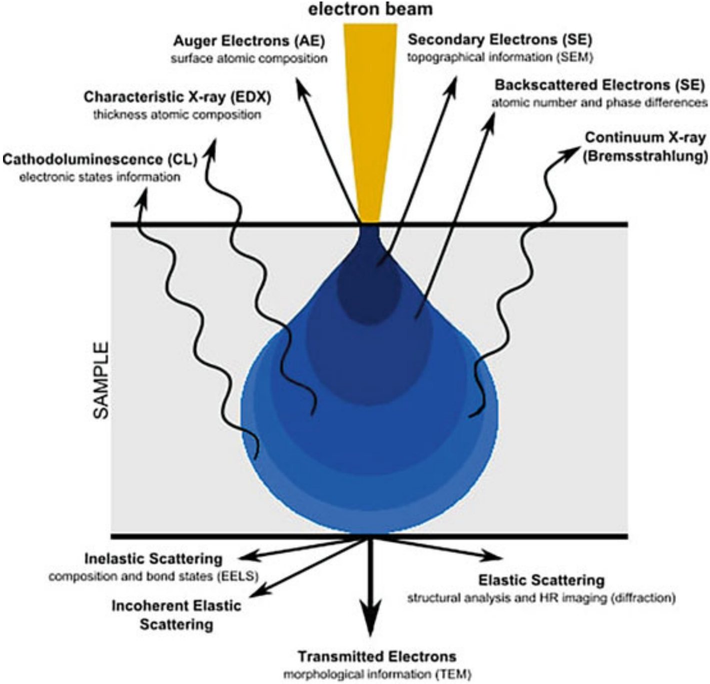


Fig. 16.2 Types of electrons produced and electron physical traits

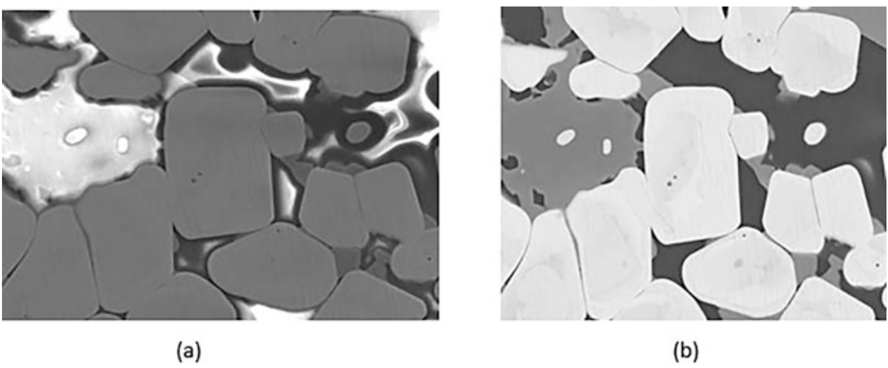


Fig. 16.3 (**a**) Secondary electron image. (**b**) Backscattered electron image (compositional)

- *Backscattering electrons (BSE)*: Some of the electrons from the primary beam may scatter in such a way that they are reflected back off the specimen but do not pass through it. Backscattered electrons are a popular term used to describe them. Since these electrons are from the initial beam, their energy level is close to that of the gun voltage. When displaying information regarding an object's topographical structure and relative atomic density, backscattered imaging mode operation can be helpful as shown in Fig. 16.3.

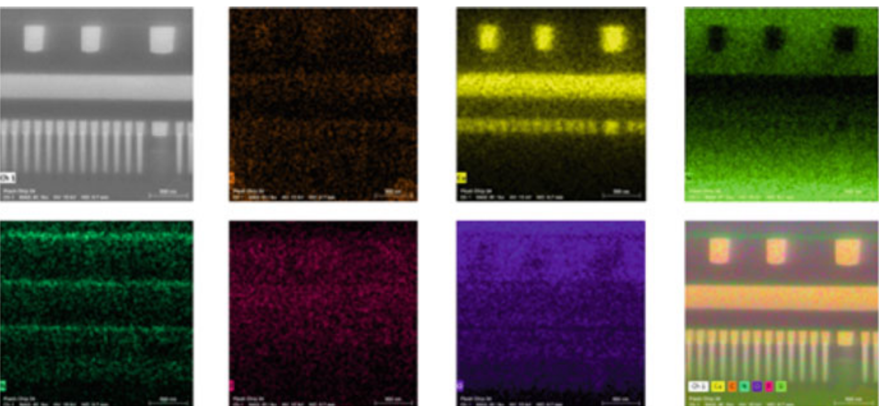


Fig. 16.4 EDS full spectrum imaging of a flash IC

- *Secondary electrons (SE)*: The secondary electron imaging mode may be the most often employed imaging mode. When a primary electron from the electron beam strikes the material and knocks an electron from its surface, secondary electrons are produced. These electrons are formed by secondary electrons as well and have a low energy level of just a few electron volts. They are therefore only detectable when they are displaced near the specimen surface. The sample absorbs back secondary electrons that are produced but are unable to escape. Getting topographical information and high-resolution images are two of the main benefits of using this imaging mode. One great benefit of using this imaging mode is that the images have contrast and soft shadows that seem to be similar to those of a specimen that has been illuminated by light. As a result, the images seem more familiar and are simpler to analyze (Fig. 16.4).
- *X-Rays*: X-rays are emitted from a specimen when electrons are dislodged from particular atomic orbits. An electron from an atom's K, L, M, or N shell gets ejected by a scattering event. To fill the empty space, an electron from an outer shell falls. This energy difference causes the release of an auger electron or an X-ray with a specific energy and wavelength. When X-rays collide with other particles, their energy is lost, changing their wavelength, which causes problems. The necessary amount of energy is lost as the amount of hits increases. As a result, it is impossible to classify these X-rays, and their discovery will be considered as a background noise. X-ray spectrometer detectors measure energy level (energy dispersive spectrometer, EDS) or wavelength (wavelength dispersive spectrometer, WDS). For more information, please read the supplement material, Whitepaper on the Working Principles of Scanning Electron Microscopy by Thermo Fisher [1].

16.2.3 Display and Record System

A micrograph's quality is mostly determined by its contrast, brightness, resolution, magnification, noise, depth of field, and composition, which ultimately determine the quality of an image displayed on a screen.

- *Brightness*. Brightness is a term used to describe the value of each individual pixel that contributes up the image. The brightness of the image increases with higher pixel overall values.
- *Contrast*. The difference between the two pixels is what is referred to as contrast. The difference between the highest and lowest pixel values can be used to calculate the overall contrast.
- *Resolution*. Resolution is the capability to distinguish between two points. The size of the electron beam's spot is the most important factor in determining resolution. Working distance (WD), aperture size, voltage, beam current, and beam shape are additional variables. Poorly defined edge boundaries that cause out-of-focus in a micrograph make it easy to identify those with low resolution.
- *Magnification*. The size of the viewing area and the magnification depend on each other (CRT or Film). The raster coils and the distance from the primary beam's focal point to the final lens are the two elements that govern and modify the magnification. Notably, the sample can be raised or lowered into the primary beam's focal point by adjusting the working distance. When concentrating on a sample for an accurate magnification, this is required. Using an excessively high magnification when taking a micrograph is a common error. Like other forms of photography, SEM microphotography allows for the creation of micrographs at magnifications higher than the resolution limit. The term "empty amplification" describes this. Enlarging an image without including any additional information is known as empty magnification.
- *Depth of field*. The depth of field is the acceptable sharpness region in front of and behind the point of focus. The sample's distance from the final lens determines the depth of field. The working distance can be changed to increase resolution and decrease depth of field by moving the sample closer to the final lens. The farther the specimen is from the final lens, the greater the depth of field and the lower the resolution. The SEM's excellent depth of field makes it simple to create stereo micrographs.
- *Noise*. Any amount of brightness, whether white or black, that is shown in a micrograph but is not due to the intended interaction between the beam and the object is referred to as noise. An illustration of electronic noise is the snow that can be seen on a television tuned to a weak signal. When the signal-to-noise ratio deteriorates, noise becomes more apparent. This occurs when a sizable amount of the signal resulting from the interaction between the beam and the sample comes from the system's electrical noise. The signal-to-noise ratio can be improved in two ways: (1) by boosting the sample signal and (2) by reducing electronic noise. There are many techniques to increase the signal, including expanding the aperture, raising the bias voltage, etc. Increased scan time is a

different approach that hasn't yet been mentioned. The raster rate slows down as the scan time lengthens. This will improve the signal-to-noise ratio and therefore will reduce considerable noise in the micrograph or on the CRT being viewed. The results are as fast as the scan rate when the aforementioned perimeters are adjusted. Therefore, no adjustment is required if the image seems clearly defined on the CRT, although the brightness or contrast may need to be adjusted for the image to be transferred to film or a computer.

16.2.4 Specimen Preparation

The main principle of sample preparation is to touch the sample as minimally as possible while making sure it is the right size, electrically conductive, and stable with the vacuum and has properties close to those of its natural condition. The majority of metallic samples meet these requirements with little to no preparation. Many other materials, such as ceramic, polymers, and minerals, only need to be coated with a conducting metal. It is necessary to coat most nonmetallic specimens with metal to make them conductive. Sputter coating and vacuum deposition are the two most often utilized techniques for coating samples. The coating tool of preference is the sputter coater. From start to finish, coating a sample takes approximately 30 minutes. Metal molecules from various atoms make up the sputtered coating. These molecules have the ability to splatter like paint when they hit the sample. As a result, a structure's underside can have a thin coating applied to it.

16.3 Setting Up the Experiment for Image Acquisition with the SEM

In this section, we will discuss the steps of setting up the experiment for image acquisition with the SEM. The first step includes specimen preparation, and the second step includes the loading of the samples inside the SEM chamber. We will be performing the image acquisition on our TESCAN FERA3 and LYRA3 dual-beam systems. The image acquisition will be performed on the mechanically polished and further FIB-delayered IC samples of different node technologies. The IC samples used in this module include Xilinx FPGA (28 nm) and AMD Opteron (65 nm) as shown in Figs. 16.5 and 16.6. These ICs will already be de-capsulated and will be thinned down by mechanical polishing and FIB delayering to make them ready for the experiment.

Fig. 16.5 AMD Opteron (65 nm)

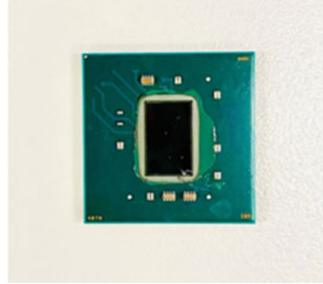


Fig. 16.6 Xilinx FPGA (28 nm)

16.3.1 Sample Preparation

The first step in the process involves sample preparation of the ICs to make them ready for the experiment. This process involves:

- When handling anything that will go into or come into contact with the SEM, always use gloves.
- The specimen should be conductively fixed or glued to a specimen stub (12.5 mm specimen pin-stubs). Carbon tape or copper tape can be used for this process depending on the sample.
- Nonconductive samples need to be coated by a conductive layer using either a carbon coater or sputter coater.

16.3.2 Sample Loading Inside the SEM

Once the sample is fixed over the specimen stub, the next step involves loading the sample inside the SEM chamber.

- To open the microscope's exhaust valve, click VENT. Wait until the venting is complete.
- Once finished, to open the chamber, simply pull the corners in your direction.

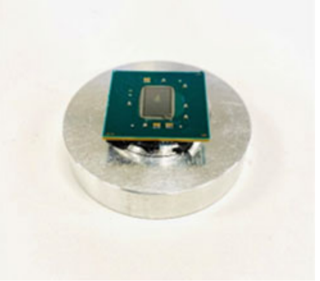


Fig. 16.7 Sample fixed on the stub

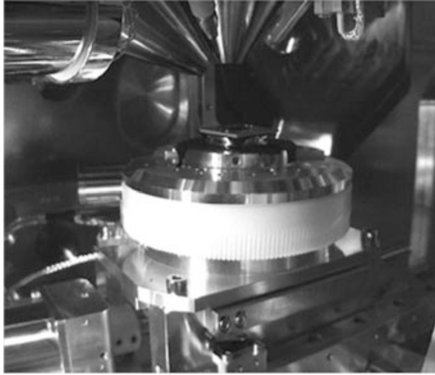


Fig. 16.8 Sample loaded in the SEM chamber

- If necessary, rotate the stage to reach the screw port. Clamp on the specimen stub using the provided tweezers, and then use an air gun to blow an airstream over the entire specimen stub.
- The specimen stub should be gently inserted into the specimen stage after loosening the screw, and the screw holding should then be tightened.
- Close the door of the chamber carefully by pushing it inward, and then press the pump and wait for the bar graph to turn green or to display "VACUUM Ready."

The setup is ready to perform the SEM imaging as shown in Figs. 16.7 and 16.8.

16.3.3 SEM Image Acquisition

To get started, we need to follow the following steps to get the best-resolution SEM images of the samples.

16.3.3.1 Turning on the Electron Beam

1. Click on BEAM ON as shown in Fig. 16.9 from the electron beam panel to turn on the electron beam.

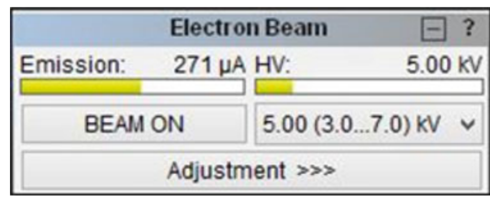
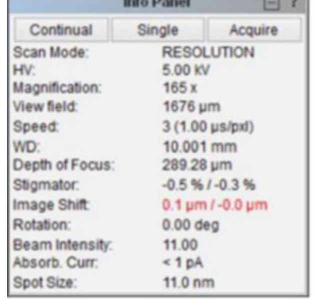


Fig. 16.9 BEAM ON from the electron beam panel



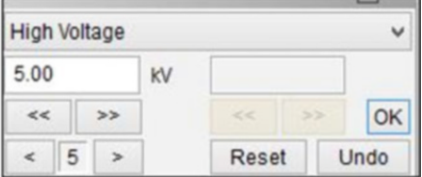


Fig. 16.10 Select HV in pad drop down

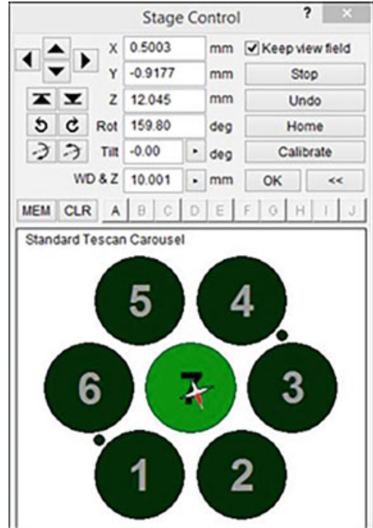
MODE	Characteristics
Resolution	High resolution, Lower depth of focus
Depth	Good resolution, Increased depth of focus
Field	Lower resolution, large field of view, High depth of focus
Wide-Field	Extra-large field of view

Fig. 16.11 Choose desired scanning mode

2. In the Info panel, select HV, or, as illustrated in Fig. 16.10, choose HV from the Pad Drop Down. In the Pad panel, set a specific high voltage (set 5 kV as starting voltage).
3. The Auto Gun Heating option must be used after clicking Adjustment >>> if a black screen appears after turning on HV.

16.3.3.2 Imaging Mode

1. Click MODE and make sure that Continual Wide Field option is checked.
2. After that, choose desired scanning mode (default = Resolution) as shown in Fig. 16.11.



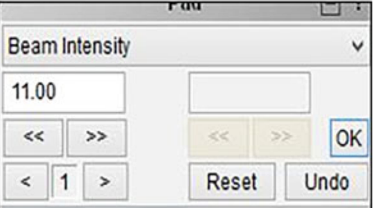


Fig. 16.12 Stage control

16.3.3.3 Beam Intensity, Brightness, and Contrast

1. Using the stage control, center the SEM window on the desired sample, as shown in Fig. 16.12.
2. Now use BI button to adjust the beam intensity using the << and >> as shown in Fig. 16.13.
3. Recommended BI of value 15 to start at low magnification.
4. To automatically correct the contrast and brightness if it is excessively bright or dark, click Auto.
5. To manually adjust the contrast and brightness, click Brightness as shown in Fig. 16.14.
6. Additionally, be sure to click the IR camera button to reveal the chamber's view as shown in Fig. 16.15.

16.3.3.4 Magnification, Focus, and Scan Speed

1. To alter the magnification, click MAG.
2. Change the sensitivity if necessary by moving the trackball from left to right or by simply entering a value in the Pad panel.
3. Click WD to change the focus distance after that. Focus can be changed by sliding the Trackball from left to right. A focused image with a WD value displays the actual working distance.
4. In the SEM scanning window, double-left click to create a Focus Window.

Fig. 16.13 Beam intensity

Magnification	Beam Intensity
Min – 200	13 – 18
200 – 2000	8 – 12
2000 – 10k	7 – 10
>10k	4 – 7

Fig. 16.14 Brightness and contrast adjustment

5. Click SPEED to adjust scan speed.
6. To determine how SPEED and BI will affect the quality of your images, use the Focus Window. It is recommended that the initial focusing BI setting's SPEED of 1–4 corresponds to the MAG value.
7. Higher values of SPEED seem more attractive but take longer to focus. When you're ready to save images, use higher SPEED values of 5–8.

16.3.3.5 Working Distance

1. To achieve the best focus possible, combine focusing and MAG.
2. Ensure that mode is depth or resolution (if not, keep increasing the MAG).

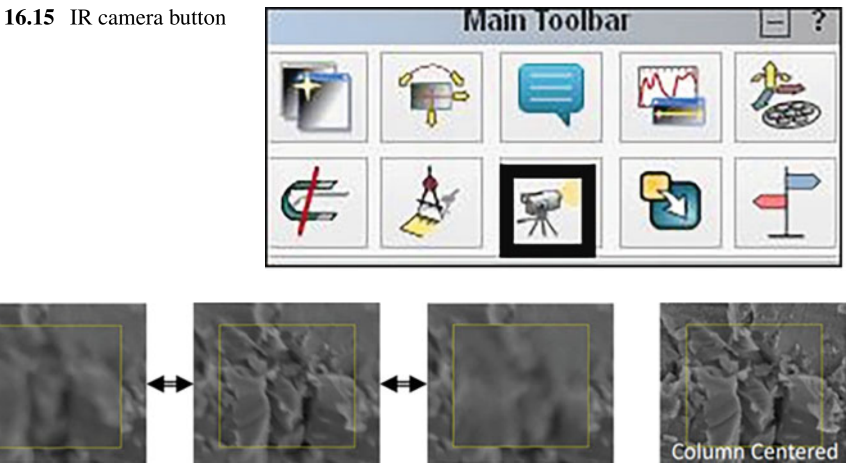


Fig. 16.15 IR camera button

Fig. 16.16 Focus window

3. Identify current WD by focusing on samples at different locations over the specimen depending on the desired location.
4. Always makes sure to keep a safe working distance to avoid any damage to the sample or the SEM column (note: approximately 9 mm for SE and 7 mm for BSE). Note: Imaging at MAG $\geq$ 10 kX requires further optimization steps. As we are going to take the SEM Images at a much higher magnification, column centering and stigmation correction are also required before proceeding to the image acquisition directly.

16.3.3.6 Column Centering (Wobbler Effect)

1. Around a feature of interest, create a focus window. Bring the feature into focus as indicated by clicking WD as shown in Fig. 16.16.
2. When you focus, if the image moves or shifts, you must finish column centering. If the image does not move or shift, proceed on to stigmatization correction.
3. Select Manual Column Centering from the menu. Click Next when the Manual Centering Wizard appears.
4. Now, the focus of your image will “wobble” back and forth. It must be removed if the image wobbles in either the X or Y direction.
5. By adjusting the OBJ Centering with the trackball, you may minimize image movement.

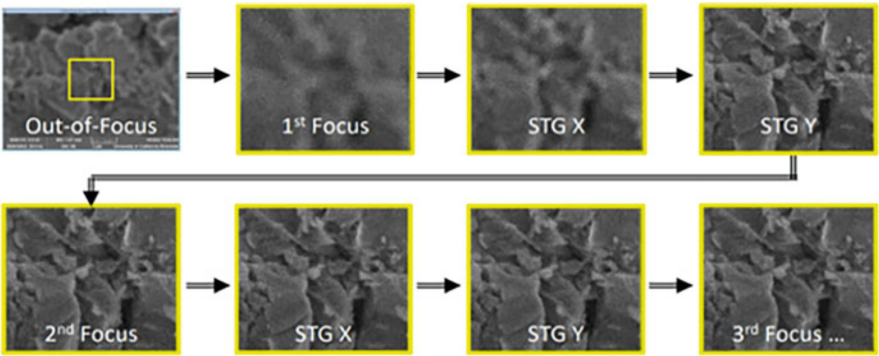


Fig. 16.17 Stigmator component window

6. No X / Y translations should be included; instead, the image should only move in and out of focus. Click Finish.

16.3.3.7 Stigmatism Correction

1. Create a focus window on a feature of interest.
2. To check for streaking on irregular features, click WD and bring the feature in and out of focus (both sides).
3. Any streaks show that stigmatization correction is required. A focused image will become substantially sharper when stigmation is removed, as shown in Fig. 16.17.
4. To make the STG an active function, click it. The trackball should be slowed down for precision close to the "sweet spot" with STG Sensitivity set to 6. Adjust the stigmators one at a time for a sharper image (X and Y).
5. Each stigmator component (X and Y) should be adjusted *carefully and slowly* until the "perfect" or set with the sharpest image can be identified.

16.3.3.8 Image Acquisition

1. Create focus window and achieve the BEST focus (recommend sensitivity = 2).
2. Reset the desired magnification by clicking MAG and entering values such as "desired mag = 10 kX."
3. The focus window should be activated then over the desired feature (a smaller window means a quicker refresh).
4. Choose the speed that corresponds to your image's maximum acquisition time, like for 2 minutes, by typing SPEED = 7.
5. To automatically adjust the brightness and contrast as you modify the BI, select auto.

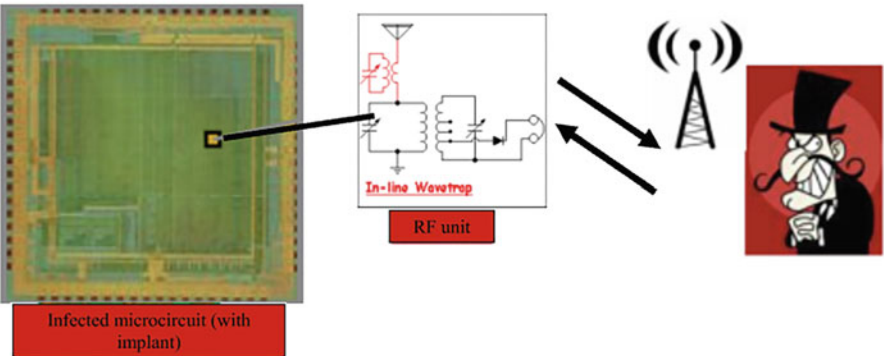


Fig. 16.18 IC level hardware Trojan

6. Increase the image acquisition time (e.g., SPEED = 7 − > 8) if high resolution is desired but there is excessive graininess.
7. Click Acquire to capture image and save it.
8. After image acquisition is done, don't forget to turn off the electron beam.

In the next section, we will discuss in detail how to detect hardware Trojans in ICs using SEM Images.

16.4 Hardware Trojan (HT) Detection in ICs Using SEM Images

Hardware Trojans are malicious alterations to integrated circuits (ICs) designed to jeopardize an electronic system's security and reliability (see Fig. 16.18). This experiment uses advanced nano-imaging and image processing with neural networks to detect hardware Trojans inserted by untrusted foundries. An IC with on-chip trusted test structures (logic cells) with layout OR SEM images of the IC with trusted (golden circuits) and not investigated logic cells will be provided as a starter data set. The on-chip golden circuits provide authentic samples for image-based Trojan detection. The experiment will be performed on a 28 nm backside thinned FPGA.

16.4.1 Equipment and Software Needed for This Work

This experiment involves multidisciplinary knowledge of the following subject and requires the following equipment's, samples, and software:

- IC samples: IC samples under authentication in which presence of hardware Trojan needs to be investigated.
- Golden layout: Trusted Trojan free layout design
- IC sample preparation instrument: Allied X-Prep
- Dual-beam SEMs: An SEM (scanning electron microscope) with FIB (focused ion beam)
- Software: Anaconda environment that supports Python v 3.9 or later running on a Windows-/Mac-based machine.

16.4.2 Prerequisites

- Fundamental knowledge of semiconductor device layer and packaging.
- IC sample preparation (optional if polished IC sample is available).
- Scanning electron microscope training (see Sect. 16.3).
- Fundamentals of image processing and data science using Python.

16.4.3 Experimental Setup for HT Detection in ICs Using SEM

In this section, we will discuss the steps of setting up the experiment for HT detection in ICs using SEM.

- Sample Preparation Station (Optional)
- SEM imaging station (See Sect. 16.3)
- A computer setup with anaconda Python 3.9 or later environment with OpenCV, pandas, matplotlib, and scikit-learn Python packages installed.

16.4.3.1 Procedure

This experiment will be performed in the following steps as sub-modules:

- Sample preparation (optional, if the sample is ready).
- FIB and SEM Imaging for data image collection.
- Trojan detection system using image analysis and artificial intelligence.

16.4.3.2 Sample Preparation

Decapsulation When the die is decapsulated, internal die, lead frame, and die connection components, such as bond wire and ball grid arrays, are all made visible. Techniques for decapsulation that are nonselective include mechanical polishing and CNC multi-tool machining. Even after revealing the bare die, SEM imaging is

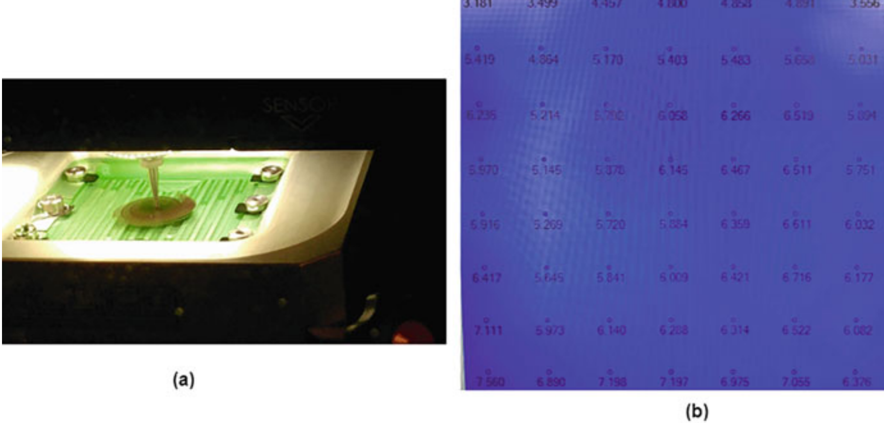


Fig. 16.19 (**a**) Chamber view of allied X-prep (**b**) 2D thickness measurement mapping after polishing

impossible because electrons can't penetrate through a thick silicon substrate layer of the sample.

The substrate must be thinned even more using precise polishing techniques. Furthermore, the bare die employed in the process is not flat, and its curvature changes during polishing, potentially resulting in uneven silicon substrate removal over the chip. Advanced sample preparation machine, Allied X-Prep, sophisticated silicon die polishing technology (see Fig. 16.19), can be utilized to accomplish backside thinning of up to 1–2 μm to mitigate these difficulties. Two common selective decapsulation methods are wet etching and plasma etching. We employ a 28 nm FPGA in this experiment. FPGAs are widely used in communication systems and military systems television boxes. By embedding a Trojan in the circuitry, an adversary can acquire sensitive or confidential information, cause a data breach, and inflict financial loss on an entity.

An FPGA die can be a flip chip or enclosed in a mold epoxy resin. The first step in decapsulating (if packaged) the FPGA chip to expose the die by grinding the top surface followed by the exposed die can be polished and thinned down to less than 1 μ by using precise polishing.

16.4.4 FIB and SEM Imaging

16.4.4.1 FIB Delayering

For SEM imaging and analysis, the silicon substrate needs to be thinned below 1 μ. The advanced polishing machines cannot be used for thinning sub-micron, so

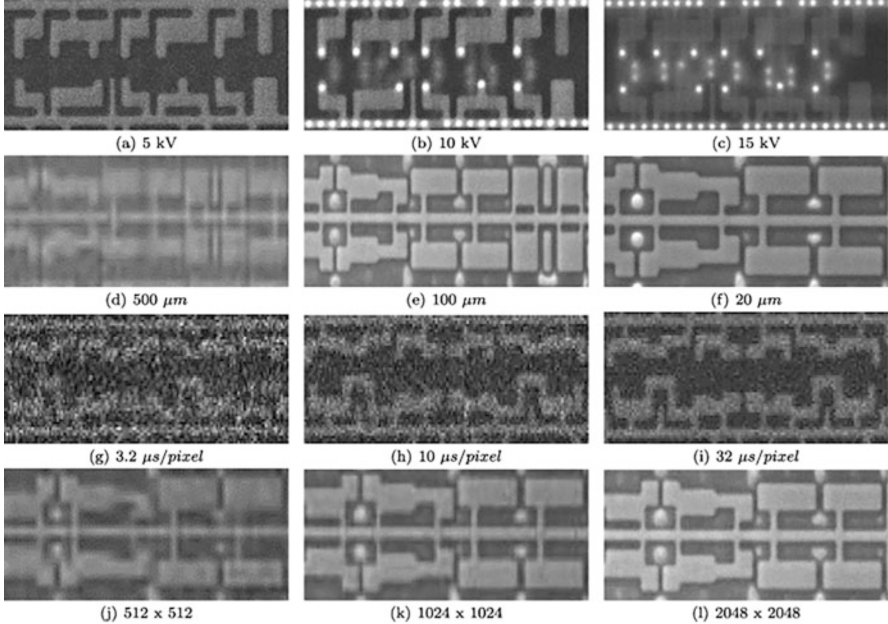


Fig. 16.20 SEM image variations with different beam voltages [(**a**), (**b**) and (**c**)], field of views [(**d**), (**e**), and (**f**)], dwelling times [(**g**), (**h**), and (**i**)], and resolutions [(**j**), (**k**), and (**l**)]

a plasma-assisted focused ion beam will be used for further silicon thinning. In this step, a trainee has expected to finished the experiment as in Sect. 16.3.

16.4.4.2 SEM Imaging

High-resolution images from the FPGA are needed and can be taken with a scanning electron microscope (SEM). The goal is to scan the entire die as quickly as possible while obtaining enough feature details to compare with the SEM image of authentic logic cells. The following SEM parameters influence the timing and quality of SEM images. By altering one parameter at a time while leaving all other factors constant, we can see the effect of one parameter on SEM images (see Fig. 16.20).

- Beam voltage – The depth of an item's penetration by electrons is determined by the electron beam's accelerating voltage (measured in kV). A 5 kV beam can reveal active regions when imaged from the backside, but a 10 kV beam can reveal additional sub-surface characteristics, such as the polysilicon and first few metal layers.
- Field of view (FOV) – The magnification of the image is inversely proportional to the field of view. Because of the low magnification, a large field of view covers

more features, but they are fuzzy. With a smaller field of view, imaging duration increases.

- Dwelling time (speed) – A higher dwelling time improves the signal-to-noise ratio of the image, leading to better SEM image quality, but also lengthens the image capturing process. It also has an impact on surface charge, which can cause artifacts while imagining.

The microscope can also be configured to scan the entire die in the form of small windows of images, which are then stitched together to make a complete panorama image when the abovementioned parameters have been specified. Images captured with a large field of view and a short dwell period take less time to image, but their quality is not good. A small field of view, extended dwell time, and high-resolution capture more excellent image quality, but it collects more data than necessary and lengthens the imaging process. As a result, there is a trade-off between imaging time and image quality in order to achieve higher Trojan detection findings. SEM parameter optimization is done to balance time consumption and detection confidence.

16.4.5 Trojan Detection System

In this section, we are going to design an end-to-end real-time trojan detection system. Our computer vision-based approach consists of logical cell detection and a cell recognition unit. A schematic diagram of the system is presented in Fig. 16.21.

Each SEM image is pre-processed and then analyzed to separate out cell rows. Cell images are extracted from each of those rows. Extracted cell images are passed through the cell recognition unit. Based on the output of the cell recognition unit and the corresponding entry in DEF file, possible Trojan presence is decided as mentioned in Fig. 16.21.

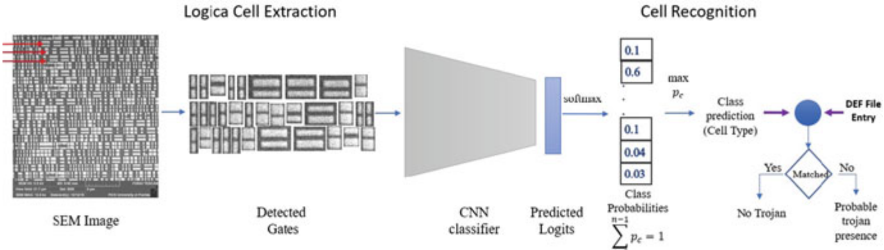


Fig. 16.21 Full system architecture

16.4.5.1 Pre-processing

By isolating out the pixels with the same intensity, components of an image are detected. Each image is binarized such that the foreground pixels and background may be easily distinguished. To avoid noisy binarization, the image is denoised beforehand. Denoising the input image is the first step in our pre-processing stage, which is followed by binarization and the grouping of foreground or gate regions using connected component analysis. (see Fig. 16.22).

Denoising In order to denoise images, nonlocal pixel methods are used. To determine the value of the target pixel to smooth the image, similar patches are discovered throughout the image and averaged out instead of averaging out a group of pixels (let's say a 5 × 5 area) surrounding the target pixel. To obtain the denoised image, the entire procedure and parameter settings outlined in [3] are applied to the input image I, I_D (Fig. 16.22b).

$$I_D = \text{FastNonLocalMeans}(I)$$

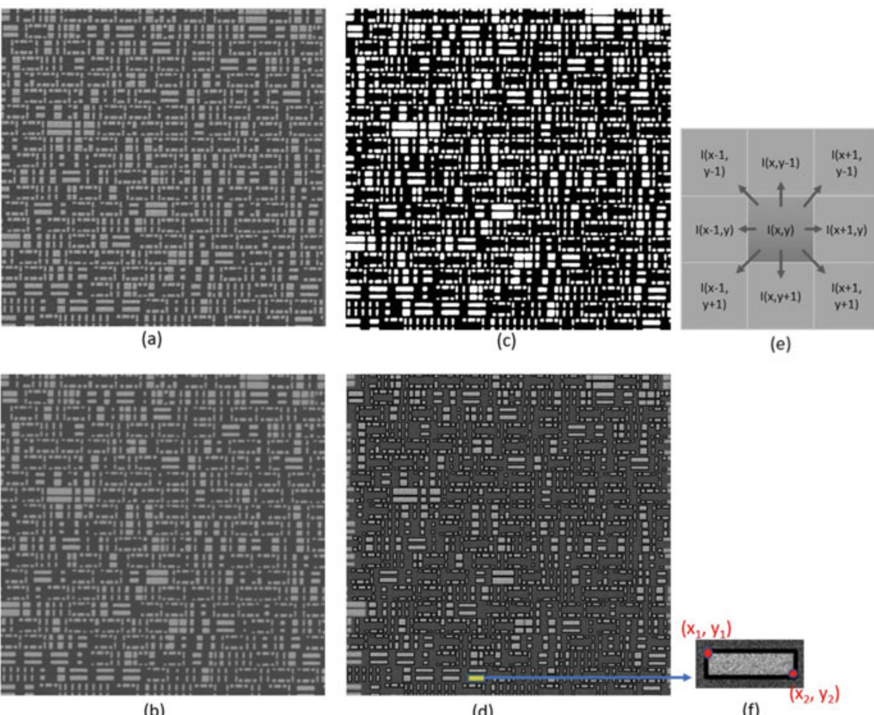


Fig. 16.22 Pre-processing stages. (**a**) An SEM image of dwelling time 5. (**b**) Denoised image. (**c**) Binarized image. (**d**) Connected components. (**e**) 8-connectivity of a pixel with intensity $I(x, y)$. (**f**) A component

Binarization To prevent disturbance brought on by potential salt-and-pepper noise in I, the denoised image ID is used to generate the binarized Image I_B (Fig. 16.22c). We investigated both globally and locally adaptive thresholding methods to figure out the optimal threshold for binarization. Otsu's method [6] is used to determine the global threshold. Fast binarization is followed as in [7] for local thresholding, which entails thresholding individual portions of images.

$$I_B = \text{Binarization}(I_D)$$

Components Gate regions in the foreground pixels (pixels with value 255) are grouped using components from the binary image I_B (pixel value set, $P_B = \{0, 255\}$). From top to bottom and left to right, I_B is scanned pixel by pixel, and adjacent pixels that share the same intensity values are grouped together. Different connection measurements are used in the labeling process. In our scenario, grouping is done by taking into account each pixel's 8-connectivity (Fig. 16.22e). When calculating related components, the background pixel was ignored. This stage results in a collection of explicitly labeled connected components, $C = \{c_0, c_1, c_2, \ldots\ldots, c_{T-1}\}$ (Fig. 16.22d) where each c_i denotes a rectangular component region with top-left point (x_1, y_1) and bottom-right point (x_2, y_2) as shown in Fig. 16.22f and T represents a total number of components in the image I. Cs are sorted vertically by y_1 (Fig. 16.23).

16.4.6 Cell Extraction

Each image I along with its connected component list C are used to extract out cell images. Image I is divided into N rows $R = R_1, R_2, R_3, \ldots\ldots, R_N$ where each R_i represents a row of gate (see Fig. 16.24). The experimental files of cell extraction can be found at http://cad4security.org/index.php/trainings/hsl/ch16_sem/.

This row division can be done by first vertically sorting the component list C and then comparing y coordinates between each consecutive component.

Cells can be made of single or multiple entities (Fig. 16.24). So we need to check whether any component is itself a cell or part of a cell. Rs are individually scanned to extract the cells. As shown in Fig. 16.24b(i)), each R is made up of two vertically symmetric parts, or Ps. For convenience, we chose one of the Ps (see Fig. 16.24b(ii)). In order to estimate the distances D between subsequent components, the component list $C_i = c_0, c_1, \ldots\ldots, c_{m-1}$ associated with row P_i is scanned from left to right (Fig. 16.24b(ii)) where

$$D = \{d_1, d_2, \ldots\ldots, d_{m-1}\}$$
$$d_j = |c_j.x_1 - c_{j-1}.x_2|; \qquad 1 \le j \le m-1$$

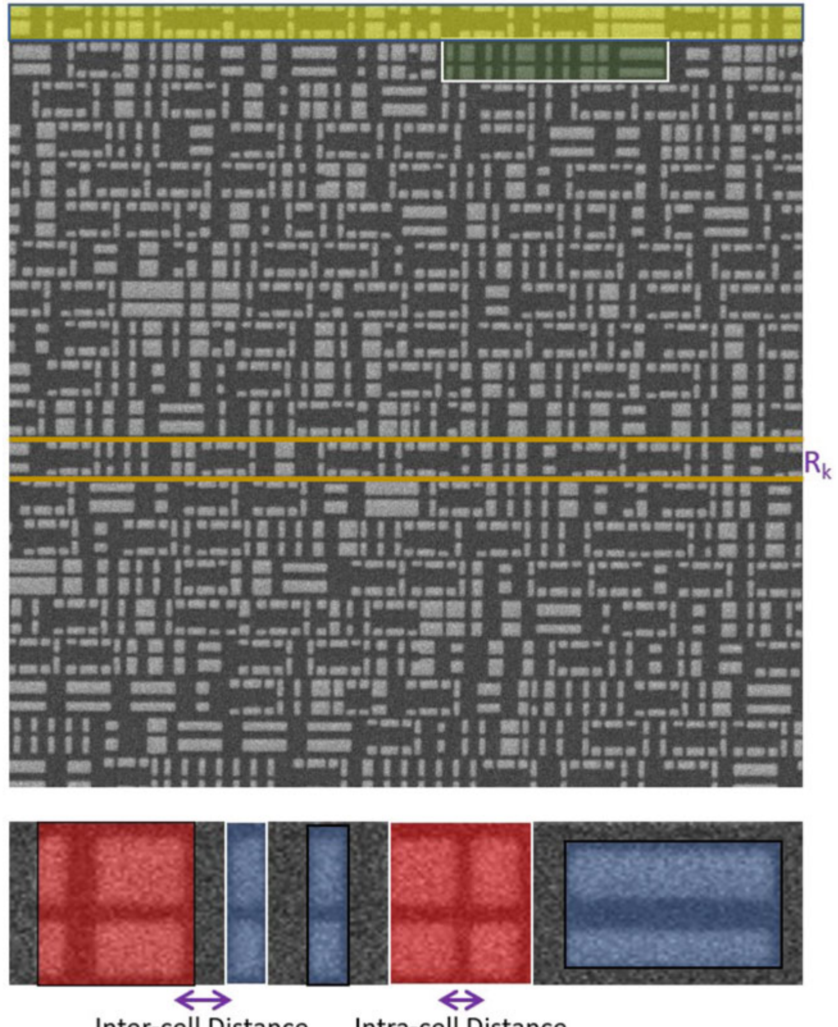


Fig. 16.23 Top – An SEM image, a bottom-green highlighted portion from top image with single (blue shaded) and composite cells (red shaded) marked

Two consecutive components are merged if their distance is less than a certain threshold. That is,

$$c_j = \text{merge}(c_{j-1}, c_j) \qquad if\ d_j < \text{threshold}$$

In our experiments, we set the threshold to 9 pixel but it differs from image to image. A global threshold can be determined by analyzing gaps from all available images which will be done in the future.

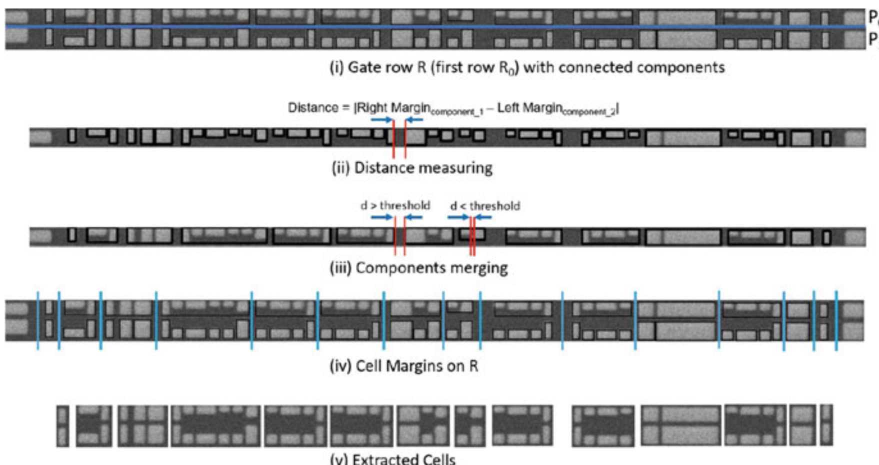


Fig. 16.24 SEM image and cell extraction from its rows

After merging, we obtain margins in P for both composite and individual components. Cells are located and extracted from the relevant gate row R using the same margin (see Fig. 16.24b(iv)–(v)). Cells are then manually annotated. These annotated cells constitute the real image dataset D_R.

16.4.7 Synthetic Cell Image Generation

We must train an effective classifier in order to develop a complete recognition system. To accomplish that goal, a sizable dataset is needed; however, it is not always accessible. A significant constraint that cannot be completely overcome by IC SEM image collection is the need to create a dataset. This procedure involves a variety of specific variables, such as magnification, dwelling time, brightness, and contrast, which can have a detrimental impact on the acquisition time. Additionally, there are variations in the noise, pixel intensity, brightness, and contrast of the IC SEM images. This further limits the possibilities of applying traditional image augmentation techniques together with the imaging differences brought on by the quality of the sample polishing process.

In order to solve the problem of insufficient data, we will create synthetic copies of cell pictures using the retrieved ones. The generative adversarial network (GAN) [2] training setup, which holds the current state of the art in the relevant field, is favored among the various methods of creating synthetic data.

Generative Adversarial Network A generator function called G creates synthetic data in the adversarial setup by mapping random noise variables to data space. Random variables are sampled from a Gaussian distribution in this instance.

$$z \sim N(0, 1)$$
$$I_s = G(z) \tag{16.1}$$

where *N(0,1)* represents normal distribution with zero mean and unit variance.

On the other hand, a discriminative function D acts as a critic and outputs a probability value indicating whether the input is close to real data distribution or not.

$$D(x) = 1 \quad if \quad x \in D_R$$
$$D(x) = 0 \quad if \quad x \in G(z) \tag{16.2}$$

Throughout the learning process, G trains D to produce the most accurate labels for both real and synthetic images. D is therefore taught to optimize expectation for both actual and artificial data.

$$E[D] = \max \quad E_{x \sim D_R}[\log D(x)] + E_{z \sim N(0,1)}[\log(1 - D(G(z))] \tag{16.3}$$

The aforementioned expectation is maximized when D produces values for genuine data samples that are close to 1 and zero for artificial data samples.

G, on the other hand, has been trained to duplicate real images in order to trick the discriminator. A minimal $\log(1 - D(G(z))$ is obtained when D starts producing greater values for synthetic images $G(z)$. For the generator, the subsequent expectation is minimized:

$$E[G] = \min \quad E_{z \sim N(0,1)}[\log(1 - D(G(z))] \tag{16.4}$$

Putting Eqs. (16.3) and (16.4) together, we see that G and D play a two-player minimax game with value $V(D, G)$:

$$\min_G \max_D V(D, G) = E_{x \sim D_R}[\log D(x)] + E_{z \sim N(0,1)}[\log(1 - D(G(z))] \tag{16.5}$$

Choice of GAN Our training scheme for GAN needs to be selected based on two design decisions:

- Images need to be generated class-wise to reduce human efforts of labeling.
- Prevent mode collapse, i.e., refrain generator from generating samples for only one class.

Firstly, to generate synthetic images conditioned on classes, we have encoded class label information with the sampled random variables (see Fig. 16.25) and passed it to the generator. The class conditional formulation (Eq. (16.6)) is followed from the conditional GAN [5].

$$L_{cGAN} = E_{x \sim D_R}[\log D(x, c)] + E_{z \sim N(0,1)}[\log(1 - D(G(c, z))] \tag{16.6}$$

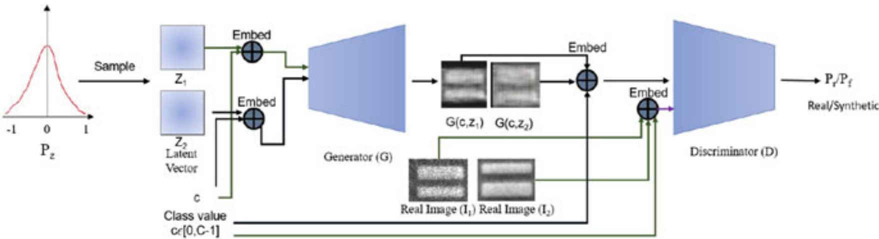


Fig. 16.25 Implementation of MSGAN for our system

Secondly, in case of mode collapse, mapped images are collapsed into a few modes, i.e., two closely sampled latent codes $z1$ and $z2$ are highly likely to be mapped to the same mode of images (Eq. (16.7)):

$$G(c, z_1) \approx G(c, z_2) \text{ where } z_1, z_2 \approx N(0, 1) \text{ and } z_1 \approx z_2 \tag{16.7}$$

To address this issue, the distinctive mapping of two closely sampled random variables ($z1$, $z2$) can be enforced by mode-seeking regularization term (Eq. (16.8)) as in mode-seeking GAN [4].

$$L_{ms} = \max_G \frac{d_I(G(c, z_1), G(c, z_2))}{d_z(z_1, z_2)} \tag{16.8}$$

The overall objective function can be written as follows:

$$L = L_{cGAN} + \lambda_{ms} L_{ms} \tag{16.9}$$

where λ_{ms} controls the weight of the mode-seeking ratio.

Synthetic data D_S generated this way is used alongside D_R to train the CNN classifier.

16.4.8 Logical Cell Recognition

A CNN classifier will be trained on the real (D_R) and synthetic (D_S) datasets. Any popular architecture can be chosen as the backbone of the network. The network should generalize on the logical cell images enough to differentiate even among classes with less inter-class variability.

Two things should be kept in mind while designing the network.

- Attacker can change the gate shape in a way that it may closely resemble one of the existing cells. The classifier should be able to identify if any cell image is even

slightly out of the distribution of known cells. Domain adaptation techniques can be explored to solve this.
- Different cell images come with drastically different aspect ratios. The classifier should be size invariant.

On testing time, we will pass cell images one by one from each location of an SEM image into the classifier. The classifier will generate probability maps over classes from where the class with the maximum probability value will be selected (see Fig. 16.21). The output will be matched with the entry from the DEF file for the corresponding location. Based on the agreement between the two values, the presence of trojan can be detected.

16.5 Conclusion

In this chapter, we worked on TESCAN FERA3 and LYRA3 dual-beam systems. Xilinx FPGA and AMD Opteron are used to perform detailed SEM training. Through this experiment, we will gain a comprehensive understanding of performing high-quality SEM imaging. The purpose of this chapter is for readers to understand how to operate a scanning electron microscope and then how to use electron imaging in a variety of applications. In addition, this work was done to detect malicious modifications by an untrusted foundry inside an IC using Xilinx FPGA. After successfully completing this experiment, readers will understand what hardware Trojan, sample preparation process, and various SEM imaging methods are. Through this module, several algorithms will be developed to automate functions that will increase the productivity of remote imaging.

References

1. Fisher, T.: Whitepaper on the working principles of scanning electron microscopy (2022). https://www.thermofisher.com/us/en/home/global/forms/industrial/sem-working-principle.html
2. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM, **63**(11), pp.139–144 (2020)
3. Karnati, V., Uliyar, M., Dey, S.: Fast non-local algorithm for image denoising. In: 2009 16th IEEE International Conference on Image Processing (ICIP), pp. 3873–3876 (2009). https://doi.org/10.1109/ICIP.2009.5414044
4. Mao, Q., Lee, H., Tseng, H., Ma, S., Yang, M.: Mode seeking generative adversarial networks for diverse image synthesis. CoRR **abs/1903.05628** (2019). http://arxiv.org/abs/1903.05628
5. Mirza, M., Osindero, S.: Conditional generative adversarial nets (2014). arXiv preprint arXiv:1411.1784
6. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cyber. **9**(1), 62–66 (1979). https://doi.org/10.1109/TSMC.1979.4310076
7. Saddami, K., Munadi, K., Away, Y., Arnia, F.: Effective and fast binarization method for combined degradation on ancient documents. Heliyon **5**, e02613 (2019)

Index

A
Advanced encryption standard (AES), viii, 74, 81, 82, 84, 87–90, 96, 107, 109, 111, 113, 120–127, 130–132, 134, 135, 139–162, 165–181, 201–218, 221–234, 280, 287, 290, 291

C
Clock glitch fault attack, 201–218
Correlation, 140, 141, 143, 144, 157, 157, 159, 161, 165–168, 177, 180
Correlation electromagnetic analysis (CEMA), 165–168, 177–180
Correlation power analysis (CPA), 140, 141, 144–145, 157–161, 165, 178
Counterfeiting, vii, 1, 37, 38, 53, 185
Cryptography, vii, 1, 19, 74, 77, 79, 81, 82, 96, 107, 124, 139–144, 167, 180, 201, 207, 222, 237, 240, 261, 262

D
Degradation, 43, 53, 55, 57, 76, 79, 98
Device aging, 44, 49
Device authentication, 2, 3, 15

E
EM analysis, 166, 167, 178
Encryption, 7, 40, 84, 87, 120–127, 139, 140, 142–144, 146, 152, 153, 157, 167–170, 173, 177, 185, 201–203, 206, 210, 215, 216, 218, 222, 222, 225–228, 230–234, 280, 287, 290, 291

F
Fault-injection attack (FIA), viii, 201, 202, 218, 221, 222, 234, 237–257, 275–279, 285, 287
FIA attack scenarios, 238
Field programmable gate arrays (FPGAs), 2, 4–5, 7, 8, 11, 12, 15, 19–21, 23, 24, 26–28, 31, 36, 39–44, 46, 48, 49, 53–71, 74, 79, 80, 82–88, 90, 97, 106, 139, 140, 144–154, 161, 166, 168–170, 173, 177, 180, 181, 188, 189, 202, 204, 208–212, 215, 218, 221–234, 238, 240–246, 248, 261, 262, 267, 268, 270, 275, 276, 279, 280, 282–285, 287, 289, 291, 293, 296, 301, 302, 309, 311, 312, 320
Finite state machine (FSM), 77, 190, 201–218, 222, 224, 226–227, 234, 254, 266, 271
Foundry, 35, 38, 39, 73–77, 95, 97, 166, 185–187, 296, 309, 320
FTC sensor, 277–279, 283–285, 291, 292

H
Hardware security, vii, viii, 90, 139, 197, 208, 227, 237
Hardware Trojan detection, viii, 75, 87, 95–113, 295
Hardware Trojan insertion, viii, 73–90, 96, 97

I
IC backside, 295
IC supply chain, 185, 186, 188
Inspection, 36, 43, 76, 101, 262, 268, 295

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

M. Tehranipoor et al., *Hardware Security Training, Hands-on!*,
https://doi.org/10.1007/978-3-031-31034-8

Integrated circuit (IC), 1, 2, 37, 38, 40, 41, 44, 46, 49, 53, 73, 76, 167, 185, 186, 187, 197, 221, 238, 241, 246, 248, 253, 263, 272, 295, 296, 299, 301, 309, 310, 317, 320
IP piracy, 1, 185, 197
IP protection, 1, 2, 15, 185, 186

K
Key gates, 186, 187, 194–196, 266, 268, 272

L
Laser beams, 238–240
Laser fault injection attack, viii, 237–257
Logic locking, viii, 40, 185–197, 261–272
Look-up tables (LUTs), 40, 41, 44–48, 54–57, 59, 83, 87–89, 124, 143, 188–189, 262, 279

M
Malicious functionality, 74, 76, 95, 98, 100, 101
Metrics, 2, 6–7, 11, 15, 98, 102

O
On-chip sensors, 53
Optical probing, viii, 237, 261–272

P
Physical unclonable functions (PUFs), vii, 1–15, 40

R
Recycled chip detection, vii, 35–49
Recycled FPGA, vii, 36, 39–41, 49, 53–71
Ring Oscillator-based PUF (RO PUF), 2–4, 7–11, 13, 15
Ring oscillators (ROs), vii, 2–4, 8, 15, 19–28, 35–49, 54–57, 59, 63–70, 77, 79, 104–106
RO-based odometer, 35–49
Root of trust, vii
RTL design, 7, 23, 46, 74, 75, 79, 83, 88, 97, 128, 192

S
Satisfiability-based attack, 186
Scanning electron microscope (SEM), viii, 103, 104, 295–320
Secret key generation, 1, 15
Security properties, 96, 101, 102, 109, 111, 117–119, 123, 125–126, 134
SEM imaging, 303, 310–313, 320
Side-channel attacks (SCAs), vii, viii, 7, 104, 139–142, 165–181, 201, 221, 240, 261
System-on-chip (SoC), 4, 11, 42, 73–76, 95, 97, 108, 117, 119–124, 197, 201, 202

T
Tamper-proof memory, 82, 186, 187, 272
Technology-independent TRNG (TI-TRNG), 20, 23, 26
Threat models, 74, 75, 117, 119, 121–123, 125, 130, 134, 135, 187–188
Trojans, viii, 73–90, 95–113, 295, 296, 309–320
True random number generator (TRNG), vii, 19–31, 121, 122

V
Verification, viii, 75, 95, 96, 98, 101–102, 104, 107–109, 112, 113, 117–135, 186, 187
Voltage glitch fault attack, viii, 221–234, 275, 276, 285, 289, 290
Vulnerabilities, vii, 1, 87, 101, 108, 117, 127, 165, 192, 206, 207, 218, 226, 227, 234, 262, 266, 272