UNIVERSITY OF Southampton

University of Southampton Research Repository
ePrints Soton

Copyright © and Moral Rights for this thesis are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given e.g.

AUTHOR (year of submission) "Full thesis title", University of Southampton, name of the University School or Department, PhD Thesis, pagination

http://eprints.soton.ac.uk

UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

School of Electronics and Computer Science

Techniques and Validation for Protection of Embedded Processors

by

Jedrzej J. Kufel

Thesis for the degree of Doctor of Philosophy

May 2015

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
School of Electronics and Computer Science

Doctor of Philosophy

TECHNIQUES AND VALIDATION FOR PROTECTION OF EMBEDDED PROCESSORS

by Jedrzej J. Kufel

Advances in technology scaling and miniaturization of on-chip structures have caused an increasing complexity of modern devices. Due to immense time-to-market pressures, the reusability of intellectual property (IP) sub-systems has become a necessity. With the resulting high risks involved with such a methodology, securing IP has become a major concern. Despite a number of proposed IP protection (IPP) techniques being available, securing an IP in the register transfer level (RTL) is not a trivial task, with many of the techniques presenting a number of shortfalls or design limitations. The most prominent and the least invasive solution is the integration of a digital watermark into an existing IP. In this thesis new techniques are proposed to address the implementation difficulties in constrained embedded IP processor cores.

This thesis establishes the parameters of sequences used for digital watermarking and the tradeoffs between the hardware implementation cost, detection performance and robustness against IP tampering. A new parametric approach is proposed which can be implemented with any watermarking sequence. MATLAB simulations and experimental results of two fabricated silicon ASICs with a watermark circuit embedded in an ARM® Cortex®-M0 IP core and an ARM® Cortex®-A5 IP core demonstrate the tradeoffs between various sequences based on the final design application. The thesis further focuses on minimization of hardware costs of a watermark circuit implementation. A new clock-modulation based technique is proposed and reuses the existing circuit of an IP core to generate a watermark signature. Power estimation and experimental results demonstrate a significant area and power overhead reduction, when compared with the existing techniques. To further minimize the costs of a watermark implementation, a new technique is proposed which allows a non-deterministic and sporadic generation of a watermark signature. The watermark was embedded in an ARM® Cortex®-A5 IP core and was fabricated in silicon. Experimental silicon results have validated the proposed technique and have demonstrated the negligible hardware implementation costs of an embedded watermark.

Contents

Nomenclature xv

Declaration of Authorship xvii

Acknowledgements xix

1 Introduction 1
- 1.1 Motivation for IP Protection . . . 1
- 1.2 Research Focus . . . 2
- 1.3 State-of-the-Art . . . 4
  - 1.3.1 System Architecture . . . 4
  - 1.3.2 Watermark Detection . . . 4
- 1.4 Research Objectives . . . 5
- 1.5 Contributions . . . 6
- 1.6 Thesis Structure . . . 6

2 Literature Survey 9
- 2.1 IP Protection Approaches . . . 10
  - 2.1.1 Deterrent . . . 10
  - 2.1.2 Protection . . . 10
  - 2.1.3 Detection . . . 10
- 2.2 Types of Design Levels . . . 11
- 2.3 Digital Fingerprinting . . . 11
- 2.4 Invasive Digital Watermarking . . . 12
  - 2.4.1 Hard IP . . . 13
  - 2.4.2 Firm IP . . . 14
  - 2.4.3 Soft IP . . . 17
  - 2.4.4 Summary . . . 19
- 2.5 Non-Invasive Digital Watermarking . . . 21
  - 2.5.1 Power Watermark . . . 21
  - 2.5.2 Threshold-Based Watermark Detection . . . 23
  - 2.5.3 Statistical-Based Watermark Detection . . . 24
    - 2.5.3.1 Differential Power Analysis . . . 24
    - 2.5.3.2 Leakage-Based Differential Power Analysis . . . 27
    - 2.5.3.3 Correlation Power Analysis . . . 28
    - 2.5.3.4 Correlation Power Analysis Watermark Detection . . . 29
    - 2.5.3.5 Null Hypothesis Significance Test Watermark Detection 31
  - 2.5.4 Hardware Trojan Detection . . . 34

2.5.4.1 Principles of Hardware Trojans . . . 34
2.5.4.2 Principles of Trojan Detection . . . 35
2.5.4.3 Ring Oscillator Based Trojan Detection . . . 36
2.5.4.4 Multiple Supply Pad Trojan Detection . . . 37
2.5.4.5 Multiple-Parameter Trojan Detection . . . 38
2.5.5 Summary . . . 41
2.6 Concluding Remarks . . . 43

3 Principles of Non-Invasive Digital Watermarking 45
3.1 Dynamic Power Consumption in Digital Circuits . . . 46
3.2 Power Watermark Architecture . . . 48
3.3 Power Watermark Detection . . . 48
3.3.1 Power Trace Length . . . 49
3.4 Third Party IP Attacks . . . 51
3.4.1 General Watermarking Model . . . 52
3.4.2 Types of IP Attacks . . . 52
3.4.2.1 Removal Attacks . . . 52
3.4.2.2 Ambiguity Attacks . . . 53
3.4.2.3 Key-Copy Attacks . . . 53
3.4.3 Preventing IP Attacks . . . 54
3.5 Summary . . . 55
3.6 Concluding Remarks . . . 56

4 Characterization of Sequences for Cost-Efficient Power Watermark 57
4.1 Watermark Sequences . . . 58
4.1.1 Linear Feedback Shift Register . . . 58
4.1.2 Barker Codes . . . 59
4.2 Pearson Correlation Coefficient Analysis . . . 60
4.3 Simulation Results . . . 64
4.3.1 Maximum Correlation Coefficient . . . 65
4.3.2 Correlation Coefficient Difference . . . 67
4.3.3 Null Hypothesis Significance Test . . . 69
4.4 FPGA Validation . . . 70
4.4.1 Watermark Circuit Architecture . . . 71
4.4.2 Experimental Results . . . 72
4.4.2.1 Repeatability . . . 73
4.4.2.2 Detectability . . . 74
4.5 Silicon Validation . . . 76
4.5.1 Test Chips Overview . . . 76
4.5.2 Watermark Module Architecture . . . 76
4.5.3 Experimental Setup . . . 78
4.5.4 Experimental Results . . . 80
4.6 Area and Power Overheads . . . 80
4.7 IP Attacks and Robustness . . . 81
4.7.1 Tampering . . . 82
4.7.2 Finding Ghosts and Forging . . . 82
4.8 Application Specific Watermark Implementation . . . 86

4.9 Summary . . . 87
4.10 Concluding Remarks . . . 88

5 Clock-Modulation Based Watermark Power Pattern Generation 89
5.1 Clock Modulation Watermarking Technique . . . 90
5.1.1 Clock Gating . . . 91
5.1.2 Clock Modulation Watermark Architecture . . . 93
5.2 Silicon Validation . . . 95
5.2.1 Watermark Sub-Module Circuit Architecture . . . 95
5.2.2 Methodology . . . 96
5.2.3 Experimental Results . . . 96
5.3 Area Overhead Reduction . . . 98
5.4 Improved Robustness . . . 100
5.5 Summary . . . 100
5.6 Concluding Remarks . . . 101

6 Instruction Based Activation of Watermark Power Pattern 103
6.1 Motivation . . . 104
6.2 Proposed Watermark Circuit Instruction Based Activation Technique . . . 104
6.3 Case I: Modulation of ARM® Cortex®-A5 Clock Signal With ARM® Cortex®-M0 . . . 108
6.3.1 System Architecture . . . 109
6.3.2 Experimental Setup . . . 110
6.3.3 Detection Methodology . . . 113
6.3.4 Detection Algorithm . . . 114
6.3.5 Experimental Results . . . 116
6.4 Case II: Modulation of ARM® Cortex®-A5 Top Level Clock Gate . . . 116
6.4.1 Test Chip Overview . . . 116
6.4.2 Watermark Circuit Architecture . . . 117
6.4.3 Experimental Setup . . . 117
6.4.4 Detection Methodology . . . 117
6.4.5 Detection Algorithm . . . 119
6.4.6 Experimental Results . . . 121
6.5 Hardware Implementation Costs Reduction . . . 126
6.6 Improved Robustness . . . 127
6.7 Concluding Remarks . . . 128

7 Conclusions and Future Work 129
7.1 Thesis Contributions . . . 129
7.2 Future Work Directions . . . 132
7.2.1 Improved Instruction Based Activation of Watermark Power Pattern 132
7.2.2 Watermark Detection Through Power Classification . . . 133
7.2.3 Classification of Watermark Activation Instructions . . . 134

A Pearson Correlation Coefficient Analysis 137

B Watermark Circuits Integrated on Test Chips 141
B.1 Test Chips I and II . . . 141

B.1.1 Brief Description . . . 141
B.1.2 Interface . . . 141
B.1.3 Watermark Generation Circuit . . . 142
B.1.4 Watermark Power Pattern Generator . . . 142
B.1.5 Interval Counter . . . 144
B.1.6 Watermark Controller . . . 144
B.1.7 Noise Generator . . . 144
B.1.8 RTL . . . 145
B.1.8.1 Top Level Wrapper . . . 145
B.1.8.2 Watermark Generation Circuit . . . 151
B.1.8.3 Watermark Power Pattern Generator . . . 157
B.1.8.4 Interval Counter . . . 160
B.1.8.5 Watermark Controller . . . 162
B.1.8.6 Noise Generator . . . 166
B.2 Test Chip III . . . 171
B.2.1 Brief Description . . . 171
B.2.2 Watermark Generation Circuit . . . 171
B.2.3 Watermark Controller . . . 171
B.2.4 RTL . . . 172

References **181**

List of Figures

1.1 Design-reuse methodology. . . . 2
1.2 Impact of design-reuse on SoC design costs [7]. . . . 3
1.3 Block diagram of the current state-of-the-art digital watermark circuit. . . . 4

2.1 Design flow levels, based on [1]. . . . 11
2.2 4 NAND gates circuit based on a static current consumption measurements [17]. . . . 12
2.3 (a) Buffer insertion technique design flow [18]. (b) An example of buffer insertion: a) original design, b) added buffers, c) watermarked design. . . . 13
2.4 Digital watermarking of a partitioning solution [25]. (a) Original solution. (b) Solution with an embedded public and private watermark signatures. 14
2.5 Digital watermarking of a graph coloring solution [25]. (a) Original graph and host for a public watermark. (b) Watermark signature '00'. (c) Watermark signature '01'. (d) Watermark signature '10'. (e) Watermark signature '11'. . . . 15
2.6 Watermarked (a) and non-watermarked (b) design with path timing constraint technique [19]. . . . 16
2.7 Original (a) and watermarked (b) state transition graphs in a FSM watermarking [34]. . . . 17
2.8 Schematic diagram of watermark generation and test circuits embedded in a soft IP [46]. . . . 18
2.9 Original VHDL code (a) and additional "case" statements (b) in a memory structure watermarking [49]. . . . 19
2.10 (a) Image of a 130nm chip with an optical imaging (top) and scanning electron microscope imaging (bottom). (b) Chip annotation after SEM imaging [6]. . . . 20
2.11 Signal wire removal with a FIB system [57]. . . . 21
2.12 (a) Power watermark block diagram [58]. (b) Threshold-based watermark detection algorithm [58]. . . . 22
2.13 Resonance effect caused by a watermark signature generation [58]. . . . 23
2.14 Power signal of a DES device during cryptographic operation [61]. . . . 24
2.15 a) DES encryption algorithm. b) DES round function. . . . 25
2.16 DPA applied to a smart card [61]. . . . 26
2.17 CPA applied to a cryptographic core on FPGA [73]. . . . 29
2.18 (a) Spread-spectrum watermark architecture [8]. (b) Input modulated watermark implementation [74]. . . . 30
2.19 The spread spectrum of the CPA watermark detection. . . . 31
2.20 Correlation distributions of the *null* (left) and *alternative* (right) hypotheses in the Null Hypothesis Significance Test [73]. . . . 32

2.21 Null Hypothesis Significance Test error types: (a) Type I, (b) Type II. . . 33
2.22 (a) Voltage drop caused by trojan circuit insertion. (b) Impact on clock cycles difference by an addition of a trojan circuit [81]. 36
2.23 (a) Ring oscillator network implemented on an FPGA. (b) 5-stage inverter ring oscillator [81]. 37
2.24 (a) Block diagram of the measurement setup for multiple power supply pin trojan circuit detection. (b) Regression analysis with uncalibrated (left) and calibrated (right) signals for trojan circuit detection [84]. 38
2.25 I_{DDT} vs F_{max} plot for inter (left) and intra (right) measurement variations [87]. 39
2.26 Experimental results of (a) 16-bit sequential trojan with a single side-channel parameter (left) and multi-channel parameter (right) detection schemes, and (b) 4-bit sequential trojan with a 2% (left) and 1% (right) trend lines [88]. 39
2.27 Trojan circuit detection with a multi side-channel parameters [88]. 40
2.28 Regional RON calibration for trojan circuit detection [89]. 41
2.29 3D signature of the smart card processor obtained with an electromagnetic field attack [92]. 42

3.1 a) CMOS inverter. b) CMOS inverter switching voltage (top) and current of PMOS (middle) and NMOS (bottom) transistors [96]. 46
3.2 (a) Architecture of a power watermark circuit; (b) Simulation results of the effect of the watermark power signal on the total device power. . . . 48
3.3 The measured clock (top) and device power (middle) signals with the sampling frequency of oscilloscope much higher than the frequency of the system. The averaged device power (bottom) represented as a single value per system clock cycle. 49
3.4 Power trace length impact on correlation coefficients. 50

4.1 (a) Block diagram of the 12-bit LFSR; (b) States of LFSR in consecutive clock cycles. 58
4.2 Influence of *activity factor*, α, and *overlapping factor*, θ, on (a) maximum correlation coefficient, ρ_{PEAK}, and (b) correlation coefficient difference, ρ_{DIFF}. 62
4.3 Simulation results of watermark sequences comparison, based on maximum correlation coefficient, ρ_{PEAK}. 64
4.4 Frequency spectra of (a) noise β, (b) 2-bit Barker code and (c) 6-bit m-sequence. Convolutions of (d) 2-bit Barker code and (e) 6-bit m-sequence with noise β. Frequency spectra of convolved (f) 2-bit Barker code and (g) 6-bit m-sequence with noise β. 66
4.5 Simulation results of watermark sequences comparison based on coefficient difference, ρ_{DIFF}. 67
4.6 Spread spectra of (a, c) 12-bit m-sequence and (b, d) 11-bit Barker code for noise with power of $6dB$ and $40dB$, respectively. 68
4.7 Null Hypothesis Significance Test of simulated watermark sequences with 5% significance level. 69
4.8 Architecture of a 6-bit m-sequence power watermark circuit implemented on FPGA. 70
4.9 Voltage measurement at the back of the FPGA board. 72

4.10 FPGA experimental results of ρ_{PEAK}. 73
4.11 Box plots of ρ_{DIFF} at various sizes of WPPG circuit on FPGA: (a) 64 SRL16, (b) 32 SRL16, (c) 16 SRL16, (d) 8 SRL16. 73
4.12 Null Hypothesis Significance Test of watermark sequences implemented on FPGA, with 5% threshold level. 74
4.13 The layout (middle), die photo (left) and the hard macro watermark module ('W'), highlighting the watermark circuit (top right) and noise generator (bottom right), integrated in chip I. 75
4.14 Layout (middle) and die photo (left) of test chip II, with the soft macro watermark module embedded in the SoC, highlighted in yellow (right). 75
4.15 Schematic diagram of the watermark module embedded in test chips. . 77
4.16 Silicon chips test board. 78
4.17 Box plots of correlation coefficient difference, ρ_{DIFF}, representing the influence of run-to-run and and process variation on watermark sequence correlation results in test chips: (a) chip I, (b) chip II. 79
4.18 Implementation (a) and detection (b) diagrams of secure digital signature. Detection of a correct (c) and an incorrect (d) signature bit. Correct (e) and an incorrect (f) detection of a digital signature. 84

5.1 Architecture of the current state-of-the-art power watermark circuit. . . 90
5.2 Timing diagram of current state-of-the-art (middle) and clock modulation (bottom) watermark architectures. 90
5.3 Clock tree of hard macro watermark module integrated on chip I. 91
5.4 Architecture of the (a) latch-free and (b) latch-based clock gates and (c) timing diagrams [112]. 92
5.5 Clock tree connections for (a) high, and (b) low fan-out settings [118]. . . 93
5.6 Architecture of the proposed clock modulation power watermark circuit. 94
5.7 Schematic diagram of the clock modulated watermark circuit embedded in test chips. 95
5.8 Spread spectra of correlation results from test chips. Chip I with active (a), and inactive (b), watermark circuit. Chip II with active (c), and inactive (d) watermark circuit. 96
5.9 Box plots of correlation coefficient results from chip I (a) and chip II (b), when the experiment was repeated 100 times. 97

6.1 Architecture of the proposed technique for clock modulation based watermark power pattern generation with instruction based activation. . . 105
6.2 Watermark activation techniques. 106
6.3 Timing diagram of the proposed technique for clock modulation based watermark power pattern generation with instruction based activation during the WFI pipeline flush (middle) and immediately after WFI pipeline flush (bottom). 107
6.4 Block diagram of chip II implementation for the modulation of Cortex®-A5 with Cortex®-M0. 109
6.5 a) Block diagram of clock signal distribution on chip II. b) Timing diagram of clock signals on chip II. 110
6.6 Timing diagrams of the modulated Cortex®-A5 clock signal, depicting the beginning of the program (a) and a single period of a watermark sequence (b). 112

6.7 Representation of the averaged power signal (bottom), from the captured clock (top) and power (middle) signals in chip II. 113
6.8 (a) Results of ρ_{DIFF} while varying the mask rotation (chip I). Spread spectra when X_M and Y are in phase (b) and are not in phase (c). 115
6.9 The layout (left) of chip III and ARM® Cortex®-A5 integrated on the chip (right), with the highlighted watermark circuit. 117
6.10 The effect of WFI instruction on a processor dynamic power consumption. 118
6.11 Power signals obtained from oscilloscope (reduced to processor switch point from an active mode to sleep mode). 118
6.12 (a) Synchronized power traces. (b) Synchronized power traces after application of moving average technique. 120
6.13 Power vectors extracted from consecutive columns in a power trace matrix. (a) Power vector Y_1. (b) Power vector Y_2. 121
6.14 Power signal moving average for (a) mode I, (b) mode II. 122
6.15 Spread spectra of correlation results for 6-bit m-sequence when watermark circuit is active (a) and inactive (b), after the pipeline has been flushed. 123
6.16 Spread spectra of correlation results for 11-bit Barker code active after the pipeline has been flushed. (a),(b) Watermark inactive. (c),(d) Watermark active. 123
6.17 Frequency spectra of power vectors when watermark was off for (a) 11-bit Barker code (single vector), (b) 6-bit m-sequence (single vector), (c) 11-bit Barker code (all vectors), (d) 6-bit m-sequence (all vectors). Convolution of power signal with watermark for (e) 11-bit Barker code, (f) 6-bit m-sequence. Frequency spectra of (g) 11-bit Barker code, (h) 6-bit m-sequence. Frequency spectra of power vectors when watermark was on after the pipeline flush for (i) 11-bit Barker code, (j) 6-bit m-sequence. 124
6.18 Frequency spectra of the convolved power signal with the watermark sequence for (a) 11-bit Barker code (watermark off), (b) 6-bit m-sequence (watermark off), (c) 11-bit Barker code (watermark on - after) and (d) 6-bit m-sequence (watermark on - after). 125

7.1 2-D Feature space. Watermark active (blue) and off (red). (a) All repetitions. (b) Repetitions averaged. 134

List of Tables

4.1 Barker codes [103, 104]. 59
4.2 Parameters of Watermarking Sequences. 63
4.3 Area of Watermark Circuit Implemented on FPGA. 71
4.4 Area and Power Reduction in ASIC 80

5.1 Power Consumption of Placed and Routed WPPG Circuit 98
5.2 WPPG Circuit Implementation Costs 99

B.1 Watermark Circuit Working Registers (Chips I and II) 142
B.2 Watermark Circuit Working Registers (Chips I and II) 143
B.3 Watermark Circuit Working Registers (Chips I and II) 172

Nomenclature

ADC	Analog to Digital Converter
ADP	ASCII Debug Protocol
AHB	Advanced High Performance bus
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CPA	Correlation Power Analysis
DES	Data Encryption Standard
DPA	Differential Power Analysis
DRC	Design Rule Check
EDA	Electronic Design Automation
EM	Electromagnetic
FIB	Focused Ion Beam
FSM	Finite State Machine
HDL	Hardware Description Language
HW	Hamming Weight
I/O	Input/Output
ICID	Integrated Circuit Identification
IEU	Integer Execution Unit
IP	Intellectual Property
IPP	Intellectual Property Protection
KL	Karhunen-Loève
LDPA	Leakage-based Differential Power Analysis
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
LVS	Layout Versus Schematic
NHST	Null Hypothesis Significance Test
OOK	On-Off Keying
PCB	Printed Circuit Board
PRNG	Pseudo Random Number Generator
PUF	Physically Unclonable Function
PV	Process Variation
RO	Ring Oscillator

RON	Ring Oscillator Network
RTL	Register-Transfer Level
SCU	Snoop Control Unit
SEM	Scanning Electron Microscope
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SPA	Simple Power Analysis
STG	State Transition Graph
SVM	Support Vector Machine
VCD	Value Change Dump
VSI	Virtual Socket Interface
WGC	Watermark Generation Circuit
WPPG	Watermark Power Pattern Generator

Declaration of Authorship

I, Jedrzej J. Kufel, declare that the thesis entitled *Techniques and Validation for Protection of Embedded Processors* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- Parts of this work have been published as declared on page 57 and page 89.

Signed:.....

Date:.....

Acknowledgements

My deepest thanks go to my supervisors, Prof. Peter Wilson and Prof. Bashir Al-Hashimi, for their guidance and support throughout this research. I am grateful for their encouragement and continuous motivation, which have helped me to withstand the dark days of a Ph.D. student life. It has been a pleasure to work with them.

I would like to thank ARM-ECS Research Center in the School of Electronics and Computer Science, University of Southampton for enabling the collaboration between ARM® Ltd. and ECS and providing state of the art research facilities. I would like to extend my sincere gratitude to Stephen Hill, an Engineering Director at ARM® Ltd., Austin, USA, and Prof. David Flynn, a Fellow at ARM® Ltd., Cambridge, UK, for their invaluable input and support during chip design and allowing me to spend 3 months working at ARM® Ltd., Cambridge office as part of my research. I would like to convey my warm regards to Dr. Paul Whatmough at ARM® Ltd., Cambridge, UK, and Prof. Steve Gunn, for their help and patience in reviewing parts of this research. My heartfelt thanks go to Dr. Jatin Mistry and James Myers at ARM® Ltd., Cambridge, UK, without whom the integration of numerous test chips would not have been possible. My special thanks go to Dr. Reuben Wilcock, for ensuring the delivery of silicon chips.

I would like to take this opportunity to acknowledge my colleagues for their invaluable support and useful discussions throughout my Ph.D. These people include, but are not limited to: Dr. Sheng Yang, Dr. Amit Acharyya, Dr. Mustafa Imran-Ali, Dr. Shida Zhong, Anand Savanth, Matthew Walker, Anup K Das, Domenico Balsamo, Luis Maeda-Nunez and Mauricio Gutierrez Alcala.

Finally, I would like to thank my lovely wife, Aga, my parents, Teresa and Jan, my parents-in-law, Alicja and Stanislaw, my brother and sister, Marek and Anna, and their families, for their love and support. To them I dedicate this work.

Chapter 1

Introduction

"The general infringement of all types of intellectual property (IP) has become a major problem. In 1998, ..., it was estimated that the cost of IP infringement approaches $1 *billion per day. ...* 20% *of all infringements of electronic designs occur at external points of vulnerability, caused by the ease with which end-products can be reverse engineered, copied or simply stolen."*

VSI Alliance[1]

1.1 Motivation for IP Protection

The problem of hardware piracy has become a major issue, with billions of dollars worth of Intellectual Property (IP) being stolen every year. It has now reached the extent that it is an imperative for major technology providers to protect their IP to prevent this. Continuous technology scaling has led to an ongoing reduction in transistor minimum size and therefore the ability to pack more circuitry on a single silicon die [2]. The complexity of circuits containing IP is also increasing and this is a direct cause of the so-called design (productivity) gap, with immense time-to-market pressures and shorter design cycles [3]. It is therefore desirable to source or re-use complex sub-systems from external IP suppliers, such as processors, and integrate them into an Application Specific Integrated Circuits (ASIC) along with any new design blocks. This led to the approach of design reuse being in common use Figure 1.1. Based on a system specification, circuit designers can take necessary components from an in-house IP library or from third party IP providers [4]. To incorporate innovation in successive *system-on-chip* (SoC)

[1]The Virtual Socket Interface (VSI) Alliance supports the needs of the industry for design reuse, and investigates the ways to reduce the technology and business barriers in order to accelerate the industry transformation [1]. The main goal of the VSI Alliance is the specification of a set of hardware and software interfaces, formats, and design practices for the creation of functional blocks that enable the efficient and accurate integration, verification, and testing of multiple blocks on a single piece of silicon [1].

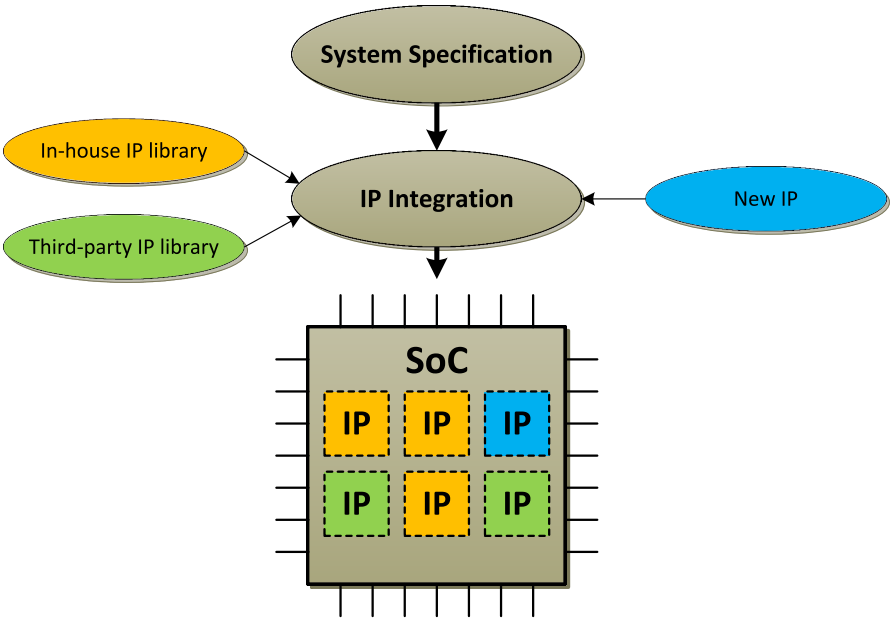


Figure 1.1: Design-reuse methodology.

designs, new IP components can also be created and included using this methodology. With such an approach, designers can exploit this reuse-based methodology to build an IP core in a much more efficient and faster way, than "design-from-scratch" [4]. In Figure 1.2, the conventional "design-from-scratch" register-transfer level (RTL) and reuse-based methodologies are compared, based on the SoC design costs. As can be seen, with an IP reuse-based approach significant costs reduction can be achieved. Nevertheless, the IP sub-systems are often supplied as unprotected design files that SoC integrators can use without any complication of their design flow. Due to the lack of appropriate mechanisms which prevent an access to a design by a third party, the IP protection (IPP) has become critically important as in many cases the value of the IP is significant and relatively unprotected. As a result of theft, cloning and other nefarious techniques, auditing the presence of an IP in finished products is becoming an important and increasing challenge for IP providers to protect their designs. Reverse engineering techniques can be used to prove the presence of specific IP but the process is slow and costly [5, 6]. It is therefore considered desirable to identify and prioritize IP candidates to be short-listed for more thorough investigation.

1.2 Research Focus

The modern electronics market is occupied in a great measure by hand-held mobile devices, such as smartphones or tablets. One of the cornerstones of these devices is the battery life. This causes tremendous pressure on new technologies to optimize the trade-off between cost, performance and power. Embedded processors used in such mobile devices [4], are increasingly constrained in terms of circuit area and power

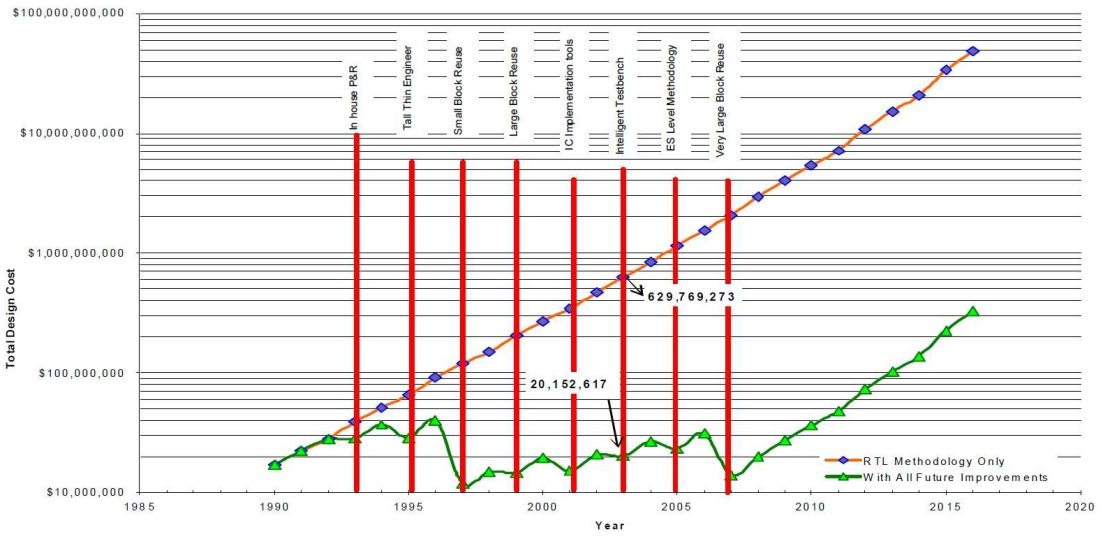


Figure 1.2: Impact of design-reuse on SoC design costs [7].

consumption to satisfy these requirements. IP protection techniques in general increase the hardware implementation costs such as area and power consumption which leads to reduced battery life. Furthermore, the applicability of such techniques is influenced by the design level in which they are implemented. The VSI Alliance [1] proposes several techniques for IP protection at various design levels, such as hard, firm and soft, but they are not all equally applicable. Trade-offs exist between the value (perceived or real) of an IP, difficulty of implementation of the protection scheme, and the resulting usability of the protected IP, by both the integrator and the end user [5]. The use of soft IP is often the most desirable, as it offers IP owners and SoC integrators the greatest level of flexibility. As IP protection techniques are implemented at an early stage of development, the original design flow approach can be retained. The methods for the protection of soft IP can be divided into two groups, where the architecture of a watermark circuit is closely related to the knowledge of a system and detection techniques, namely invasive and non-invasive. Invasive detection techniques require access to device internals, such as input and output (I/O) ports, memories and a full or partial knowledge of a system. Since, the final system architecture may not be known, an IP owner lacks a sufficient knowledge to be able to use such detection techniques and prove an IP infringement. In non-invasive detection techniques, the knowledge of a system's internals is significantly reduced and access to those internals is often not required. Other sources of information, also known as side-channel parameters, such as electromagnetic (EM) field radiation and power consumption can be used to detect an embedded watermark.

Although techniques based on analysis of EM field offer a high degree of detectability, the work presented in this thesis focuses on non-invasive power analysis techniques for identification and protection of embedded processor soft IP cores with the minimization of hardware implementation costs, such as area and power overheads .

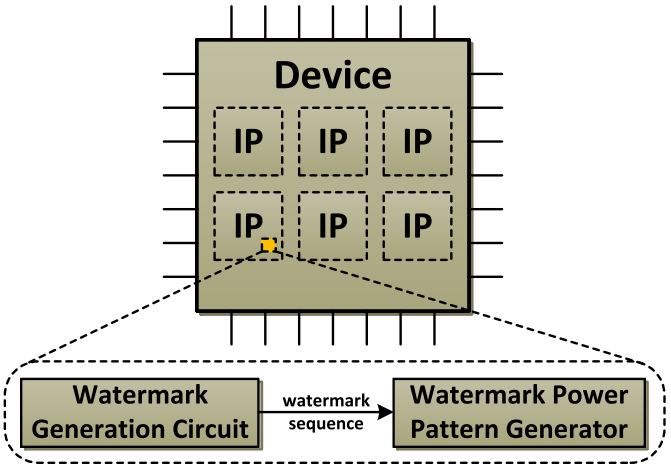


Figure 1.3: Block diagram of the current state-of-the-art digital watermark circuit.

1.3 State-of-the-Art

In recent years, various techniques for the protection of soft IP cores have been proposed through an integration of a digital signature, known as a digital watermark. Such digital signatures are usually a finite sequence of symbols, characteristic of the IP and act as a unique and exclusive identifier [5].

1.3.1 System Architecture

The current state-of-the-art digital watermark technique implements a redundant stand-alone circuit in a soft IP core. Therefore, such circuit has no impact on the functionality of the original IP. In Figure 1.3, a device with numerous soft IP sub-systems is illustrated. The watermark circuit is embedded in one of the IP components and consists of two circuits: the watermark generation circuit (WGC) and the watermark power pattern generator (WPPG) [8]. The WGC generates a digital pattern, known as the watermark sequence. It can be represented as a string of binary data, where each bit acts as a trigger for the WPPG in consecutive clock cycles. For example, when a watermark bit is '1', the WPPG is activated and consumes a significant portion of dynamic power. Otherwise, when a bit is '0', the WPPG is idle and no power consumption occurs. Based on the deterministic charge and discharge pattern of the WPPG logic, the digital watermark signature is generated and can be further subjected to various detection techniques.

1.3.2 Watermark Detection

The watermark circuit generates a specific power pattern which is superimposed on a device power consumption. Such a unique power pattern must be detected by an

IP supplier, to determine if an IP infringement has occurred. The technique known as Correlation Power Analysis (CPA) [9] analyses the dynamic current variations on the supply voltage rail and is used to identify the digital watermark. It is a non-invasive technique and no additional dedicated connections to the input or output (I/O) ports of a device are required. The CPA is the current state-of-the-art detection technique for power watermarks and does not require any knowledge of an original IP architecture. However, due to the nature of the CPA algorithm a significant power consumption is required to generate a strong and detectable signal. Therefore, the current state-of-the-art digital watermark architecture consists of a large redundant WPPG circuit. This causes a considerable area and power overheads and has a significant impact on reduction of a device battery life.

1.4 Research Objectives

The research work presented in this thesis achieves the following:

- **The most cost efficient watermarking sequences in terms of area and power overheads and detection performance.**

  The watermark architecture found in the literature uses maximum length sequences, generated with Linear Feedback Shift Registers (LFSR). The impact of other watermarking sequences on area, power and detection performance has not been investigated and is addressed in this work.

- **Minimization of watermark circuit hardware implementation costs.**

  The current state-of-the-art digital watermark requires an implementation of a redundant and a significant size WPPG circuit, to be detected with the CPA detection technique. This causes a substantial increase in both area and power overheads. Therefore, this work investigates new digital watermark implementation techniques and architectures.

- **Improvement of a digital watermark robustness against unauthorized use.**

  Digital watermark circuits must demonstrate high robustness against various types of IP attacks. The robustness of the current state-of-the-art digital watermark is significantly impaired, due to a large WPPG circuit which leads to its high visibility in the system. Henceforth, the proposed watermark architectures and implementations are analyzed in terms of such robustness throughout this work.

1.5 Contributions

The research presented in this thesis has contributed to the following:

- The characterization of watermark sequences, supported by theoretical analysis and validations through simulations and experiments on FPGA and ASIC, in terms of hardware implementation costs and detection performance.
- Identification of the most cost-efficient sequences for embedded power watermarks in the context of a SoC implementation.
- Clock-modulation based implementation technique, which removes the need of the significant size WPPG circuit, to reduce the area overhead of the power watermark circuit.
- An instruction-based activation of the WGC in a clock-modulation based implementation, for minimization of area and power overheads of the power watermark circuit.
- The first integration of an embedded power watermark in the ASIC implementation.

1.6 Thesis Structure

The thesis is composed of seven chapters. **Chapter 1** presents the motivation and focus of this research and identifies the current state-of-the-art power watermarking technique for the protection of a soft IP. Furthermore, the research objectives are presented and the contributions of this work are summarized.

Chapter 2 provides a comprehensive review of the methodologies for IP protection.

Chapter 3 introduces the principles of digital watermarking and gives an overview of third party IP attacks against watermarks embedded in an IP.

Chapter 4 expands on the fundamentals of the CPA algorithm and presents the characterization of watermark power patterns with their intrinsic parameters. It analyzes various architectures and sequences and demonstrates a strong relationship between the choice of a watermark sequence and the hardware implementation costs and detection performance. The theory is validated with experimental FPGA and ASIC results. Furthermore, the chapter discusses the effect of process variation (PV) on detection performance, in the context of commercial embedded processors in a SoC implementation.

Chapter 5 presents a novel watermark generation technique where clock-gate elements of an existing system are modulated with the watermark sequence. The technique is

validated through experiments on ASIC and demonstrates a significant area reduction, without compromising the detection performance. The improved robustness against IP attacks is demonstrated.

Chapter 6 creates the fundamentals for zero area and power overhead watermark implementation for embedded IP cores. It combines the cost-efficient watermark patterns found in Chapter 4, and reuses the clock-modulation technique introduced in Chapter 5, to activate the watermark generation with specific instructions. The simulations and silicon validations demonstrate an approximately zero area and power overheads, without compromising the detection performance.

Chapter 7 concludes the research work presented in this thesis with concise summarization of contributions and improvements to the current state-of-the-art digital power watermarking techniques and discusses further research directions in the area of power watermarking and IP protection for embedded processors.

Chapter 2

Literature Survey

Intellectual property protection (IPP) has become an important topic with the increasing trend of design reuse approach and modularity of IP blocks. The advantage of these techniques are faster time-to-market and product cycles, however, it poses significant challenges to IC designers. This chapter provides a general overview of the methodologies for IP protection, classified as deterrent, protection and detection, defined by the VSI Alliance [5]. Each protection scheme is analyzed based on its applicability, drawbacks and design tradeoffs in reuse-based commercial "soft" IP designs.

The rest of this chapter is organized as follows. The IP protection approaches are briefly discussed in Section 2.1. It is followed by the summarization of the types of design levels in a design flow in Section 2.2. The digital fingerprinting techniques are discussed in Section 2.3 and the invasive and non-invasive digital watermarking techniques are discussed in Section 2.4 and Section 2.5, respectively. The chapter is concluded in Section 2.6.

2.1 IP Protection Approaches

2.1.1 Deterrent

The deterrent approach provides exclusivity to the IP originator for a specific period of time through patents, copyrights, trade secrets, contracts or lawsuits. It may deter the IP infringement from occurring, however it does not provide any physical protection [5].

2.1.2 Protection

The protection mechanism prevents unauthorized use of an IP and imposes a high degree of security through encryption and licensing agreements. Encryption available in current electronic design automation (EDA) tools is often far from universal and pain-free [5]. A common practice widely adopted in the industry is to encrypt a source code, written in a hardware description language (HDL) such as Verilog or VHDL. However, such an approach may enforce the use of a particular design platform and may not be acceptable to many SoC designers and integrators, who prefer the flexibility of various design tools during the design flow [10]. Another solution is simply to release an IP information in indirect form, such as GDSII, to make masks for the complete chip or in the form of programmable devices, such as FPGAs [5]. Nonetheless, access to a complete design is greatly limited and reduces the marketability of such a product.

2.1.3 Detection

Piracy of an IC occurs at fabrication facilities, due to the full access to a design and sophisticated tools. The protection of an IP can be achieved through an addition of digital fingerprints or digital watermarks. Digital fingerprinting, also known as passive watermarking, is based on the integrated circuit identification (ICID) [11]. A unique ID for each IC is generated and allows a design house to gain a post-fabrication identification of manufactured ICs. Therefore, it prevents a foundry from producing more chips than specified. Digital watermarking, regarded as an active form of IP watermarking, intentionally embeds digital information into an IP for identification purposes, without altering its functionality. The information consists of an author's name, company name or other message closely related to an IP supplier. If necessary, such information can be further used to prove that an IP infringement has occurred. The architecture and therefore the implementation depends however on the design level in which digital watermarks are embedded. Furthermore, the digital watermarking detection techniques can be classified as invasive and non-invasive, and vary based on the resources that are required to obtain a successful detection. In the rest of this chapter, detection techniques are considered.

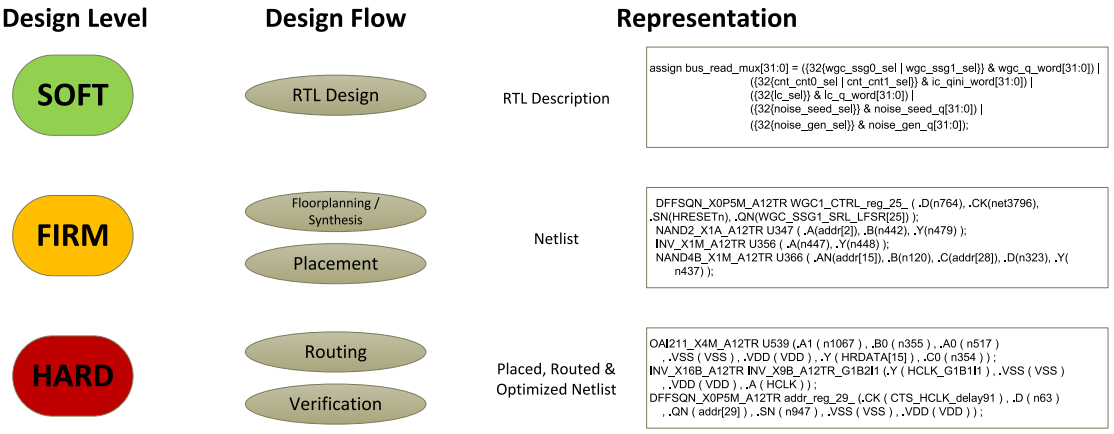


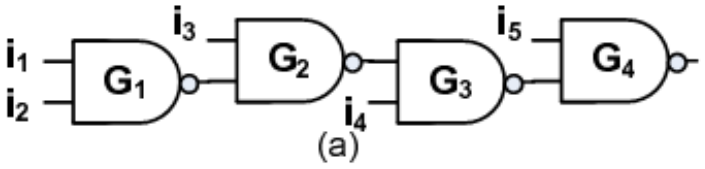
Figure 2.1: Design flow levels, based on [1].

2.2 Types of Design Levels

Intellectual properties can be distinguished in various abstract levels, described also as their "hardness". The VSI Alliance [1] describes such levels as soft, firm and hard (Figure 2.1). Soft IP is delivered as a synthesizable RTL description, written in a hardware description language (HDL). The advantage of a design in this form is its flexibility, however its performance is not fully predictable. Firm IP is delivered as a combination of RTL, technology library, floorplan and a full or partial netlist and is optimized for performance and area. It is more flexible than hard IP and more predictable than soft IP, hence it provides a compromise between both. Hard IP can be delivered as a netlist, fully placed, routed and optimized, a custom physical layout or combination of both. It is the least flexible of all design levels and it is optimized for power, size or performance and is mapped to a specific technology library.

2.3 Digital Fingerprinting

Digital fingerprinting techniques extract unique and usually intrinsic IC features, which are further stored in a database. A comparison of an unknown IC with the database is performed to determine if an IP infringement has occurred. One such feature is an internal IC path (net) timing delay [12–14]. Its variations across the same silicon die are caused by mask variations, while the variations across the wafer (die to die) are introduced by process variation (PV), caused by changes in temperature and pressure during the manufacturing process [12]. To exploit such behaviour challenge-response pairs are generated, based on Physically Unclonable Functions (PUFs). A PUF is a physical function that provides mapping from a device inputs to outputs and it is based on unique fluctuations in the unclonable device material properties. The input to a PUF is typically called a challenge, and the output from a PUF is known as a response. It is expected that a generation of two identical PUFs is technically impossible [15, 16].



(a)

NAND2	
Input	Leakage (nA)
00	37.84
01	100.3
10	95.7
11	454.5

(b)

Gate	Scaling factor	
	IC 1	IC 2
G_1	0.5	2.4
G_2	1.3	0.6
G_3	2.1	4
G_4	3	0.9

(c)

Input vector	Leakage (nA)	
	IC 1	IC 2
00011	1391	2055
10101	2082	1063
01110	1243	2150
11001	1841	1905

(d)

Figure 2.2: 4 NAND gates circuit based on a static current consumption measurements [17].

The input ports of an IC are subjected to a digital or analog signals (challenge), and a specific transient response waveforms are captured at the output ports (response). Each response is unique due to the effect of process variation. The database of challenge-response pairs is further created for future IC comparison. Another IC feature which enables the generation of a unique ID, is static (leakage) current consumption [17]. An example of a four NAND gate circuit is shown in Figure 2.2. Figure 2.2(b) represents the static current of the G1 NAND gate for various input vectors. In deep-submicron technologies the current varies from one gate to another, as shown by the scaling factor in Figure 2.2(c). Finally, Figure 2.2(d) represents the static current consumption of an entire circuit. As can be seen, the current varies between both ICs for the same input vectors. Therefore, each circuit can be uniquely characterized. Figure 2.2 creates the fundamentals for the multi-million gates designs, where the probability of a collision of two identical IDs is negligible [17].

The digital fingerprinting techniques discussed require an access to the entire design flow and a full knowledge of a system. Therefore, they are not feasible for the protection of soft IP, since an IP supplier may not have such knowledge, after an IP has been integrated as a part of a system and the chip has been fabricated.

2.4 Invasive Digital Watermarking

Invasive digital watermarking requires an access to device internals, such as I/O ports, memories and a full or partial knowledge of a system, to detect an embedded watermark. The architecture of a watermark circuit is based on the design level in which such a watermark is embedded (refer to Section 2.2).

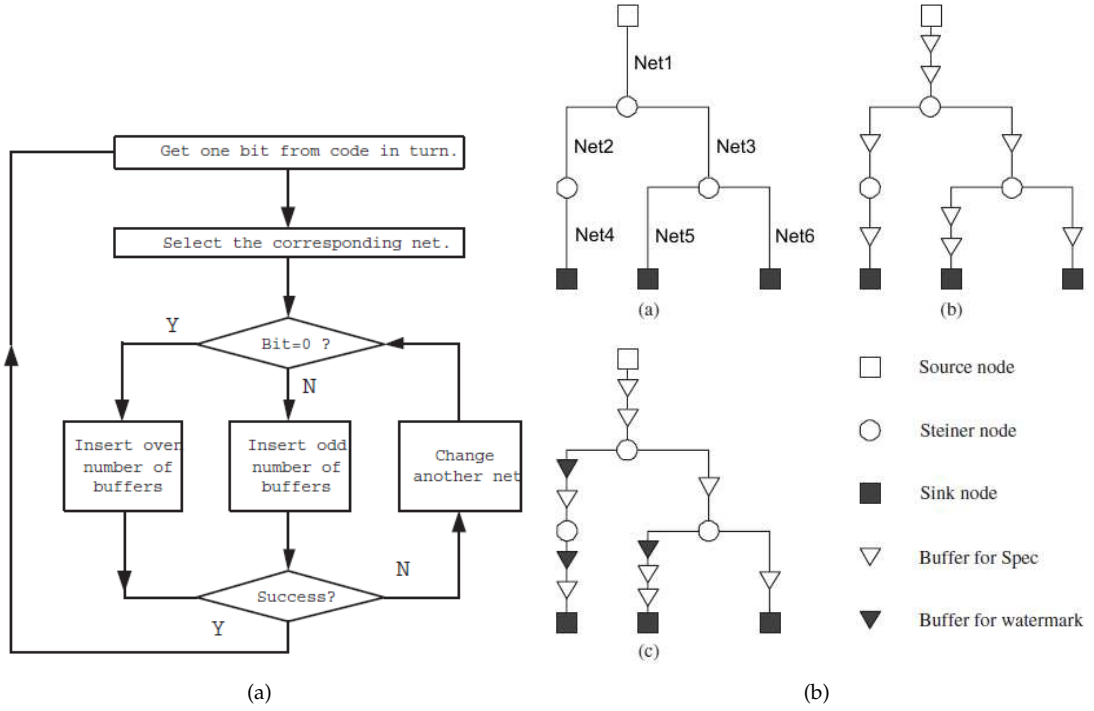
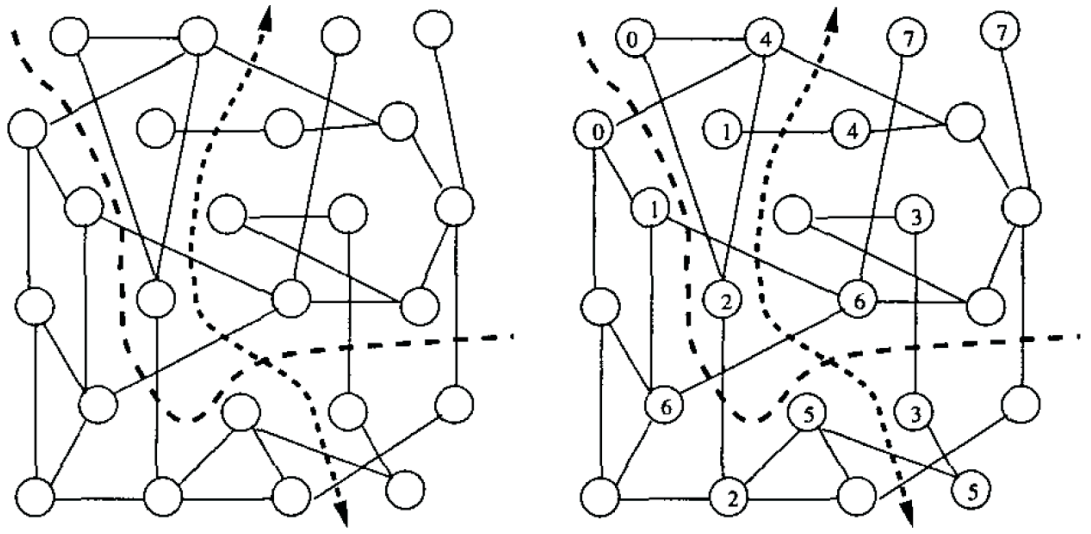


Figure 2.3: (a) Buffer insertion technique design flow [18]. (b) An example of buffer insertion: a) original design, b) added buffers, c) watermarked design.

2.4.1 Hard IP

In hard IP, a digital watermark is represented as physical modifications to the IC layout. One such technique alters the placement of technology library cells through the parity modification [19] or scattering [20,21]. Other techniques modify the number of vias in digital devices or bends in analog devices [22,23], based on the watermark signature. For example, if a watermark bit is '1', a net is altered with an even number of vias and when a watermark bit is '0', the odd number of vias is expected. Since digital devices require additional vias to be implemented, results in [22] demonstrate that the area overhead of the watermarked IC is approximately 12 – 13%. In analog devices, a negligible area overhead is achieved, since the original wires are retained and only small bends must be incorporated [23]. A post-layout parity modification technique [18], embeds additional redundant buffers in specific nets. The design flow of such technique is shown in Figure 2.3(a). If a watermark bit is '1', a buffer parity in a net is odd. Otherwise it is even. In Figure 2.3(b), nets from 1 to 6 correspond to watermark bits '001011'. According to the technique, nets 2, 4 and 5 do not meet the criteria of the algorithm and extra buffers must be embedded.

Other hard IP watermarking techniques utilize the intrinsic features of physical IC layout, obtained from EDA tools. In [24], it is shown that the possibility of any point in a layout overlapping with a polysilicon is usually less than 15%. If n number of watermarking points are chosen, the possibility of all these points overlapping with



(a) (b)

Figure 2.4: Digital watermarking of a partitioning solution [25]. (a) Original solution. (b) Solution with an embedded public and private watermark signatures.

a polysilicon is 0.15^n [24]. Due to such low probability, watermarking is viable. The information about a polysilicon layer is extracted from a GDSII file, which contains the data about the design layout. The experimental results in [24] demonstrate that a similarity of 91% between the expected and obtained watermark signatures is achieved.

2.4.2 Firm IP

In a design flow, electronic design automation (EDA) tools use specific problem solving algorithms, known as heuristics, to find a solution to a hardware optimization and synthesis problems. Solutions which satisfy design constraints are known as the *solution space* [25]. In a firm IP, additional constraints are applied to such optimization problems to embed a watermark. Therefore, the *solution space* must be large enough to accommodate such constraints. In [25, 26], a technique is proposed which embeds a digital watermark in various optimization steps, such as partitioning and graph coloring. Partitioning [27] plays an important role in a VLSI design and can significantly reduce the complexity of a design process, through minimization of interconnections and delay, based on constraints such as number of nodes (balance constraint), area and number of partitions [25]. A watermark is divided into public and private parts. The public part is embedded in such way to guarantee its non-restricted detectability, while the private part is shared only with an authorized person. The public part offers the advantage of the first level of an authorship when it is matched with the expected signature. In a partitioning solution, Figure 2.4(a), a watermark represented by an 8-bit ASCII character is embedded by choosing 8 pairs of vertices. To embed a watermark

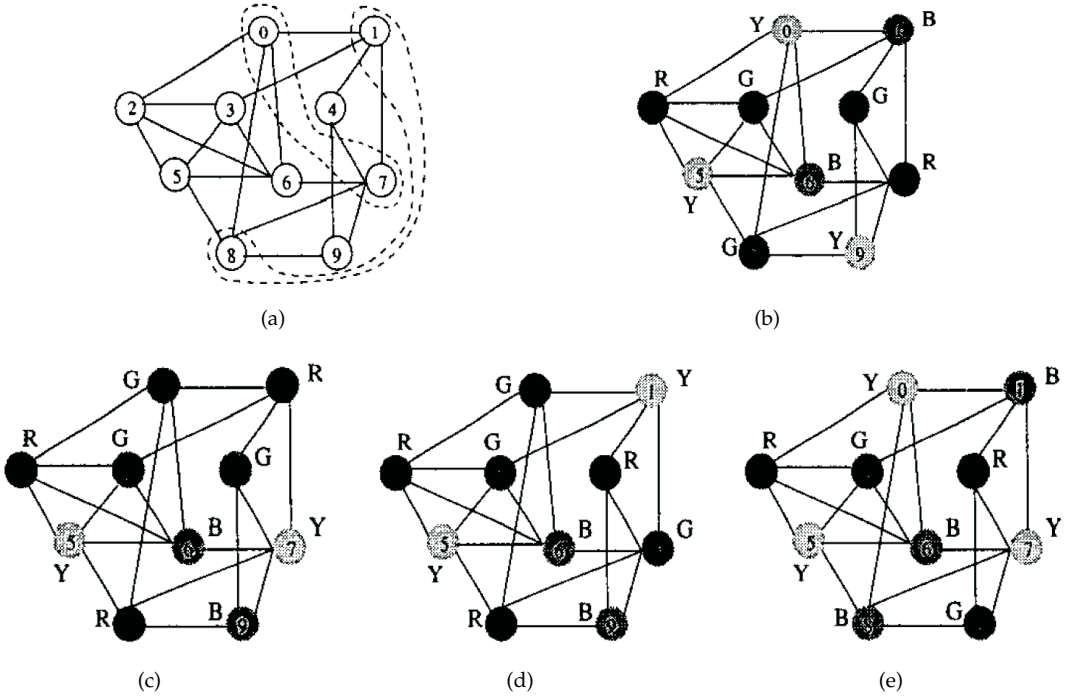
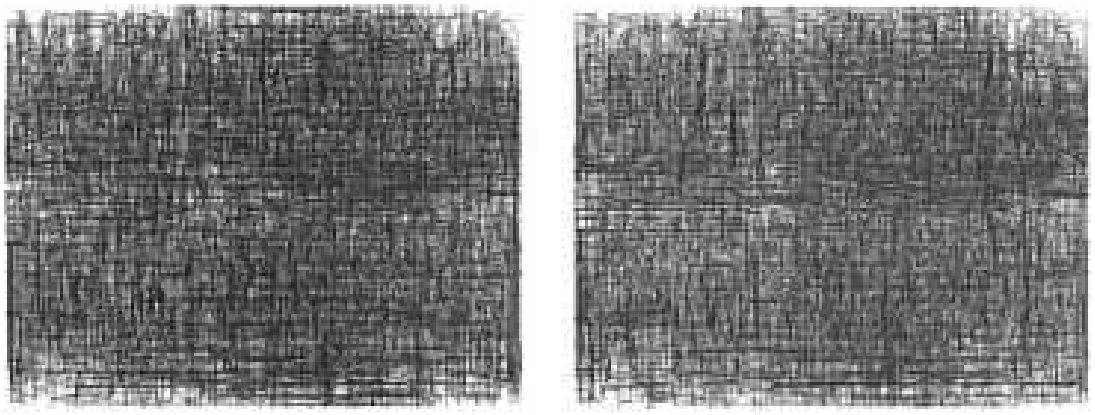


Figure 2.5: Digital watermarking of a graph coloring solution [25]. (a) Original graph and host for a public watermark. (b) Watermark signature '00'. (c) Watermark signature '01'. (d) Watermark signature '10'. (e) Watermark signature '11'.

bit '1', a pair of vertices is forced to be in a separate partition. To embed a watermark bit '0' a pair of vertices is forced to be in the same partition. In Figure 2.4(b), two watermark characters are embedded. The partitions divided by the dashed line with arrow heads represent a public watermark 'p', while the other two partitions represent the private watermark 'O'. The application of the proposed technique to the graph coloring optimization algorithm is similar. To embed a watermark bit '1', a pair of non-adjacent vertices (not directly connected nodes) is given the same colour. Otherwise, different colours are used. As can be seen in Figure 2.5(a), two pairs of nodes are chosen. These are nodes 0, 7 and 1, 8. When a watermark signature represented by a 2-bit sequence '00' is implemented, none of the pairs has the same colour, Figure 2.5(b). If a watermark signature is '11', the first pair (0 and 7) is coloured in yellow and the second pair (1 and 8) is coloured in blue, Figure 2.5(e). Other techniques aimed at the hardware optimization exist and embed a digital watermark during the partitioning [28], graph coloring [26], template matching or operation scheduling [29] steps.

The constraint-based techniques aimed at the design synthesis embed a digital watermark through nodes rewiring [30], alternate scan cells connection styles [31], transform specific local network topologies into their alternative mapped forms [32,33] or restrict the timing of specific nets [19]. For example, given a timing of a net of 50ns, it is further into two sub-nets with the timing of the 1^{st} constrained to 20ns and the timing



(a) (b)

Figure 2.6: Watermarked (a) and non-watermarked (b) design with path timing constraint technique [19].

of the 2^{nd} constrained to $30ns$ [19]. Figure 2.6, demonstrates both watermarked and non-watermarked designs. As can be seen, it is practically impossible to notice any structural change [19]. Therefore it is hard for a third party to tamper with a design. However, constraint-based techniques require a partial or an in-depth knowledge of a watermarked design to detect an embedded watermark signature.

The first implementation of a digital watermark in a state transition graph (STG) of a finite state machine (FSM) was demonstrated in [34] and allowed the detection through I/O ports of a device. Two properties of a watermarked STG were identified. First, *"each state* r_i, $1 \leq i \leq m$ *can only be reached from state* r_{i-1}*"*. Second, *"each state* r_i, $1 \leq i \leq m$ *can only be reached from state* r_{i-1} *by applying input* a_i*"* [34]. States r_i represent an additional set of states, that can be reached only through an application of a specific bit sequence. In Figure 2.7, original and watermarked state transition graphs are shown. As can be seen in Figure 2.7(a), if a bit sequence $a_i =' 010'$ is applied, the states change from q_o to q_3. The STG is modified as shown in Figure 2.7(b). The additional watermark states r_i emulate the original functionality, and if a bit sequence $a_i =' 010'$ is applied, the state changes from $r_0 = q_0$ to r_3. Therefore, the application of a watermark sequence causes a specific and deterministic behaviour of a system. The sequence is generated by encrypting a unique signature string with a known key, using a public cryptosystem tool, such as MD5 [34]. To detect an embedded watermark, outputs ports of a device are observed, while a set of test vectors is applied to the input ports. The proposed technique was implemented in various sized circuits and it was concluded that the area and performance overheads are negligible for a reasonable size systems, while generating robust watermarks [34]. Although the technique embedded a digital watermark in the netlist level, it created the fundamentals for many future soft IP techniques.

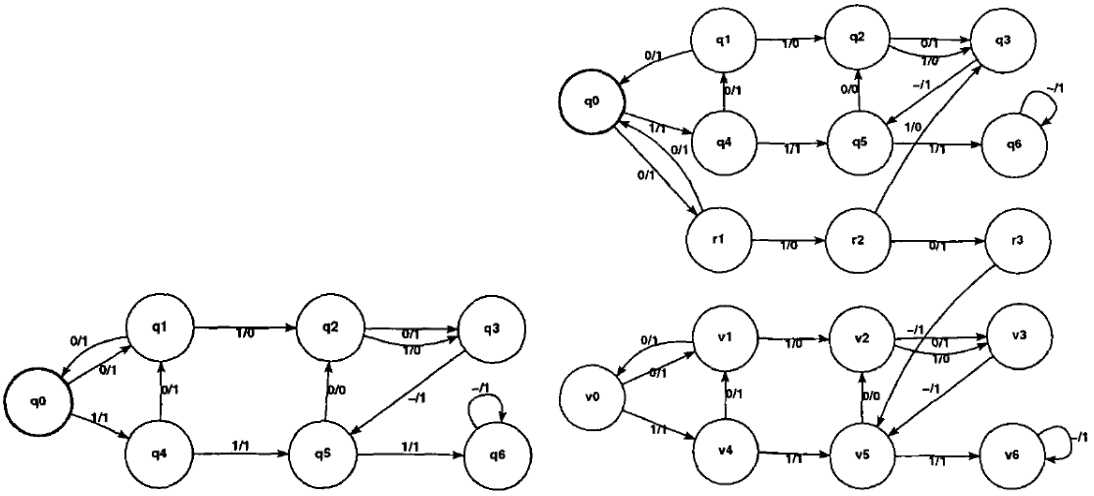


Figure 2.7: Original (a) and watermarked (b) state transition graphs in a FSM watermarking [34].

2.4.3 Soft IP

Finite state machines have been used on many occasions in soft IP watermarking with the addition of extra states as one of the approaches [35]. Enhancements to the implementation algorithm have lead to the partial reuse of existing states [36, 37] and significantly reduced the hardware implementation costs. In [36, 37], three algorithms were proposed and it was demonstrated that only 1.7% area overhead is required in large circuits. Further advances in the implementation algorithms allowed a complete integration of a watermark with existing states, thus reducing the area overhead to approximately 0% [38]. However, such techniques are only applicable in flat FSM designs and are unsuitable for the commercial IP designs. An approximately zero-overhead technique was proposed in [39], and reused most of an already existing states. However, the technique lacked the possibility of direct detection once a chip has been packaged [40]. The solution demonstrated in [40] implemented a hybrid form, by watermarking higher (RTL description) and lower (netlist) abstraction levels of a design [40]. While lower levels offer a more robust watermark implementation, the detection is time consuming. To compensate, higher levels ease the verification but are less robust. Combining RTL and netlist level description increases the robustness of a watermark and reduces the required detection resources. In RTL description the FSM watermarking was used, and in netlist descriptions scan cells reordering techniques were used [41–44]. The detection algorithm asserts input test vectors and captures a watermark signature at the output ports of a device. The technique was demonstrated to generate low area overheads and produce highly robust digital watermarks [40]. In recent years, the new approach to FSM watermarking has been proposed through the

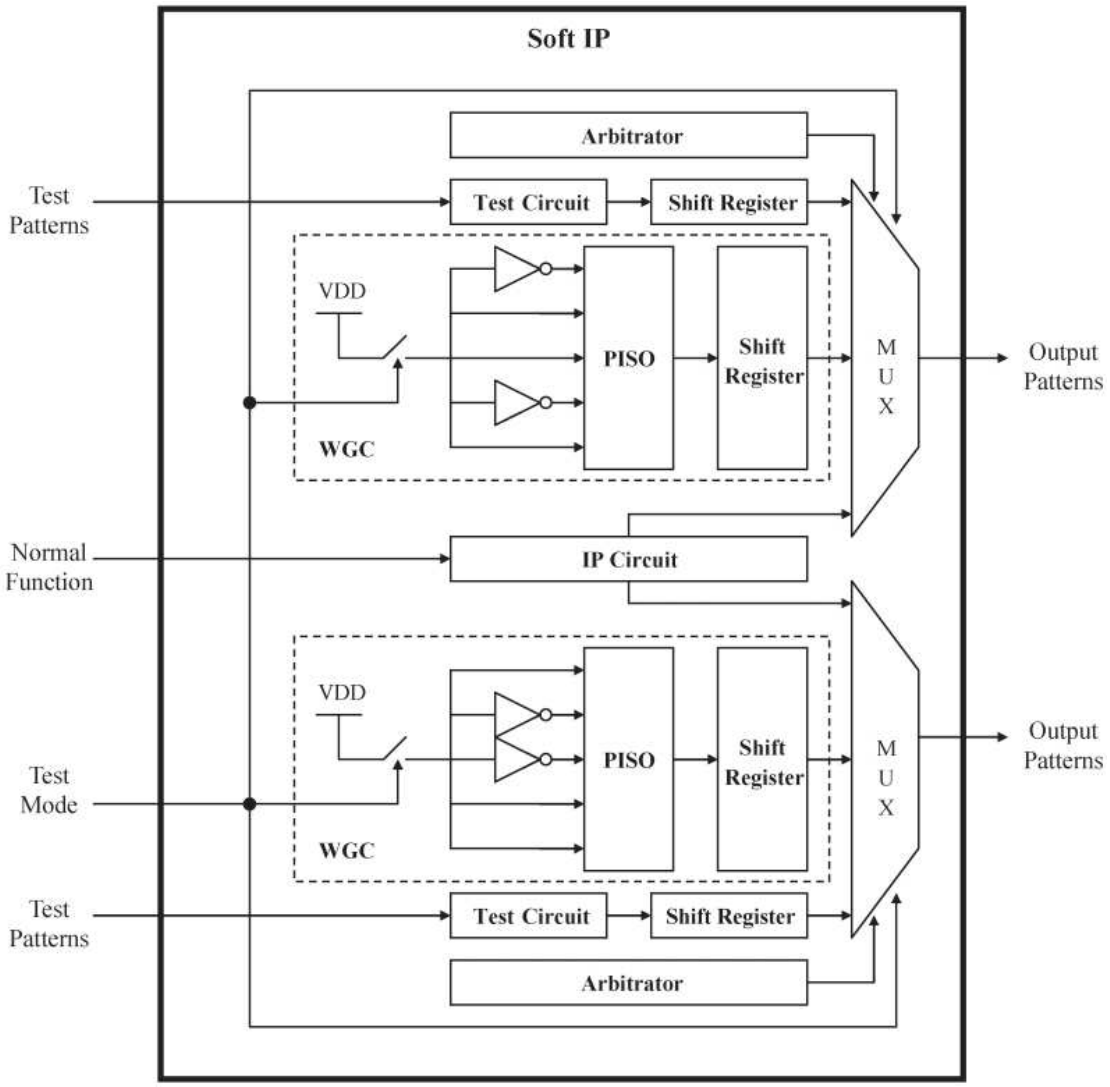


Figure 2.8: Schematic diagram of watermark generation and test circuits embedded in a soft IP [46].

use of genetic algorithms, to reduce the number of states. In [45], the experimental results have demonstrated FSMs with fewer states than before watermarking.

In [46–48], another technique was proposed by embedding watermark generation and test circuits in a single chip. Both circuits were implemented as shown in Figure 2.8. When the test signal is '1', a watermark sequence is generated and propagated to the output ports of a device. Numerous integration schemes have been demonstrated with the watermark signature bits propagated at various points in an output pattern. For example, a signature can be embedded in an output message as a preamble, prior to the actual message, or can be embedded in a message in a random fashion. To protect a watermark circuit and provide the means of RTL obfuscation, standard Verilog HDL encoding features were used. However, as in Section 2.1.2, encryption enforces the use of particular tools, which is not acceptable to many designers and integrators.

```
block_definition : process (CLK)
begin
  if CLK'event and CLK = '1' then
   case table_in is
     WHEN "000000" => data_out <= "0000000000";
     WHEN "000001" => data_out <= "1110000110";
     WHEN "000010" => data_out <= "1100001101";
     …
     WHEN "111010" => data_out <= "0001111001";
     WHEN others   => data_out <= "0000000000";
   end case;
  end if;
end process block_definition;
```

(a)

```
WHEN "111011" => data_out <= 10-bit_sig_block_1;
WHEN "111100" => data_out <= 10-bit_sig_block_2;
WHEN "111101" => data_out <= 10-bit_sig_block_3;
WHEN "111110" => data_out <= 10-bit_sig_block_4;
WHEN "111111" => data_out <= 10-bit_sig_block_5;
WHEN others   => data_out <= "0000000000";
```

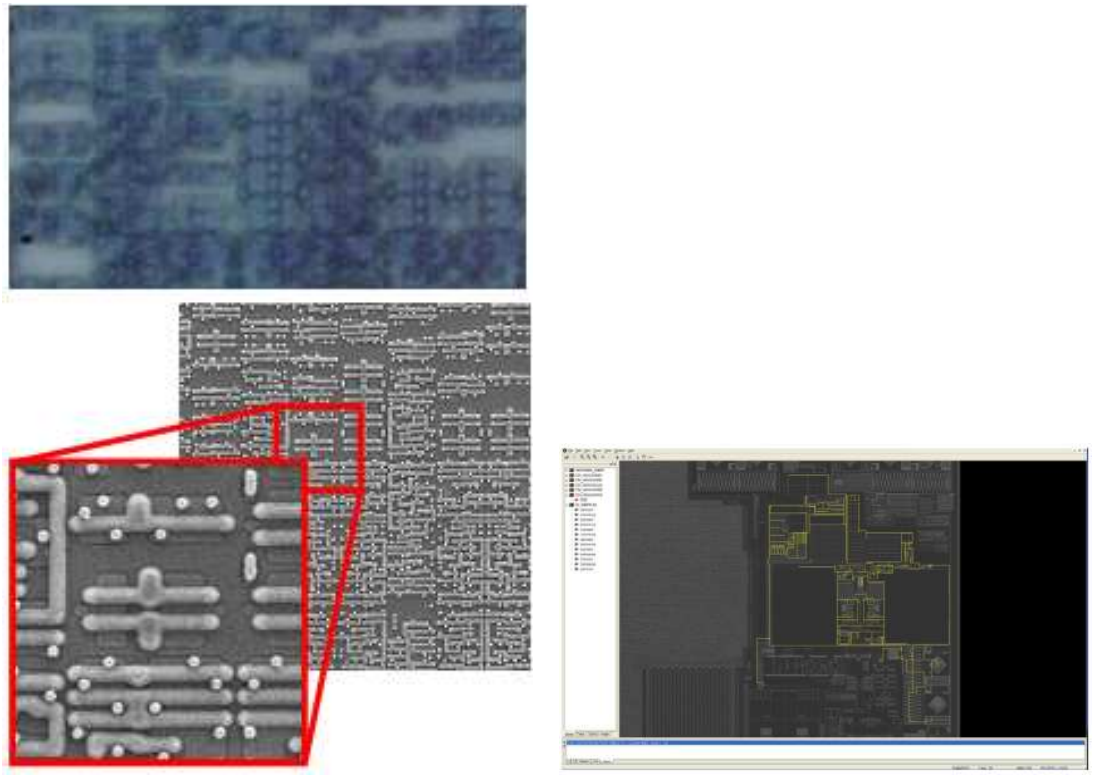
(b)

Figure 2.9: Original VHDL code (a) and additional "case" statements (b) in a memory structure watermarking [49].

In [50–53], a digital watermark is embedded in the empty positions of look-up tables (LUT) on FPGA. The technique implements an additional redundant circuit or uses memory structures and combinational logic, to expose the contents of LUTs to the output ports, when a specific input sequence is applied. Figure 2.9(a) represents the original VHDL code of a memory structure. As can be seen, not all possible combinations are utilized which creates a space for a watermark implementation. This is shown by additional "case" statements in Figure 2.9(b). However, connections between the bus and output ports may not exist after the design has been integrated in a SoC and I/O controller blocks which manage the flow of data between an IP core and device ports may prevent a direct detection [54]. To overcome such limitations two techniques were proposed in [54]. First, a signature co-processor is implemented to scan the data being fetched from the memory. Upon detection of a unique input sequence a watermark signature is copied to the specific memory location. Second, unused instruction op-codes are identified and additional watermark instructions are developed. When such an instruction is executed a watermark signature is stored in memory. Further improvements to the proposed technique were demonstrated in [55], where the watermark insertion was improved with 3 signature distribution algorithms. The results showed a reduction in the area overhead. Nevertheless, to extract an embedded watermark a processor core must be held in a reset state.

2.4.4 Summary

The IP protection through the detection of an embedded digital watermark provides a vast applicability at different levels of a design. In a hard IP level [18–24] (Section 2.4.1), a watermark is embedded through a physical modification of an IC layout. In a firm IP level [19, 25, 26, 28–30, 32–34] (Section 2.4.2), a watermark is embedded through application of additional constraints. Such techniques generate a tamper-resistant watermark with a very low area overhead. The detection technique however requires an access to a watermarked design, such as a micro photograph, GDSII file, fully placed and routed or partial netlist. In case such knowledge is not available, access



(a) (b)

Figure 2.10: (a) Image of a 130nm chip with an optical imaging (top) and scanning electron microscope imaging (bottom). (b) Chip annotation after SEM imaging [6].

to device internals is required to perform a reverse-engineering [6]. At first, a chip is de-packaged with special acids to uncover the die. Next, metal layers of a chip are separated and photographed using an optical imaging or scanning electron microscope (SEM). For technology processes down to $0.25\mu m$ optical imaging is sufficient, however, for technology processes below $0.25\mu m$, optical imaging cannot resolve the smallest features and SEM is required [6], Figure 2.10(a). Once all the images have been obtained and combined, circuit extraction begins and all components, including transistors, capacitors, diodes and other components, interconnections between layers, contacts and vias are annotated. In the past, engineers used a "crawl-around-on-the-floor" technique to annotate wires and transistors, followed by drawing the schematic on a paper, before transferring it to a software schematic editor [6]. Nowadays, special tools are used and the annotation process is automated, Figure 2.10(b). The last step involves the extraction of a netlist from the annotated SEM images and formulation of a flat schematic. The physical tests of an IC, such as micro-probing may also be required to detect an embedded watermark signature. The focused ion beam (FIB) system can be used to scan a chip with a high resolution [56]. Additionally, the FIB system can remove a signal wire as shown in Figure 2.11 or drill holes through metal layers to allow connection to an internal wires. The reverse-engineering approach is however

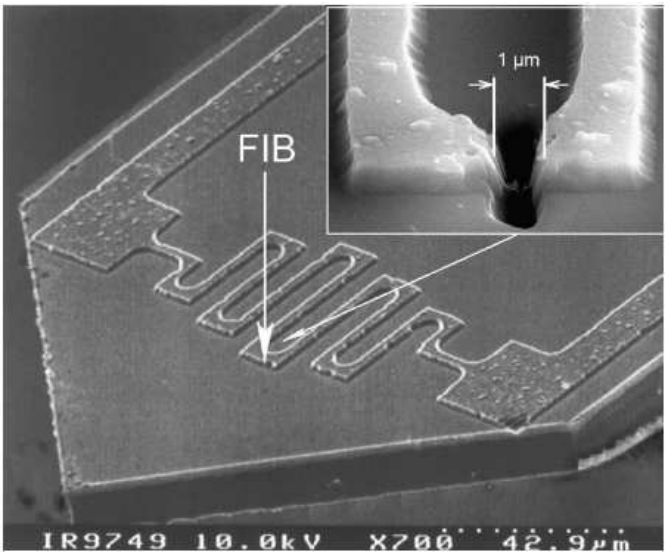


Figure 2.11: Signal wire removal with a FIB system [57].

time consuming and requires an expensive infrastructure.

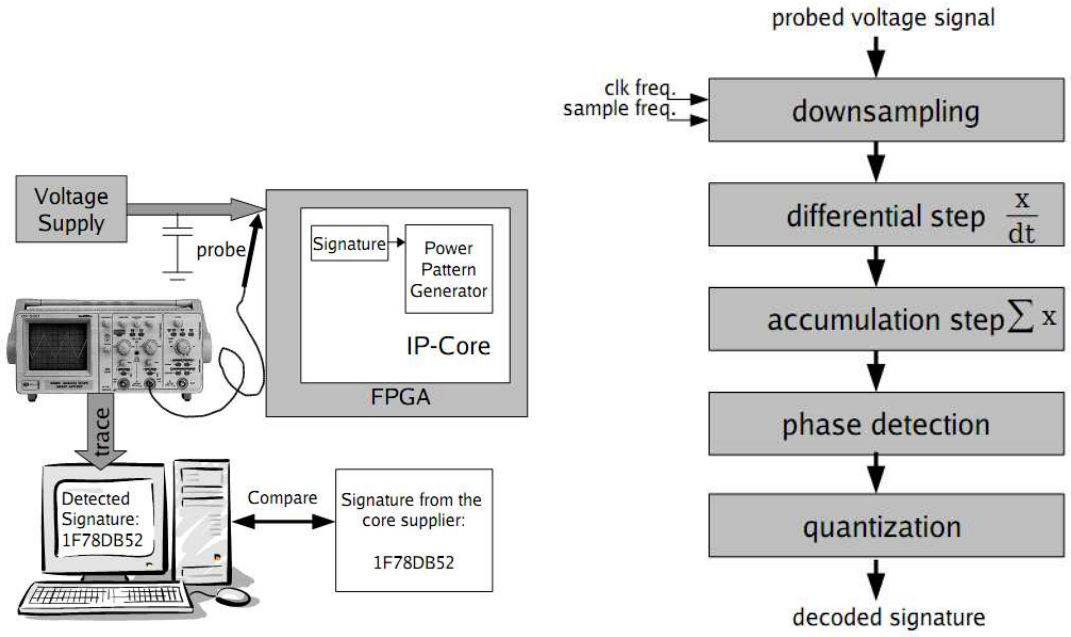
In a soft IP level (Section 2.4.3), digital watermarks are embedded in the RTL description. The FSM watermarking is one of the most commonly used and researched soft IPP techniques [35–44]. Other techniques embed a watermark in LUTs in an FPGA [50–53] or memory structures [54]. Although, most of soft IP techniques allow a watermark detection in a fabricated chip, many require an access to I/O ports or an internal memory. Since, the final system architecture may not be known, an IP owner lacks a sufficient knowledge to able to use such detection techniques and prove an IP infringement.

2.5 Non-Invasive Digital Watermarking

Non-invasive digital watermarking does not require any access to device internals and the system's knowledge is not necessary to successfully perform a watermark detection. Techniques are based on the measurement of device specific characteristics, known as the side-channel information such as power consumption. Therefore, an embedded watermark is commonly known as the power watermark.

2.5.1 Power Watermark

The first power watermarking technique proposed in [58] detected an embedded watermark signature from the power supply pin. This was the first technique to detect an embedded watermark in such way. The watermark circuit was embedded in an FPGA and consisted of two modules: watermark generator and power pattern generator (Figure 2.12(a)). The watermark generator was implemented as a small shift



(a) (b)

Figure 2.12: (a) Power watermark block diagram [58]. (b) Threshold-based watermark detection algorithm [58].

register and generated a watermark sequence to further control a bigger power pattern shift register. When a watermark bit is '1', the power pattern shift register is enabled and rotates all bits by a single position. This causes an instantaneous and substantial power consumption. Otherwise, when a watermark bit is '0' no rotation occurs and no additional power is consumed. The technique was implemented on FPGA. The shift registers were an integral part of the functional logic to increase the robustness of a watermark circuit. Therefore, if a watermark is removed the functionality of a core is impaired. The architecture of the original design was modified in the netlist level, by re-using existing LUTs. Two modes were distinguished: *normal* and *watermark*. In the *normal* mode all LUTs are configured as originally intended. When a system is held in a reset state, the *watermark* mode is asserted and the modified LUTs form a watermark circuit. The advantage of such an approach is the reduced noise from the rest of the system during the watermark detection. Since most of the system is inactive, the major contributor of the power consumption is the watermark circuit.

The watermark implementation technique proposed in [58] results in a robust and a highly tamper-proof solution. However, its applicability is solely limited to FPGAs, due to their configurable architecture. Such an approach is infeasible in case of ASICs since each LUT would synthesize into a number of logic gates [8]. Therefore, a stand-alone circuit must be integrated.

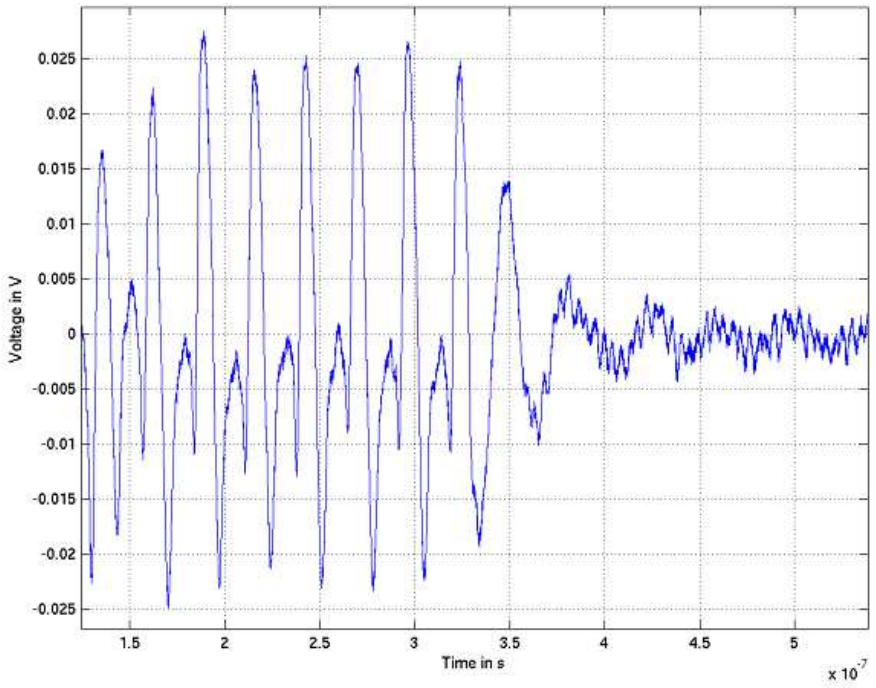


Figure 2.13: Resonance effect caused by a watermark signature generation [58].

2.5.2 Threshold-Based Watermark Detection

The first non-invasive power watermark detection techniques applied a specific post-processing algorithms and thresholds [58]. One such algorithm is shown in Figure 2.12(b). After the quantization of the measured device dynamic power consumption, the post-processed signal is compared with the anticipated threshold. If a signal is above the threshold, the detected watermark signature bit is stored as '1'. If it is below the threshold, the bit is stored as '0'. However, such detection algorithm is sensitive to generated watermark signatures [59]. When a signature contains a long sequence of '1', followed by a short sequence of '0', a resonance effect occurs. Therefore, the immediate bit '0' is stored as '1' and leads to an erroneous detection. In Figure 2.13, such an effect is shown. The watermark signature consists of eight consecutive '1' followed by a series of '0'. As can be seen, the 9^{th} bit generates a significant voltage spike and it is detected as '1'. To prevent such occurrences, a delay of a few clock cycles between consecutive bits of a watermark signature is inserted [59]. Additionally, the generated carrier frequency of the watermark signal may be shifted away from the frequency of a system clock, where most of the interferences occur. The on-off keying (OOK) modulation, proposed in [59], generates a watermark power pattern based on the following algorithm. If a watermark bit is '1', five '1' are generated followed by five '0'. If a watermark bit is '0', the operation is inverted and five '0' are generated followed by five '1'. The above techniques implement a static level of the applied threshold. This greatly reduces the sensitivity of the watermark detection algorithm. In [60], the correlation of the measured power signal with the approximated impulse response was proposed. The detection

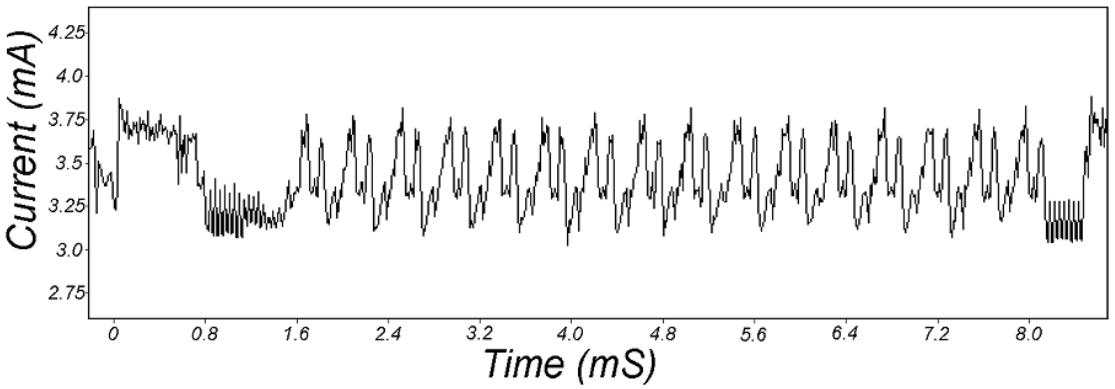


Figure 2.14: Power signal of a DES device during cryptographic operation [61].

algorithm dynamically adjusts the threshold, based on previous samples. However, as shown in previous techniques the experimental results on FPGAs failed to detect an embedded watermark, when its signal-to-noise ratio (SNR) was below $+4dB$.

2.5.3 Statistical-Based Watermark Detection

The threshold-based techniques discussed in Section 2.5.2 demonstrated that a deeply embedded watermark power signal can not be detected, and a strong watermark power signal must be generated leading to significant hardware implementation cost overheads. To overcome such limitations, detection techniques based on statistical analysis have been introduced. Previously, such techniques have been used in the cryptographic systems to extract a secret key, but their development and vast applicability have lead to the implementation in the context of an IP protection. To fully understand the fundamentals of such techniques, it is necessary to consider their development and early use.

2.5.3.1 Differential Power Analysis

The first non-invasive power analysis techniques aimed at detecting a secret key in cryptographic devices used techniques such as Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [61]. The dynamic power consumed by a device is measured, with the sampling frequency higher than the frequency of the system clock. Multiple power samples are acquired during a single clock cycle. The measured power signal is visually interpreted in the SPA approach and enables recognition of operations executed during the cryptographic procedure, such as conditional branches, multiplication and exponentiation. Due to power consumption variations between consecutive clock cycles, Figure 2.14, it is possible to determine when a cryptographic procedure is executed and distinguish consecutive steps (rounds) of such operation. However, due to environmental and system noise, it may not always be possible to directly determine

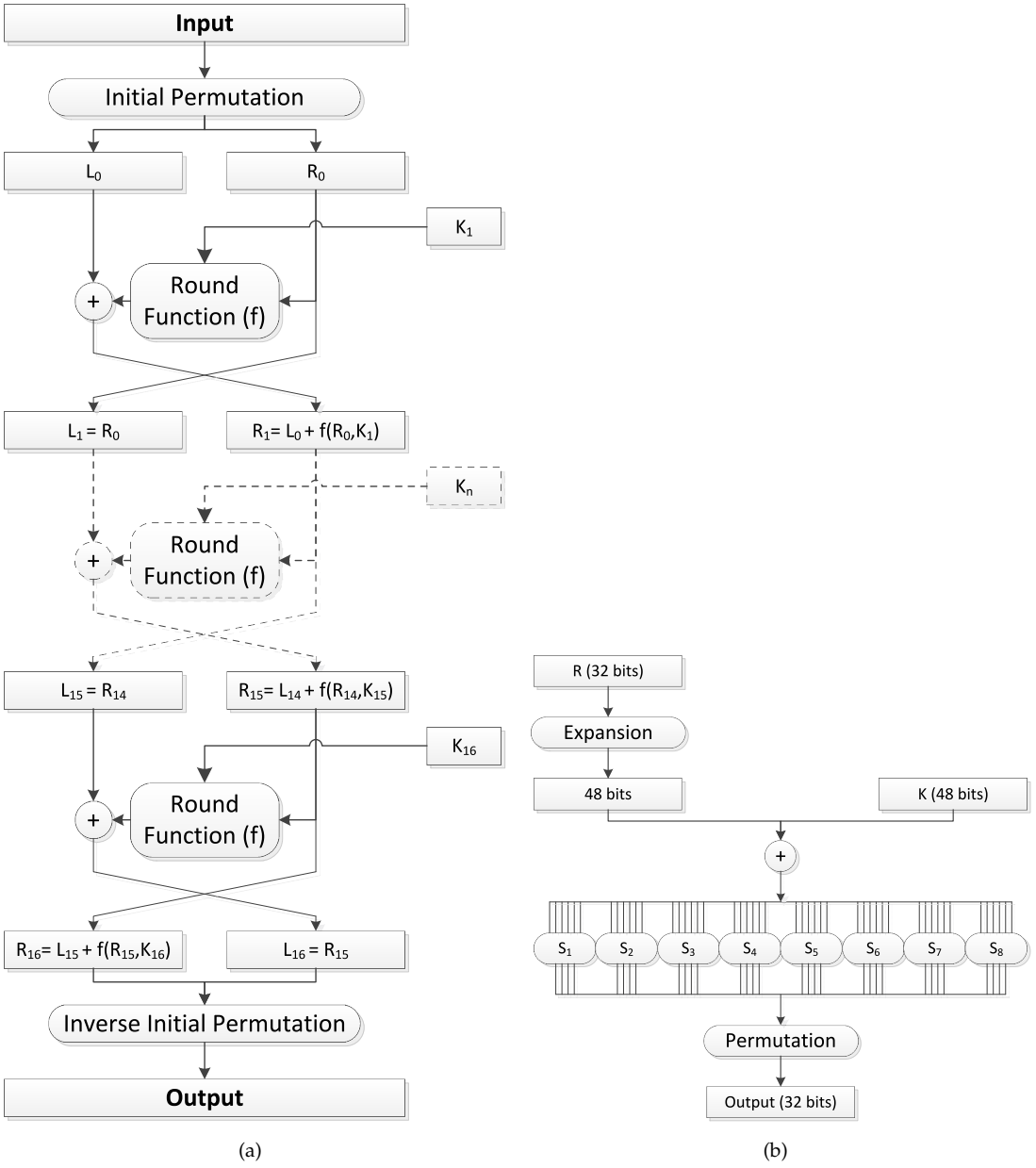


Figure 2.15: a) DES encryption algorithm. b) DES round function.

executed processes with a simple visual inspection. In such cases, the DPA technique can be used with a detection algorithm described as follows [61]:

> *"The DPA selection function, $D(C, b, K_S)$, is defined as computing the value of bit $0 \leq b < 32$ of DES intermediate, L, at the beginning of the 16th round for ciphertext, C, where the six key bits entering the S box, corresponding to bit b are represented by $0 \leq K_S < 2^6$. ... To implement the DPA attack, an attacker first observes m encryption operations and captures power traces, $T_{1..m}[1..k]$, containing k samples each. In addition, the attacker records the ciphertexts, $C_{1..m}$. ... The attacker computes a k-sample differential trace, $\Delta_D[1..k]$, by finding the difference*

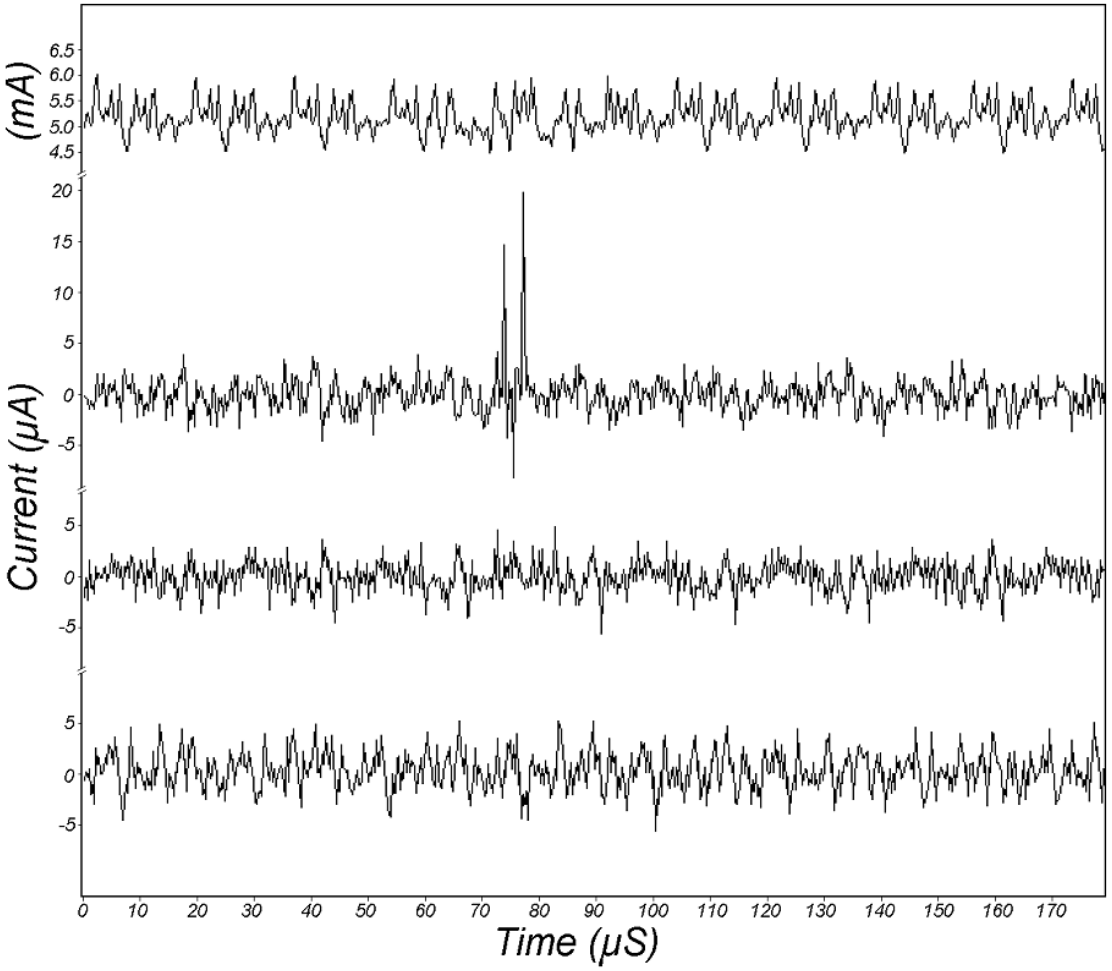


Figure 2.16: DPA applied to a smart card [61].

> *between the average of the traces for which $D(C, b, K_S)$ is one and the average of the traces for which $D(C, b, K_S)$ is zero. Thus $\Delta_D[j]$ is the average over $C_{1..m}$ of the effect due to the value represented by the selection function D on the power consumption measurements at point j." [61]*

The Data Encryption Standard (DES) encryption algorithm is shown in Figure 2.15(a), [62]. It contains 16 rounds of encryption steps, during which R and L registers are obtained, from multiple logical operations. In each round the R and L registers are subjected to repeated transformations, such as expansion and substitution (S-box) functions, Figure 2.15(b). A person who wishes to retrieve a secret key from the DES cryptographic device must first generate and apply N random messages (plaintexts), while capturing power signals (trace). The DPA algorithm retrieves a secret key from the intermediate L register. The selection function, D, is applied and divides the measured power trace into two separate sets of data. The first set contains traces for which $D = 1$, and the second set contains traces for which $D = 0$. An average trace is found for each set, and a differential trace, Δ_D, is computed by subtracting one set from the other. If a key, K_S, is incorrect, D is uncorrelated to the actual result and Δ_D approaches

0. Otherwise, if K_S is correct, D is correlated to the value of the bit manipulated in the 16^{th} round. The graph of Δ_D contains spikes in the regions where D is correlated to the values being processed [61]. In Figure 2.16, the DPA technique is applied on a smart card. The top signal represents the averaged power trace. The next three differential traces represent the correct (top) and incorrect (two bottom) key guesses, respectively. Although countermeasures exist, such as utilizing a voltage regulator as an isolation circuit, to cause the uncorrelated current consumption to the processed data [63], external components are usually required. Moreover, the DPA technique is able to detect very weak signals. Although, an addition of extra noise to a system decreases the amplitude of correlation spikes, it does not remove them completely. Furthermore, techniques to increase the SNR of the measured power signal exist. In [64,65], the types of noise are classified as external, intrinsic, quantization and algorithmic. External noise is generated by a source external to a device, but is coupled with its power signal. It can be reduced with a careful circuit design, proper equipment and filtering. Intrinsic noise is caused by the random movement of charge carriers within conductors while the quantization noise exists in the analog to digital converters (ADC), when an analog signal is converted to a digital signal. Intrinsic and quantization noise are quite small and therefore negligible. Algorithmic noise is caused by the random processing of data bytes in a device and can be reduced if a technique can average the unbiased random data.

2.5.3.2 Leakage-Based Differential Power Analysis

Dynamic power is the major contributor to the overall power consumption in micrometer technology processes. However, due to scaling in CMOS technology, the static power has become a significant factor in deep submicron processes [66]. In the 45*nm* technology process, the average contribution of the static power accounts for an approximately 24% of an entire power consumption [66]. It is expected, that with further technology scaling the static power will become the dominant factor and detection techniques will become more viable [66]. The leakage-based DPA (LDPA) technique [67] demonstrated a significant dependency of a static current on the data operation execution in a two-input NAND gate, and provided further evidence of the data dependency in inverters, XOR and NOR gates. The conventional DPA technique applied to a 180*nm* device was compared with the LDPA technique, applied to a 45*nm* device. It was concluded, that the LDPA technique retrieved a secret key much faster [67]. In [68], the static power attacks were perfomed on 3 FPGAs fabricated using 65nm, 45nm and 28nm technology processes. Although attacks were successful, it was shown that static power attacks were less efficient than dynamic power attacks.

2.5.3.3 Correlation Power Analysis

The fundamental approach of the conventional DPA technique (Section 2.5.3.1), is the computation of a distance between the means of two sets of data (difference-of-means). In [69], a high correlation between the consumed power and the Hamming Weight [1] (HW) of the processed data was demonstrated. Several such techniques have been proposed [69–71], where the hypothetical model of a cryptographic device was used to detect a secret key. However, in [72] it is argued that such hypothetical model is not necessarily correct. Experimental results on 13 cryptographic devices have shown that a successful detection could not be performed. In [9], a different technique was proposed, based on computation of the Hamming Distance[2] hypothetical model. The technique is known as Correlation Power Analysis (CPA) and assumes that the Hamming Distance is highly correlated to the data being processed. The CPA computes a Pearson correlation coefficient, ρ, as in Equation (2.1) to describe the degree of linear similarity between the two sets of data. Therefore, it can be used for a direct comparison of the hypothetical model, H, with the real data, W.

$$\rho = \frac{cov(W, H)}{\sigma_W \sigma_H} \tag{2.1}$$

The CPA algorithm consists of three stages: prediction, measurement and correlation and can be described as follows [73]:

1. The Hamming Distance for the hypothetical model is formulated. It is repeated for all 2^8 possible values of a byte of a secret key, and N random words from which $N * 2^8$ Prediction Matrix is obtained.
2. The power signal is captured for X clock cycles and a single value per clock cycle is found by taking the highest power consumption sample. It is further stored in the $N * X$ Consumption Matrix.
3. The correlations between the column in the Consumption Matrix and all columns in the Prediction Matrix are computed.
4. The column with the highest correlation value describes the correct key guess associated with the column number.

The value of the correlation coefficient can range from -1 to 1, where the sign indicates the relationship. For example, if a hypothetical model directly follows the Hamming

[1] Hamming Weight (HW) represents the number of '1' in any binary sequence. For example, a sequence '0110' has HW of 2, while '1111' has HW of 4.

[2] Hamming Distance represents the number of bit changes between two binary sequences. For example, if the 1^{st} sequence is '0110', and the 2^{nd} sequence is '1001', all 4 bits must change and Hamming Distance is 4.

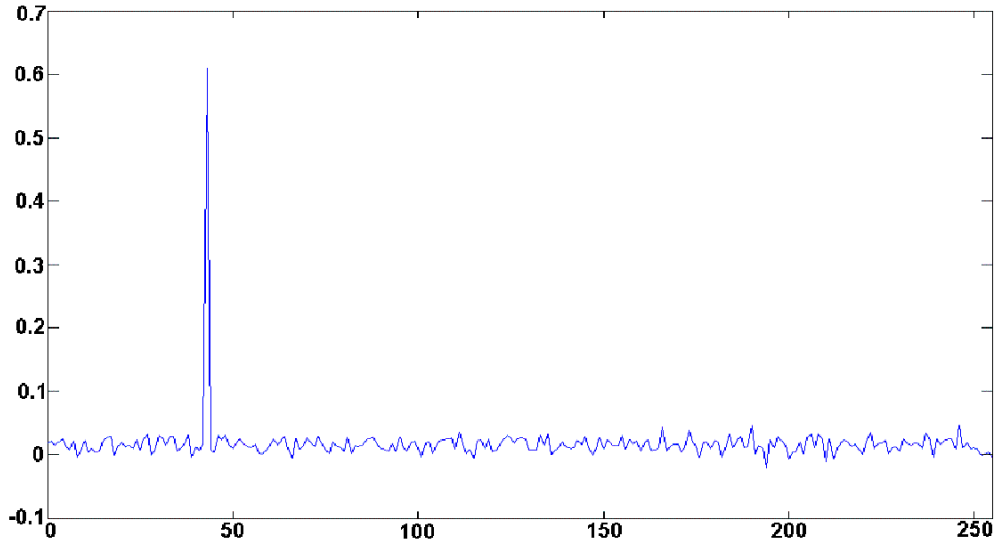


Figure 2.17: CPA applied to a cryptographic core on FPGA [73].

Distance of a measured power signal, the coefficient is 1. If a hypothetical model is exactly an inversion of the Hamming Distance, the coefficient is -1. The value of 0 indicates that no linear relationship between the model and the measured signal exists. Such value should be expected from uncorrelated noise. The CPA applied to a cryptographic IP core implemented on an FPGA is shown in Figure 2.17. It can be seen, that a significant peak stands out from other correlation coefficients. Therefore, the correct key has been found. The CPA technique is argued to be more efficient and robust than the DPA technique but requires a more in-depth knowledge about the architecture of a system [9]. The DPA may therefore be advantageous in case of hardened chips, where reverse engineering would be impractical [9].

Similarly to the DPA technique, CPA is a cryptographic technique with the aim of retrieving a secret key during the encryption process on a cryptographic device. The knowledge about the architecture of a system is therefore required to perform a successful detection. In case of an IP protection the watermark circuit is embedded by an IP supplier and the hypothetical model of a watermark circuit is simply required to successfully perform a watermark detection. Although, an IP can be embedded as a part of a system, such in-depth knowledge is not required.

2.5.3.4 Correlation Power Analysis Watermark Detection

The first application of the CPA technique in IP digital watermarking was demonstrated in [8]. Two types of power watermark architectures were proposed. The first, a spread-spectrum watermark (Figure 2.18(a)) and the second, an input modulated watermark (Figure 2.18(b). In the spread-spectrum, a watermark architecture is similar as in Section 2.5.1 [58]. The watermark generation circuit (WGC) consists of a Pseudo Random

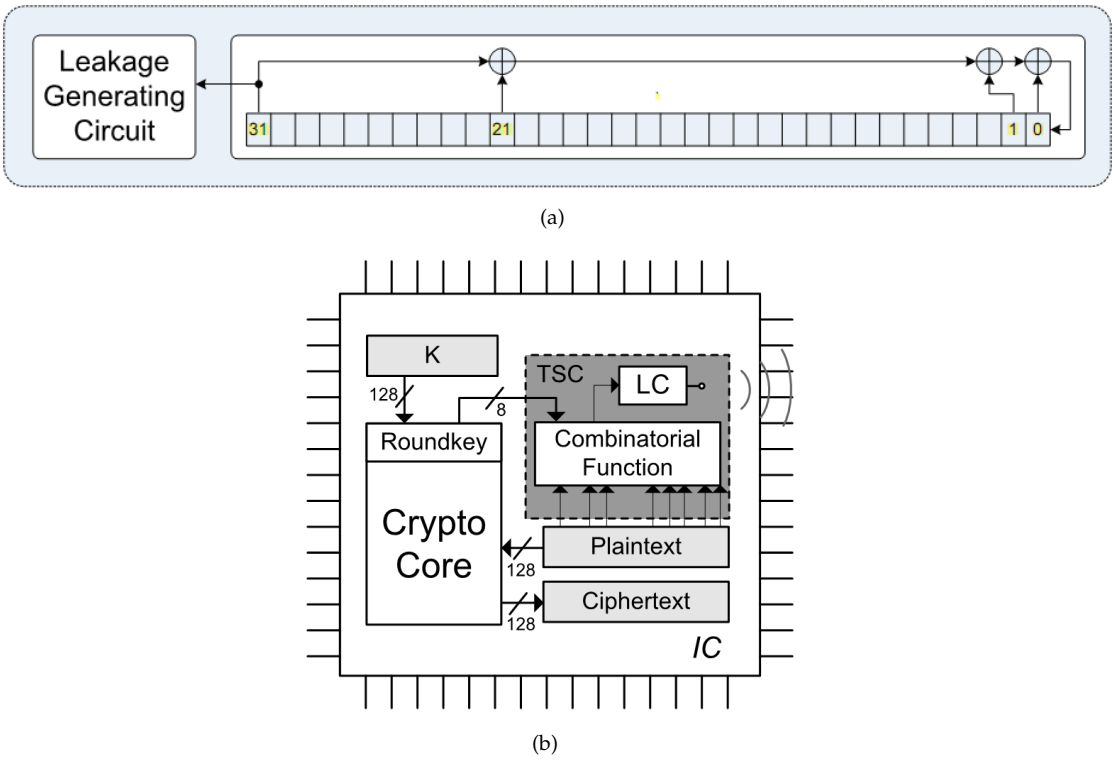


Figure 2.18: (a) Spread-spectrum watermark architecture [8]. (b) Input modulated watermark implementation [74].

Number Generator (PRNG), such as Linear Feedback Shift Register (LFSR) and controls the watermark power pattern generator (WPPG). The architecture of the WPPG consists a significant number of circular shift registers. The signal-to-noise ratio (SNR) of the generated watermark power signal increases with the number of registers utilized in the WPPG circuit. If a watermark bit is '1', the WPPG is rotated by a single register. Otherwise, if a watermark bit is '0' no rotation occurs. The input modulated watermark creates a specific power pattern, based on a set of secret and known bits [74]. The detection algorithm, based on the CPA, can be described as follows [8]:

1. The dynamic power consumption is measured and the average value per clock cycle is found. The data is stored in the Power Vector, Y.
2. The hypothetical model of a watermark sequence is generated and it is stored in the Model Vector, X.
3. The Pearson's correlation coefficient, ρ, is computed as in Equation (2.2), to obtain a single correlation point.

$$\rho = \frac{N\sum_{i=1}^{N} X_i Y_i - \sum_{i=1}^{N} X_i \sum_{i=1}^{N} Y_i}{\sqrt{N\sum_{i=1}^{N} X_i^2 - (\sum_{i=1}^{N} X_i)^2}\sqrt{N\sum_{i=1}^{N} Y_i^2 - (\sum_{i=1}^{N} Y_i)^2}} \tag{2.2}$$

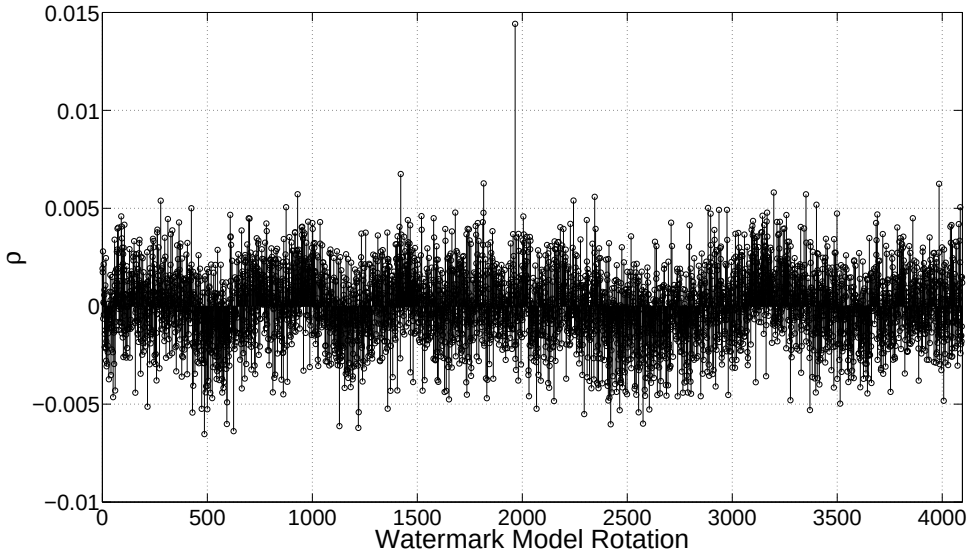


Figure 2.19: The spread spectrum of the CPA watermark detection.

4. Since X and Y may be out of phase, X is rotated by a single clock cycle.
5. Steps 3 to 4 are repeated $N - 1$ times, where N is the length of a watermark sequence.
6. Each correlation point is plotted on a graph, known as the spread spectrum.
7. If a significant peak can be distinguished and no other peaks exist, a watermark is deemed found.

In [8], a watermark circuit was implemented on an FPGA. To generate a watermark sequence, the WGC circuit used a $32 - bit$ LFSR and the WPPG circuit used 16 LUTs, each configured as a 16-bit circular shift register. A successful watermark detection was demonstrated with only 100 clock cycles, when an IP core was held in a reset state. However, $250,000$ clock cycles were required, when the core was active. In Figure 2.19, the spread spectrum is shown. The single correlation peak confirms the successful detection of an embedded watermark and proves that the technique is a viable non-invasive detection technique for watermarks embedded in soft IP cores.

2.5.3.5 Null Hypothesis Significance Test Watermark Detection

In [75], the combination of the CPA detection technique with a statistical analysis tool, known as the Null Hypothesis Significance Test [76] was proposed. The simulation results implemented the WGC using the LFSR. To emulate the environmental and system noise sources, a normally distributed random numbers were added to the generated watermark sequence. To increase the watermark circuit SNR, the amplitude of a noise

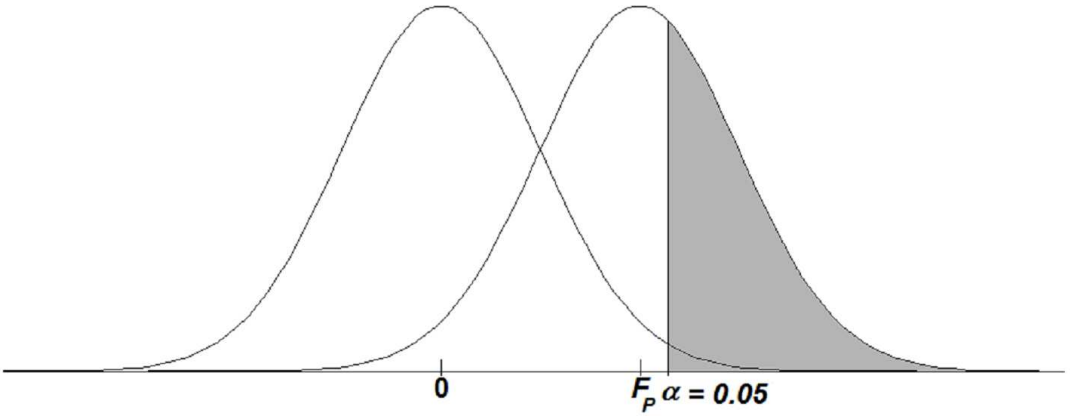


Figure 2.20: Correlation distributions of the *null* (left) and *alternative* (right) hypotheses in the Null Hypothesis Significance Test [73].

signal was increased. The correlation was computed as in Equation (2.2). Since the distribution of ρ is skewed [75], the Fisher Transform of ρ was found as in Equation (2.3), to approximate the distribution as normal.

$$Fisher(\rho) = 0.5 \ln \frac{1+\rho}{1-\rho} \tag{2.3}$$

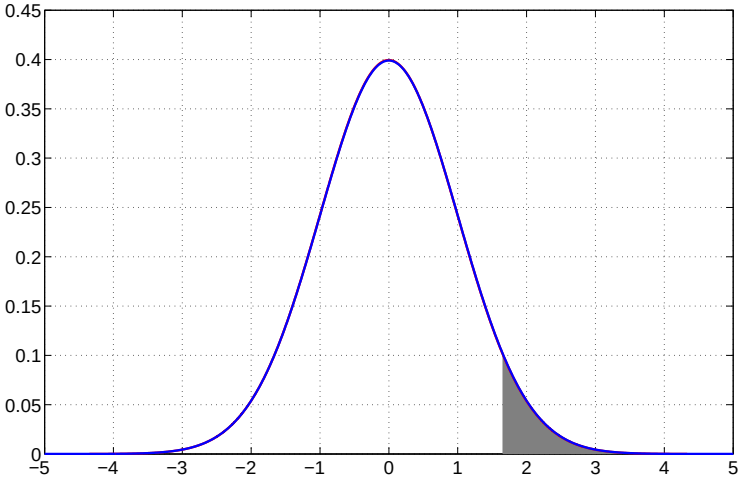
The standard deviation of $Fisher(\rho)$, $\bar{x}$, is directly related to the number of samples, N.

$$\sigma = \frac{1}{\sqrt{N-3}} \tag{2.4}$$

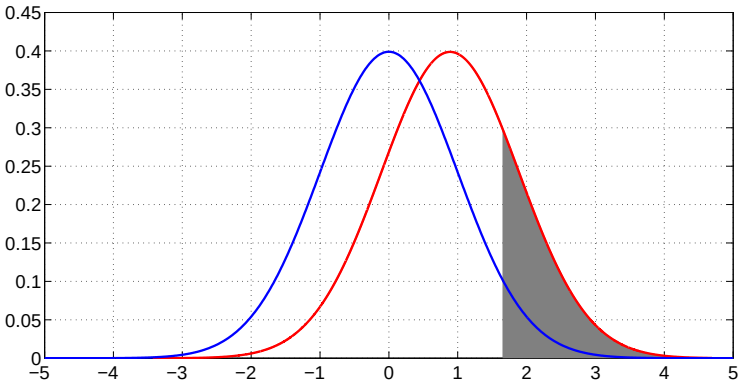
The distribution of $\bar{x}$ obtained from the non-watermarked system is known as the *null hypothesis*, Figure 2.20. The threshold level found from the *null hypothesis* is used to determine the percentage of successfully detected watermarks in consecutive tests. The distribution of $\bar{x}$ obtained from the watermarked system is known as the *alternative hypothesis*. To determine if the *null hypothesis* can be rejected and therefore the *alternative hypothesis* can be accepted, the z-value was computed as in Equation (2.5):

$$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}} = \bar{x}\sqrt{N-3} \tag{2.5}$$

The $\bar{x}$ is the Fisher transformed ρ. The μ_0 is the mean value of the *null hypothesis* and it is 0, since a watermark does not exist. The value of n is 1, as only a single $\bar{x}$ is computed. The probability of observing by chance a result, that is at least as extreme as the one being tested, known as the p-value, is found from the z-table which contains probabilities of the standard normal distribution [75]. Finally, the p-value is compared against the significance level for a given test. For example, the $z - value$ equals 1.65 and the significance level for the test is 5%. The probability found from the z-table is 0.95053 and the $p - value = (1 - 0.95053) = 0.04947(4.95\%)$. This is less than the significance level, hence the *null hypothesis* can be rejected.



(a)



(b)

Figure 2.21: Null Hypothesis Significance Test error types: (a) Type I, (b) Type II.

In [73], the change in the *sample correlation* distribution due to increase in the number of samples is demonstrated as the reduction of standard deviation, while the mean is kept constant. Since, it is assumed that the standard deviation of a distribution obtained from the watermarked system is the same as the standard deviation of a distribution obtained from a non-watermarked system, they can be standardized. It is a division of $\bar{x}$ by its standard deviation. Therefore, both distributions are a standard normal distributions. The mean value of a watermarked distribution is however non-zero ($\mu \neq 0$), as shown in Figure 2.21(b). Comparison of a watermarked distribution with the standard normal distribution, for which $\mu = 0$, is required to determine whether the *null hypothesis* can be rejected. If a line is drawn at the point of significance, the distribution of the standardized Fisher transform to the left of the line represents the *null hypothesis* not being rejected. Distribution to the right represents rejection of *null hypothesis* and acceptance of *alternative hypothesis*, Figure 2.21(b). Increasing the number

of samples causes the distribution of a watermarked system to shift to the right along the horizontal axis, increasing the shaded area and percentage of the *null hypothesis* rejection.

The technique demonstrated in [75] focuses on Type I errors, α, also known as false alarms. In such case, a watermark is assumed as detected while it does not exist. Type II errors, β, which have not been addressed in [75], occur when a watermark that is present is not detected. Type I error is controlled by the significance level, however, one can only be sure of such error when the *null hypothesis* is true [77]. As shown in Figure 2.21(a), the distributions of $\bar{x}$ and *null hypothesis* are the same. In such case, α equals the significance level. Type II error is controlled by the statistical power of the test. In Figure 2.21(b), statistical power equals the shaded area of the plot, hence $\beta = 1 - \textit{statistical power}$. The error rate is therefore very high. Furthermore according to [77], it is not known which error applies without knowing if *null hypothesis* is true or false. In Figure 2.21(a), error could be as little as 5%, if *null hypothesis* was true, or as much as 95%, if it was false. The NHST technique has already been considered misleading [76]. For example, with 99% of tests rejected, the statement "99% *of tests found a watermark*" is incorrect. The statement "99% *of tests rejected the* null hypothesis, *hence found watermark with a given statistical significance level*" is more accurate. Moreover, when the *null hypothesis* is not rejected, it should not be accepted [78]. Therefore, with 99% tests rejecting the *null hypothesis*, the remaining 1% does not confirm or deny the absence of a watermark.

2.5.4 Hardware Trojan Detection

The classification of hardware trojans and watermark circuits is very often interchangeable. For example, in [74] a watermark is regarded as a hardware trojan, since it generates a specific power pattern. The aim of a trojan circuit is to leak secret information, such as passwords, secret keys or even sabotage the functionality of a device [79]. It is embedded by a third party, also regarded as an attacker, whose sole purpose is to tamper with an IP design. It is a much smaller circuit and can consist of a few logic gates. The digital watermark is a purpose built circuit, embedded in an original design by an IP supplier, with the aim to leak an information about its existence. Since such a goal can also be reached with a trojan circuit, it offers a viable solution to an IP protection. To fully understand the implementation and detection techniques of a hardware trojan, it is important to consider its fundamental principles.

2.5.4.1 Principles of Hardware Trojans

If a power measurement, M, is performed on an integrated circuit, I, that executes a calculation, C, the measured power can be modeled by four components: mean power

consumption, $p(t;C)$, process noise, $n_p(t;I;C)$, measurement noise, $n_m(t;M)$, and an extra static power consumption caused by a trojan circuit, $\tau(t;I;C)$ [80]. The model of a power trace of an original IC, r_G, can be found as in Equation (2.6).

$$r_G(t;I;C;M) = p(t;C) + n_p(t;I;C) + n_m(t;M) \tag{2.6}$$

The model of a power trace of a tampered IC, r_T, with an embedded hardware trojan, $\tau(t;I;C)$, is given by

$$r_T(t;I;C;M) = p(t;C) + n_p(t;I;C) + n_m(t;M) + \tau(t;I;C) \tag{2.7}$$

The $p(t;C)$ is computed from measurements obtained from numerous ICs. Since it is common in both original and trojan ICs, it can be subtracted from the computation. The $n_m(t;M)$ is a random noise that varies with each measurement, M. Moreover, it is the only component which depends on M and can be minimized by averaging multiple power traces. The power trace of both ICs can be further modeled by

$$r_G(t;I;C;M) = n_p(t;I;C) \tag{2.8}$$

$$r_T(t;I;C;M) = n_p(t;I;C) + \tau(t;I;C) \tag{2.9}$$

2.5.4.2 Principles of Trojan Detection

The detection of $\tau(t;I;C)$ can be represented by the following fundamental problem [80]:

> *"Given K genuine ICs, $I_1, I_2, ..., I_K$, and process noise, $n_p(t;I_1;C)$, $n_p(t;I_2;C)$, ..., $n_p(t;I_K;C)$, generated by $I_1, I_2, ..., I_K$, respectively, during the execution of C, and given an IC, I_{K+1}, with the mean $r(t;I_{K+1};C)$ obtained from multiple executions of C (where $p(t;C)$ has been subtracted), how can we determine if the IC, I_{K+1}, contains a trojan circuit?"*

To distinguish an original IC from a trojan IC, the following two hypotheses must be met [80]:

$$Original\ IC : r(t;I_{K+1};C;M) = n_p(t;I_{K+1};C) \tag{2.10}$$

$$Trojan\ IC : r(t;I_{K+1};C;M) = n_p(t;I_{K+1};C) + \tau(t;I_{K+1};C) \tag{2.11}$$

This can be described by the signal characterization problem. The power signals of both original and trojan ICs must be characterized and compared. If a difference between the signals is significant, a trojan IC can be distinguished. Signal processing techniques, such as Karhunen-Loève (KL) expansion, can be used to detect an embedded trojan circuit [80]. The KL technique projects signals into their characteristic subspaces and separates the process noise, $n_p(t;I;C)$, from the measured signal noise.

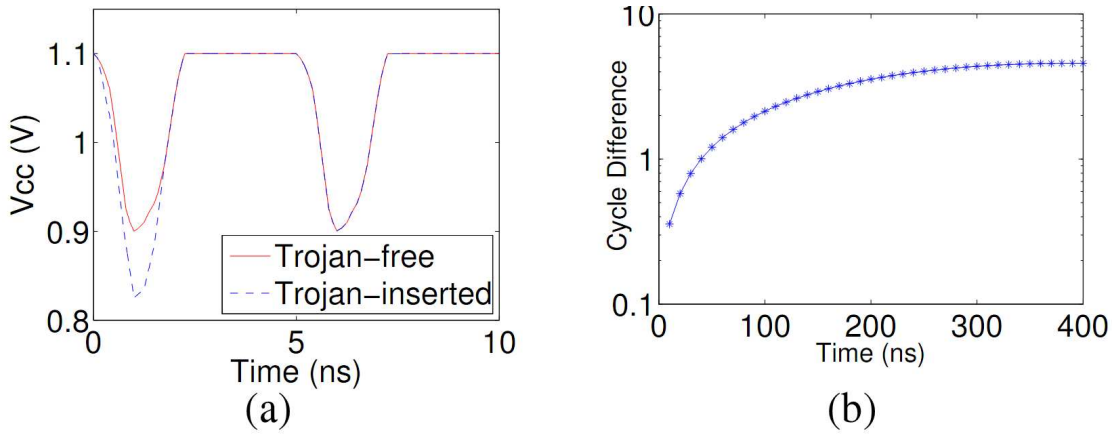


Figure 2.22: (a) Voltage drop caused by trojan circuit insertion. (b) Impact on clock cycles difference by an addition of a trojan circuit [81].

Simulation results demonstrated in [80] detect trojan circuit with the area overhead of approximately 0.01%.

2.5.4.3 Ring Oscillator Based Trojan Detection

In [81], a ring oscillator (RO) based trojan detection technique was proposed. The technique embeds a ring oscillator network (RON) in a design. Each RO acts as a power monitor where its output frequency depends on a propagation delay. Such delay is further influenced by temperature, supply voltage, load capacitance, threshold voltage, channel length, oxide thickness and transistor channel width in an IC [81]. Although the effect of temperature variations can be minimized by subjecting all ICs to the same environment, such as temperature chamber, other parameters are susceptible to power supply noise and process variations. The power supply noise is described by a voltage drop. Therefore, when a trojan circuit is embedded in an IC, it increases an overall power consumption and has the direct impact on a RO delay and output frequency, Figure 2.22. To detect an embedded trojan circuit, the network of N ring oscillators is distributed throughout a design, Figure 2.23(a). In [81], a 5-stage inverter is implemented as a single ring oscillator, Figure 2.23(b). Three detection schemes based on statistical algorithms have been demonstrated [81]: Simple Outlier Analysis, Principal Component Analysis and Advanced Outlier Analysis. The Simple Outlier Analysis measures the distribution of a RO oscillation cycle. If it is within a specified range, an IC is regarded as trojan-free. Otherwise, it is assumed that a trojan circuit is embedded. The Principal Component Analysis transforms the data from all N ring oscillators into an uncorrelated data and reduces the dimension of a data set, using linear combinations [82]. Each principal component describes the amount of variance in a data set. The first component accounts for the most of variance. The variance in successive components decreases. The first three components are represented by a 3-D space and a three-dimensional boundary, known as the convex hull [83], is constructed [81].

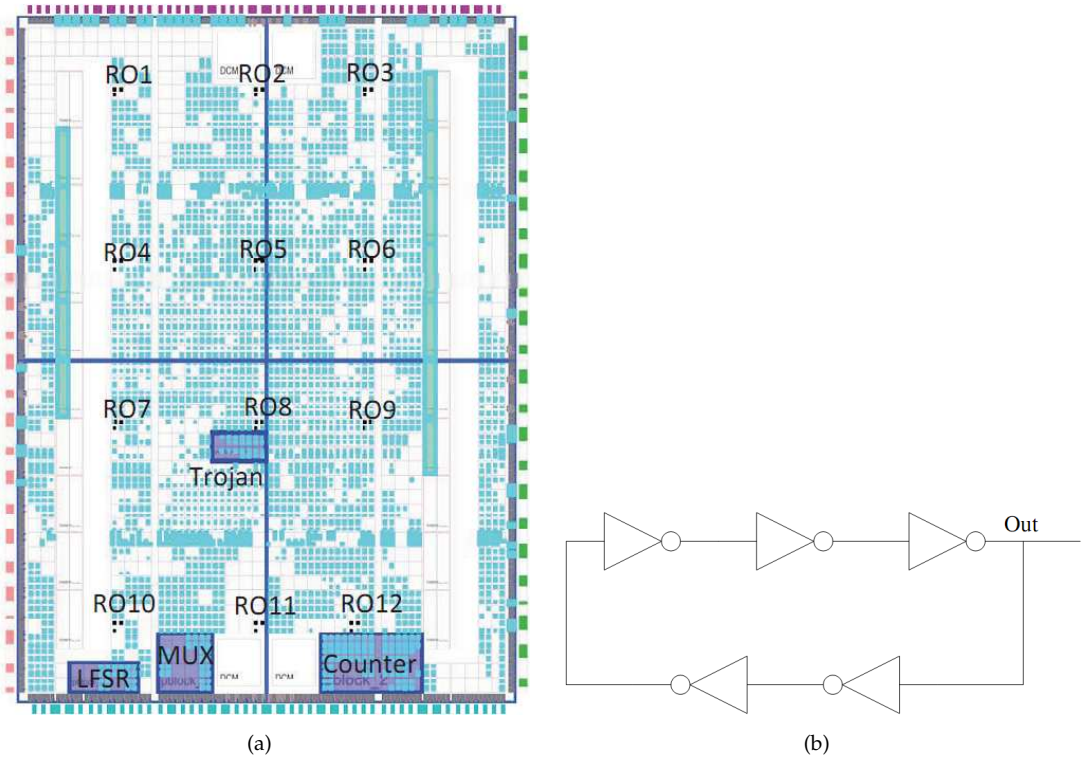


Figure 2.23: (a) Ring oscillator network implemented on an FPGA. (b) 5-stage inverter ring oscillator [81].

The data points outside a convex hull represent trojan ICs. The data can be further analyzed using Advanced Outlier Analysis to consider the relationship between ROs in a network. Experimental results on a cryptographic IP core demonstrated three sizes of trojan circuits [81]. The first accounted for 0.33%, the second for 0.25%, and the third for 0.17% of total area overhead. The combination of all three detection techniques achieved a 100% successful detection rate for the first two trojans. The successful detection rate of 80% was achieved for the third trojan circuit.

2.5.4.4 Multiple Supply Pad Trojan Detection

In [84], trojan circuits are detected through the measurement of static current consumption, from multiple power supply pins. Such pins are shown as PP_{ij} in Figure 2.24(a). To minimize the effect of process variation, a dedicated calibration process is proposed [85, 86]. It uses on-chip calibration circuits connected directly through the scan chain. Such calibration circuits are placed directly underneath the power supply pins and generate a step current. The static current measurement results are represented by a 2-D graph and a statistical analysis techniques, such as regression analysis, are applied. In Figure 2.24(b), the regression analysis of 45 ICs, fabricated in a $65nm$ CMOS technology is shown. The data points are dispersed along the middle line, called regression line. The limits are chosen based on the 3σ (3 * standard deviation). The

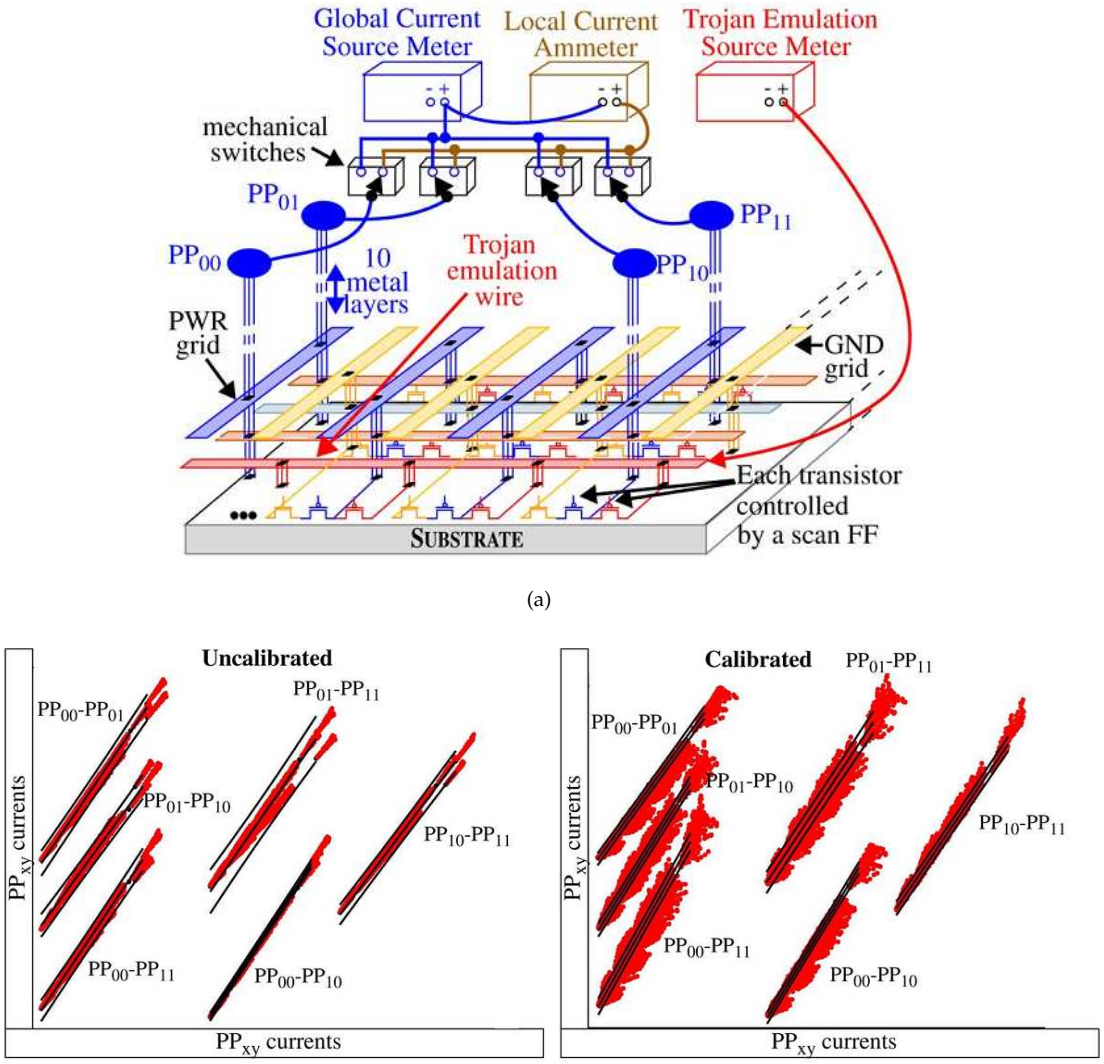


Figure 2.24: (a) Block diagram of the measurement setup for multiple power supply pin trojan circuit detection. (b) Regression analysis with uncalibrated (left) and calibrated (right) signals for trojan circuit detection [84].

calibration process reduces the variance of data and decreases the distance between the limit and regression lines. As can be seen in Figure 2.24(b), the detection of trojan circuits in un-calibrated data is nearly impossible, while most trojan ICs can be detected after the calibration process has been performed.

2.5.4.5 Multiple-Parameter Trojan Detection

The previously discussed side-channel techniques for detection of watermark or trojan circuits are based on analysis of a single parameter, such as delay, dynamic or static current consumption. In [87, 88], the multi-parameter trojan circuit detection technique is proposed, which combines the dynamic current consumption, I_{DDT}, and the maximum

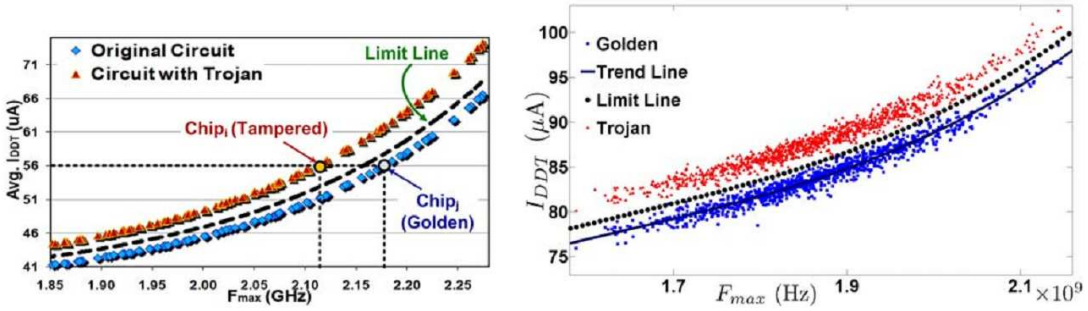


Figure 2.25: I_{DDT} vs F_{max} plot for inter (left) and intra (right) measurement variations [87].

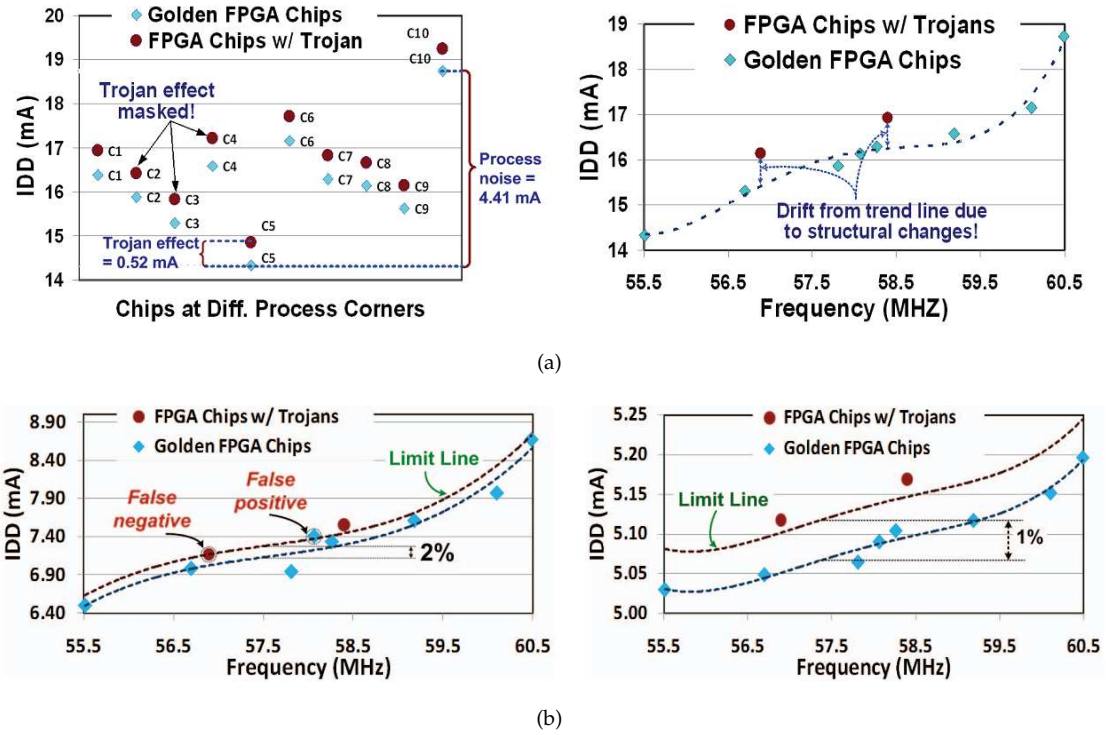


Figure 2.26: Experimental results of (a) 16-bit sequential trojan with a single side-channel parameter (left) and multi-channel parameter (right) detection schemes, and (b) 4-bit sequential trojan with a 2% (left) and 1% (right) trend lines [88].

operating frequency, F_{max}, to determine if measurements differ from the golden trend. As can be seen in Figure 2.25, trojan ICs can be clearly differentiated from original ICs. However, in large circuits an increased detection sensitivity is required. This is achieved through partial activation of a system with dedicated test vectors. Furthermore, power gating or clock gating techniques can be utilized to further enhance the detection technique. Experimental results on an Integer Execution Unit (IEU) with a 5-stage pipelined multiplier, implemented on an FPGA, were demonstrated. Two trojan circuit were embedded, represented by sequential counters occupying 256 and 4 registers, respectively.

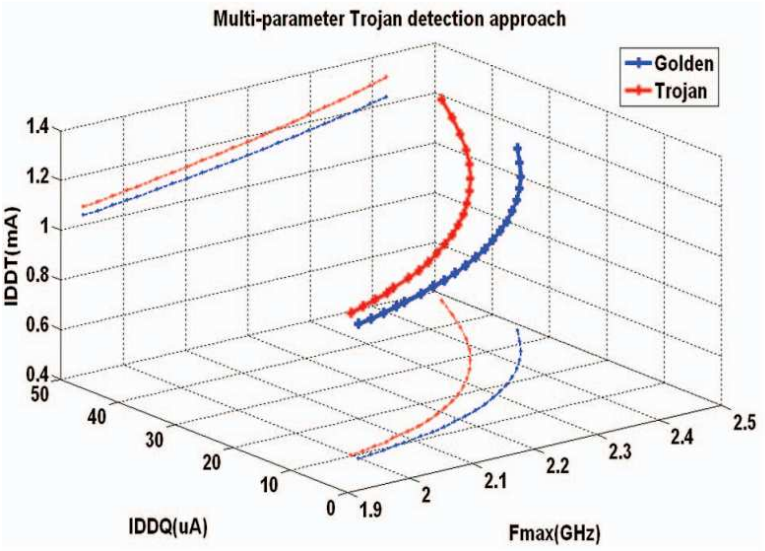


Figure 2.27: Trojan circuit detection with a multi side-channel parameters [88].

This accounts for 1.76% and 0.03% of total area overhead. Two sets of test vectors were applied to execute low and high activity logic operations. The detection of the larger trojan circuit with a single side-channel parameter is shown in Figure 2.26(a) (left). As can be seen, a single side-channel technique is not able to distinguish trojan ICs from original ICs. However, when both I_{DDT} and F_{max} are combined, a visible difference from a golden trend is noticed. Detection of the smaller trojan circuit is shown in Figure 2.26(b). If a limit of 2% is chosen, only a few trojan circuits are detected (left). If a limit is further reduced to 1%, all trojan circuits are detected. However, a small set of original ICs is also falsely claimed to be tampered with a trojan circuit (right).

In [88], an interesting three-dimensional side-channel technique was demonstrated which analyzes parameters, such as dynamic current, I_{DDT}, static current, I_{DDQ}, and maximum operating frequency, F_{max}. Figure 2.27 demonstrates the use of such parameters in a 3-D plot. The clear differentiation between the original (golden) and trojan ICs can be seen.

A similar approach proposed in [89] investigates various trojan implementation scenarios with 1-dimensional (1-D) and 2-dimensional (regression analysis) statistical analyses on FPGA, based on the ring oscillator network topology shown in Section 2.5.4.3. The 1-D analysis considers parameters, such as frequency and static current consumption separately. However, as both parameters are highly correlated (94.4% [89]), their combination is analyzed by a regression analysis. Within-die and die-to-die variation effects are minimized through *regional* calibration algorithm described as follows [89]:

1. The FPGA or ASIC is divided into separate regions as shown in Figure 2.28 and the reference IC is chosen.

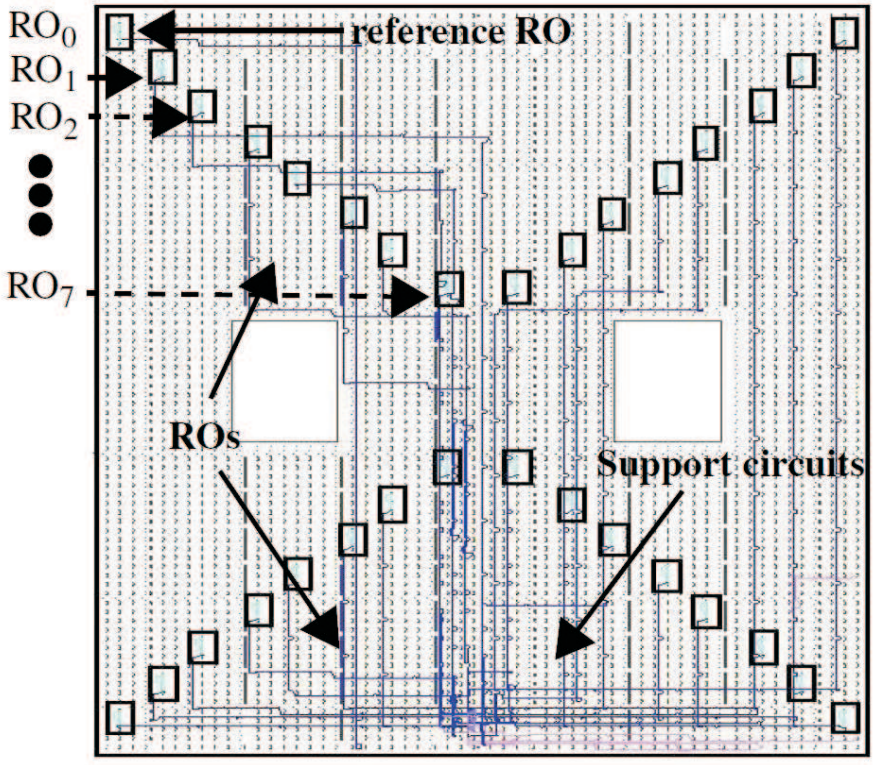


Figure 2.28: Regional RON calibration for trojan circuit detection [89].

2. The current and frequency measurements are obtained for all chips, using the same *regional* ring oscillator. For example, in the top left region, RO_3 is chosen for all ICs.

3. The ratios between the measurements obtained from the reference IC and other ICs of the same *regional* RO are computed.

4. The measurements from other *regional* ring oscillators are multiplied by the ratio found for this specific IC and its region.

5. The steps are repeated for each region in an IC.

The 1-D uncalibrated technique demonstrates 9% and 19% trojan detection with current and frequency, respectively. The calibration algorithm increases the detection to 83% and 78%. The uncalibrated regression technique achieves 80% successful detection rate while the calibration increases the detection to 100%.

2.5.5 Summary

Non-invasive detection techniques do not require any access to device internals and are much cheaper to perform. The side-channel parameters, such as timing, electromagnetic field radiation and power consumption can be used as sources of information.

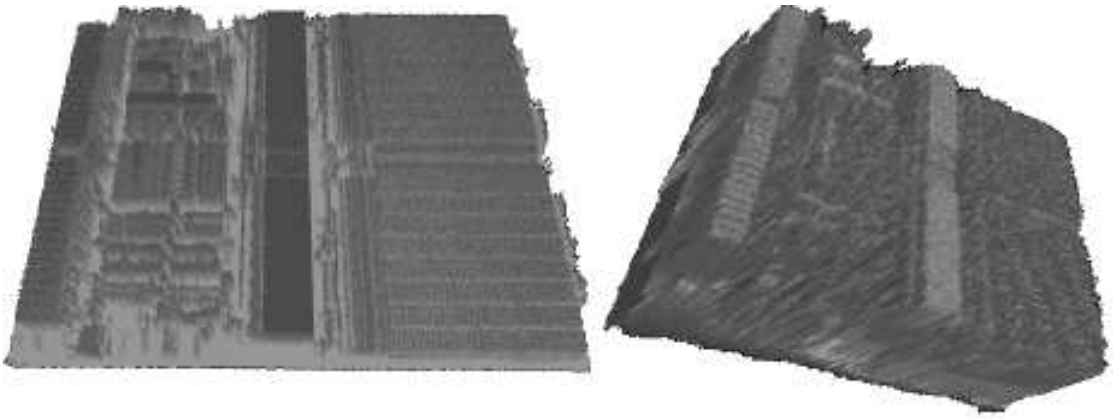


Figure 2.29: 3D signature of the smart card processor obtained with an electromagnetic field attack [92].

However, the analysis of small variations in time [90, 91] to perform cryptographic computations requires an in-depth knowledge of a system. Techniques based on analysis of electromagnetic (EM) field [92–95] with a sensor placed close to a device allow the characterization of each device in a unique way with a spectrum analyzer (Figure 2.29). However, in a EM-shielded ICs the external packaging must be physically removed before performing a detection [95]. Non-invasive power analysis detection techniques (Section 2.5) offer a viable solution to protecting commercial IP designs, since the knowledge of the architecture is not required to successfully perform a watermark detection. The threshold-based techniques [58–60] (Section 2.5.2) require a system to be held in a reset state during a watermark detection. Although, physical reset pads can be widely found on printed circuit boards (PCB), there is no guarantee that such connectivity will always be available. Moreover, such techniques are unable to detect deeply embedded signals and require a significant size power pattern generation circuit. For example, in [58] 92 out of 1332 LUTs on FPGA, a 6.9% of system area, are used with each LUT configured as a 16-bit shift register. The detection of a deeply embedded watermark power pattern is possible with the statistical-based techniques, such as Correlation Power Analysis [8] (Section 2.5.3.4), but the architecture of a power watermark circuit is similar and a significant size WPPG circuit is implemented to generate a strong enough power pattern. Other detection techniques for the detection of embedded trojans have been proposed (Section 2.5.4.1), based on integration of ring oscillator networks [81, 89], power measurements of multiple supply pads [84] or the combination of numerous side-channel parameters [87, 88], and allow a detection of a negligible sized circuits. However, a design knowledge is too fine and destructive IC tests are necessary in many cases. Furthermore, an access to I/O ports is required and usually a golden, non-watermarked IC must be identified.

2.6 Concluding Remarks

In this chapter, various Intellectual Property Protection (IPP) schemes have been discussed. It has been demonstrated, that most IPP techniques enforce an in-depth knowledge or an access to a watermarked system. Therefore, the flexibility of such techniques is greatly reduced. However, non-invasive power analysis techniques, such as Correlation Power Analysis allow IP protection through detection of an embedded digital watermark in measured device power consumption. The great advantage of such techniques is that an in-depth knowledge of an original system is not required and only the architecture of an embedded digital watermark is necessary. Despite the significant hardware implementation costs, the CPA remains the current state-of-the-art technique for the protection of soft IP cores.

Chapter 3

Principles of Non-Invasive Digital Watermarking

In Chapter 2, the methodologies for IP protection (IPP) have been discussed. It has been shown that digital watermarking techniques implemented in the soft IP demonstrate the highest level of flexibility. The introduction of statistical side-channel analysis tools, such as Correlation Power Analysis, in the context of IPP has allowed IP suppliers to distribute soft IP sub-systems, such as commercial embedded processors, and has provided the means of non-invasive detection of embedded digital watermarks after the chip has been fabricated. Nevertheless, watermarking for non-invasive detection implies additional hardware costs which impact the area and power overheads and the robustness of the final solution against third party IP attacks. In this chapter, the principles of digital watermarking are discussed and attacks against embedded watermarks are introduced.

The rest of the chapter is organized as follows. In Section 3.1, the in-depth analysis of non-invasive power watermarks, such as the dynamic power consumption in digital circuits is provided. The architecture of the current state-of-the-art power watermark for soft IP is presented in Section 3.2, and the algorithm of the current state-of-the-art detection technique for deeply embedded watermark power signals, such as Correlation Power Analysis is provided in Section 3.3. Third party IP attacks are analyzed in Section 3.4. The summary and the overview of requirements for the non-invasive IPP in soft IP through watermarking is given in Section 3.5. Finally, Section 3.6 concludes the chapter.

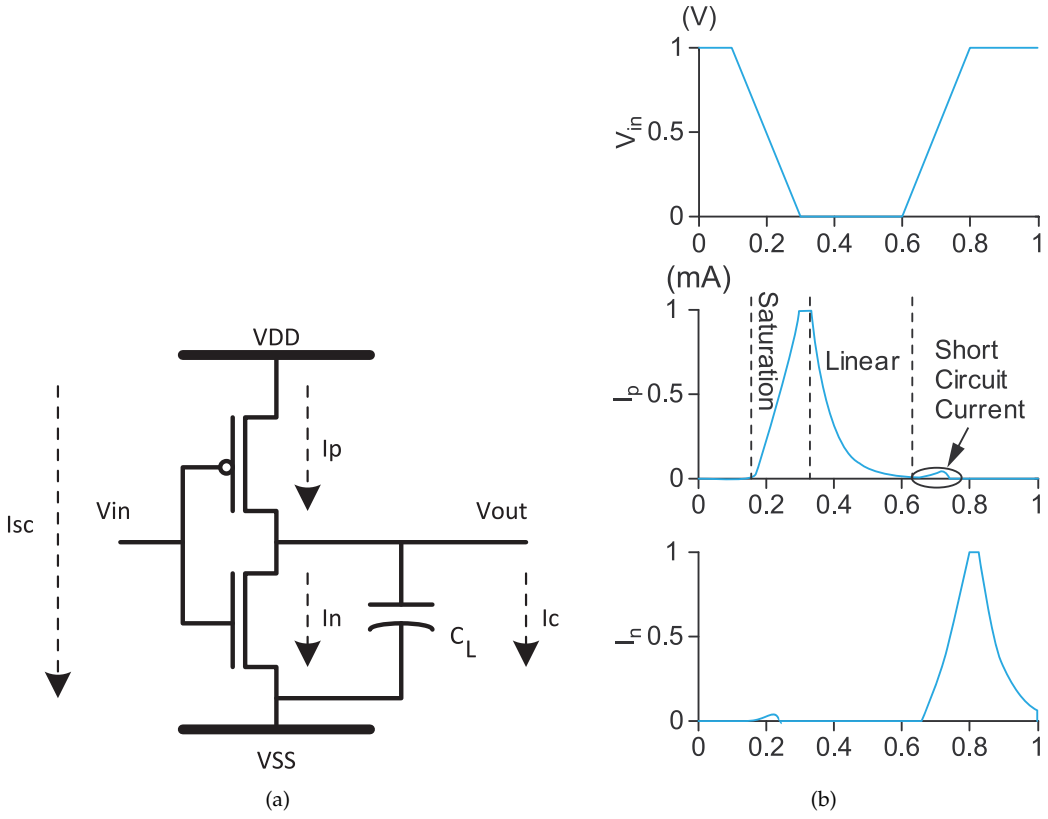


Figure 3.1: a) CMOS inverter. b) CMOS inverter switching voltage (top) and current of PMOS (middle) and NMOS (bottom) transistors [96].

3.1 Dynamic Power Consumption in Digital Circuits

The dynamic power consumption is the cornerstone of the vast majority of side-channel power analysis techniques. To explain the behaviour of power consumption in digital circuits a simple CMOS inverter is shown in Figure 3.1(a). Consider the starting state of the inverter, where the input voltage equals V_{DD}. The load capacitance C_L is fully discharged and the output voltage equals V_{SS}. When the input changes from '1' (V_{DD}) to '0' (V_{SS}) (Figure 3.1(b), top), the PMOS transistor turns ON and charges the load to V_{DD} (Figure 3.1(b), middle). While the input voltage decreases the current across PMOS transistor I_P increases. Finally, the output voltage reaches the point where PMOS transistor is in the linear regime, hence the load is charged and the I_P starts decreasing exponentially. When the input voltage increases, the PMOS starts to turn OFF while NMOS turns ON (Figure 3.1(b), bottom). Since both transistors are instantaneously conducting, a small current flows from V_{DD} to V_{SS} and is known as the short-circuit current, I_{SC}. In general, the inverter draws power from V_{DD} when input state changes from '1' to '0'. One half of the power consumed by the PMOS transistor is stored in the capacitor, while the other half is dissipated in the form of heat [96]. The energy

delivered from the power supply, E_P, is defined in Equation (3.1) [96].

$$E_P = \int_0^{\infty} I(t)V_{DD}dt = \int_0^{\infty} C_L \frac{dV}{dt} V_{DD}dt = C_L * V_{DD} \int_0^{V_{DD}} dV = C_L V_{DD}^2 \tag{3.1}$$

The energy stored in the capacitor, E_{C_L}, is given by Equation (3.2).

$$E_{C_L} = \frac{1}{2} C_L V_{DD}^2 \tag{3.2}$$

As the capacitor is discharged on the transition of input voltage from '0' to '1', the energy stored in the capacitor is released into the V_{SS}. Therefore, if a gate is observed over N clock cycles, the energy consumption, E_G, including the energy consumption of PMOS transistor and the load, becomes

$$E_G = NE_P = NC_L V_{DD}^2 \tag{3.3}$$

However, since not all gates switch every clock cycle with many registers retaining their values over long intervals, an activity factor, P_{trans}, is introduced and defines the probability of an output transition taking place. Hence the energy consumption equation becomes:

$$E_G = P_{trans} NC_L V_{DD}^2 \tag{3.4}$$

If a digital gate switches with the frequency f_{clock} over the interval T, the load is charged and discharged $T * f_{clock}$ times and the average power consumption known as the dynamic power, P_{dyn}, is given by

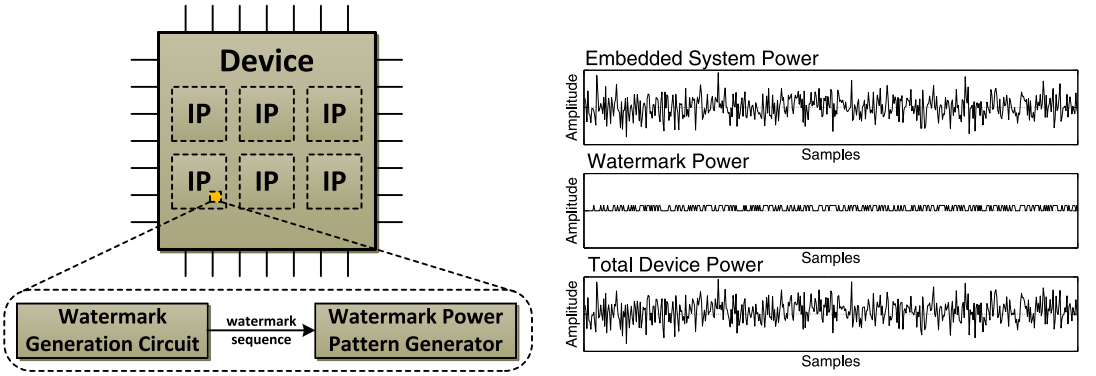
$$P_{dyn} = P_{SW} + P_{SC} \tag{3.5}$$

$$= (\frac{T f_{clock} P_{trans} E_P}{T}) + P_{SC} = (\frac{T P_{trans} C_L V_{DD}^2 f_{clock}}{T}) + P_{SC} = (P_{trans} C_L V_{DD}^2 f_{clock}) + P_{SC} \tag{3.6}$$

Where P_{SW} is the switching power and P_{SC} is the short circuit power. The P_{dyn} is dominated by P_{SW}, if I_{SC} occurs only for a short period during each transition [97], and

$$P_{dyn} = P_{trans} C_L V_{DD}^2 f_{clock} \tag{3.7}$$

In Equation (3.7), it can be seen that P_{dyn} is directly dependent on P_{trans}. Therefore, as output transitions between '0' and '1', and vice versa, the dynamic power is consumed. Such empirical behaviour creates the fundamentals for digital power watermarking.



(a) (b)

Figure 3.2: (a) Architecture of a power watermark circuit; (b) Simulation results of the effect of the watermark power signal on the total device power.

3.2 Power Watermark Architecture

A power watermark is a redundant circuit added to an existing IP block, with the aim of superimposing a weak but deterministic signal on a voltage supply rail. In Figure 3.2(a), a typical embedded system is shown. Multiple IP blocks are sub-sourced from various IP suppliers. The watermark is embedded in one of the IP blocks and consists of two circuits: a watermark generation circuit (WGC), and a watermark power pattern generator (WPPG) [8]. The WGC generates the watermark sequence which controls the WPPG load circuit. Hence, the WPPG consumes power in clock cycles where the watermark sequence is '1'. Simulation results in Figure 3.2(b), demonstrate the effect of an additional watermark circuit on device total power (in relative terms). The watermark power signal (middle) is added to the power consumed by the embedded system (top), and generates the device total power (bottom). Since the watermark power signal is a much lower amplitude, it is deeply embedded in the overall device power signal. An analytical technique is therefore required which determines the possibility and the accuracy of watermark existence. Such a technique is Correlation Power Analysis (CPA) and this has been used in this work as the fundamental watermark detection technique.

3.3 Power Watermark Detection

CPA [9] utilizes the statistical correlation technique to detect a deeply embedded power watermark signal. The detection methodology is the same as described in Section 2.5.3.4, Chapter 2. CPA requires information extracted from the measured power consumption of a device with the sampling frequency, f_s, much greater than the frequency of the system clock, f_{clk}, i.e. $f_s >> f_{clk}$. The power vector, Y, is found by averaging all the samples within a single clock cycle, Figure 3.3. The watermark model vector, X,

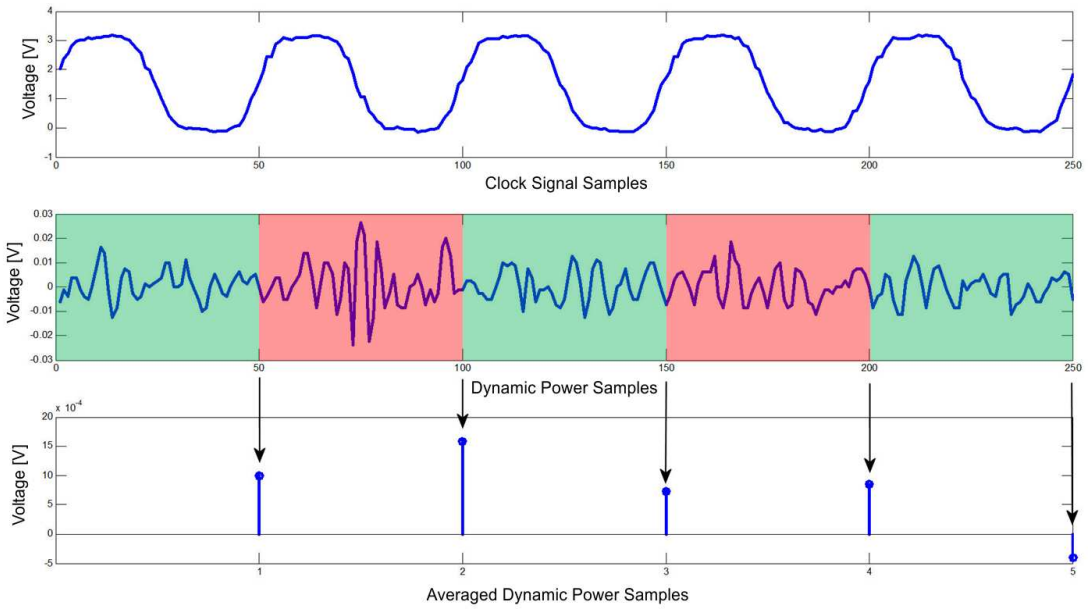


Figure 3.3: The measured clock (top) and device power (middle) signals with the sampling frequency of oscilloscope much higher than the frequency of the system. The averaged device power (bottom) represented as a single value per system clock cycle.

represents a watermark sequence. As both vectors must be of equal length, the watermark sequence is repeated many times within a vector. Next, the Pearson correlation coefficient, ρ, between the two vectors is found as in Equation (2.2) and the detection algorithm steps described in Chapter 2, Section 2.5.3.4 are repeated. The watermark is only regarded as detected if a single and significant correlation coefficient can be resolved, as shown in Figure 2.19.

3.3.1 Power Trace Length

The measured device power signal is known as the power trace. The length of the recorded with an oscilloscope power trace depends on the specification of an oscilloscope and very often a memory buffer is quite limited. For example, one of the oscilloscopes used throughout this thesis, an Agilent MSO6032 [98], allows $500k$ samples to be stored in a single capture. Therefore, if its sampling frequency is 50 times larger than the frequency of the system clock, the recorded power trace contains only $10k$ clock cycles. In order to obtain a longer power trace a trigger must be used where

$$\textit{Trigger Period} \equiv 0 \pmod{\textit{Watermark Sequence Period}} \tag{3.8}$$

A trigger signal is used to merge multiple power traces to create a longer power trace. However, to combine two different power traces one must be certain that both signals are continuous (synchronous). Therefore, each power trace must be resized to fit only

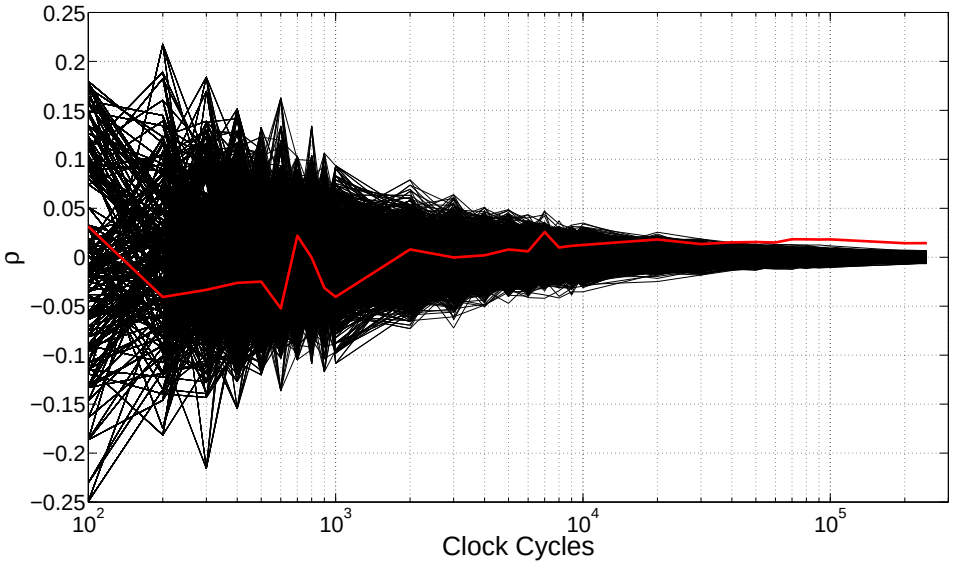


Figure 3.4: Power trace length impact on correlation coefficients.

the full periods of a watermark sequence. Henceforth, the first and the last clock cycles in a power trace always represent the same point of a watermark sequence in subsequent power traces. For example, consider the binary sequence '10011', with the period of 5 clock cycles and assume that a trigger occurs at its 3^{rd} clock cycle. For the ease of understanding, an arbitrary number of clock cycles, say 12 in this example, are used to demonstrate recorded with an oscilloscope power trace. To satisfy Equation (3.8), the trigger occurs every 15 clock cycles, since $15 \equiv 0$ (*mod* 3). The period of a trigger signal is purposely selected to be larger than the period of a recorded signal, to avoid any interference with the correlation results. Therefore, the obtained power traces are

$$\begin{aligned}
\textit{Power trace } 1 &: \textbf{10011 10011 10} \\
\textit{Power trace } 2 &: 10011\ 10011\ 10 \\
\textit{Combined power trace } (X) &: \textbf{10011 10011 10}\underline{100\ 11100\ 1110} \\
\textit{Watermark model } (Y) &: 10011\ 10011\ 10\underline{011\ 10011\ 1001}
\end{aligned}$$

If such traces are combined the continuity of a watermark sequence is disturbed, and there is a high probability that a watermark power signal will not be detected by the CPA algorithm, or it will be erroneously detected when it does not exist. As can be seen, X and Y do not match when the 2^{nd} trace is added. Therefore, to ensure that the combined power trace retains its continuity, the recorded power traces are reduced such that their lengths satisfy Equation (3.9).

$$\textit{Power Trace Period} \equiv 0 \quad (\textit{mod Watermark Sequence Period}) \tag{3.9}$$

The obtained power traces are therefore:

$$\begin{aligned} \textit{Power trace } 1 &: \textbf{10011 10011 } \sout{\textbf{10}} \\ \textit{Power trace } 2 &: 10011\ 10011\ \sout{10} \\ \textit{Combined power trace } (X) &: \textbf{10011 10011}\ 10011\ 10011 \\ \textit{Watermark model } (Y) &: 10011\ 10011\ 10011\ 10011 \end{aligned}$$

The length of the combined power trace directly influences the detection sensitivity. In Figure 3.4, an impact of the power trace length on ρ is shown. The trace length of approximately $300k$ was chosen to demonstrate the change of ρ for short trace lengths, and the effect of ρ stabilization for longer trace lengths. The last points on the graph represent Figure 2.19 (when rotated). All black lines represent ρ where X and Y are not in phase, while the red line represents ρ where X and Y are in phase. In Figure 3.4, it can be seen that until approximately $200k$ clock cycles it is impossible to find a watermark or claim a false detection. However, after $200k$ clock cycles all ρ start to stabilize and ρ obtained from X and Y not in phase tend to 0, while ρ obtained from X and Y in phase increases and reaches a stable and a significant peak value, Figure 2.19. In all experiments demonstrated in further chapters the length of the recorded power trace is approximately $300k$ clock cycles.

3.4 Third Party IP Attacks

Intellectual property cores can be protected using various techniques discussed in Chapter 2. Although many schemes are robust, provide solid security and enable a successful watermark detection, they may still be subjected to third party IP attacks. The aims of such attacks are ownership deadlock, forged authorship and counterfeit ownership. The ownership deadlock occurs when an IP designer cannot provide any solid evidence or it is not strong enough to support the ownership of an IP. The forged authorship occurs when a third party tampers with an owner's watermark and provides the proof that it was embedded in other IP. Finally, the counterfeit ownership occurs when an attacker's watermark is embedded in an owner's IP and generates a stronger signature. The means by which such security violations can be reached can be categorized into removal attacks, finding ghosts (ambiguity attacks) and forging attacks (key-copy attack) [49, 99, 100]. To fully understand the threats behind the attacks the general watermarking model [101] was proposed.

3.4.1 General Watermarking Model

The general watermarking model proposed in [101] is given as follows:

> *"We define a work to be a vector* $I = (x1, x2, ..., xn)$ *... We assume that there is a function* $Dist(\cdot, \cdot)$ *that measures the perceptual distance between two works. A watermark W is a sequence in* $\mathcal{W}^m$ *... A key K is a sequence of m binary bits.*
>
> *In our general watermarking model, there are three algorithms, a watermark generator G, a watermark embedder E, and a watermark detector D. The watermark generator G ... outputs a watermark given a key. We say that a watermark W is valid if and only if it is generated from some key K by G ... The embedder E takes an original work I and a watermark W and outputs a watermarked work* $\tilde{I}$ *... Given a work* $\tilde{I}$ *and a watermark W, the detector D declares whether W is embedded in* $\tilde{I}$ *... or not."*

In general, an IP supplier (owner) creates an original work I_O and generates a watermark W_O with a key K_O ($W_O = G(K_O)$). A watermark W_O is further embedded in an original work I_O and creates a watermarked work $\tilde{I_O}$ ($\tilde{I_O} = E(I_O, W_O)$). To prove an authorship of $\tilde{I_O}$ an owner must produce a strong evidence showing that W_O is embedded in $\tilde{I_O}$ ($D(\tilde{I_O}, W_O) = 1$). Nevertheless, an attacker can tamper with $\tilde{I_O}$ and generate his own work $\tilde{I_A}$. The similarity between $\tilde{I_O}$ and $\tilde{I_A}$ must be below a certain threshold, t, to regard both works as one ($Dist(\tilde{I_O}, \tilde{I_A}) < t$). If the tampered work is not similar to the original work ($Dist(\tilde{I_O}, \tilde{I_A}) > t$), both works are regarded as two separate IPs.

3.4.2 Types of IP Attacks

Third party IP attacks can be categorized into removal, ambiguity and key-copy attacks [49, 99, 100], and are discussed in more detail in this section.

3.4.2.1 Removal Attacks

Removal attacks are the main source of counterfeit attacks. The attacker aims to remove the original watermark W_O by complete elimination, such as watermark subtraction. However, masking of a watermark signal through an introduction of additional distortion into the system can also be equally successful, as the strength and the sensitivity of a detector are significantly reduced [99]. After removal, a watermarked work $\tilde{I_O}$ is modified to work $\tilde{I}$, where a watermark cannot be found. Therefore, an IP owner fails to provide a strong enough evidence.

$$Dist(\tilde{I_O}, \tilde{I}) < t \quad D(\tilde{I}, W_O) = 0$$

In addition to watermark removal, the attacker can also embed his own watermark W_A and claim ownership.

$$\tilde{I_A} = E(\tilde{I}, W_A) \quad Dist(\tilde{I}, \tilde{I_A}) < t \quad D(\tilde{I_A}, W_A) = 1$$

3.4.2.2 Ambiguity Attacks

The ownership deadlock can be reached through ambiguity attacks, where an attacker presents another watermark. This can be achieved by embedding a new watermark, if possible tools are available, or finding a specific pattern and presenting it as his own watermark (ghost watermark). Since both author and attacker can prove the existence of their own watermarks in the same I_O, it is impossible to determine the genuine author and therefore leads to a deadlock.

$$\begin{aligned} Owner: &\quad \tilde{I_O} = E(I_O, W_O) \quad D(\tilde{I_O}, W_O) = 1 \\ Attacker: &\quad \tilde{I_A} = E(I_O, W_A) \quad Dist(\tilde{I_O}, \tilde{I_A}) < t \quad D(\tilde{I_A}, W_A) = 1 \end{aligned}$$

Furthermore, an attacker can embed a watermark W_A in a fake work I and present this watermarked work $\hat{I_A}$ as his own. Next, the attacker demonstrates that W_O cannot be found in $\hat{I_A}$, while W_A is present. If at the same time the attacker tampers with $\tilde{I_O}$ and embeds his own watermark or finds a ghost watermark W_A and claims its authorship, the counterfeit attack can be executed and the ownership rights may be transferred to an attacker (counterfeit ownership).

$$\begin{aligned} Author: &\quad \tilde{I_O} = E(I_O, W_O) \quad Dist(\hat{I_A}, \tilde{I_O}) < t \quad D(\tilde{I_O}, W_O) = 1 \quad D(\hat{I_A}, W_O) = 0 \\ Attacker: &\quad \tilde{I_A} = E(I_O, W_A) \quad \hat{I_A} = E(I, W_A) \quad Dist(\tilde{I_A}, \tilde{I_O}) < t \quad Dist(\hat{I_A}, \tilde{I_O}) < t \\ &\quad D(\tilde{I_A}, W_A) = 1 \quad D(\hat{I_A}, W_A) = 1 \end{aligned}$$

3.4.2.3 Key-Copy Attacks

Forged authorship is a result of key-copy attack, which is based on watermarking other devices with an original author's watermark. The attacker retrieves an original watermark, W_O, from an authors work and embeds it (W'_O) in a different work I, not belonging to the author. Although the author can prove the originality of his own work, he would not be able to explain the existence of his source of security in somebody else's work.

$$W'_O = G(K_O) \quad \tilde{I_A} = E(I, W'_O) \quad Dist(\tilde{I_O}, \tilde{I_A}) > t \quad D(\tilde{I_A}, W'_O) = 1$$

3.4.3 Preventing IP Attacks

In [101], the following definitions are proposed to prevent the IP threats from occurring.

DEFINITION 1: *A watermarking scheme is t-resistant to removal attacks if for any attacker A and given any cover work $\tilde{I}$ watermarked by W, it is computationally infeasible for A to compute any work I′ such that $Dist(\tilde{I}, I') < t$ and $D(I', W) = 0$.*

The term t-resistant describes a watermarking scheme which is robust against specific third party attacks. It considers the similarity of work $\tilde{I}$, originating from an owner, and that same work after it was tampered with by an attacker (I'). If both $\tilde{I}$ and I' are not similar enough ($Dist(\tilde{I}, I') > t$), they cannot be regarded as the same work and should be treated as two separate IPs. The term computationally infeasible follows from the cryptography and means that a solution exists but it is too costly in terms of area, memory, time etc. to be regarded as a viable solution, therefore, to be pursued. In [49], the technique demonstrated that an attacker is required to re-synthesize the design after the modification, which changes the final design ($Dist(\tilde{I}, I') > t$). Furthermore in [39], the watermark removal is impossible due to mixing of the watermark logic with the test logic and therefore removal would cause a system malfunction. In general, a good IPP technique embeds a watermarking circuit as an integral part of an IP. Hence, when such a circuit is removed, further behaviour of a system is erroneous.

DEFINITION 2: *A watermarking scheme is t-resistant to ambiguity attacks if for any attacker A and any cover work $\tilde{I}$, it is computationally infeasible for A to compute a valid watermark W such that $D(\tilde{I}, W) = 1$.*

Ambiguity attacks can be executed in two ways. First, an attacker finds a ghost watermark in an owner's work $\tilde{I}$ and claims authorship. Second, an attacker embeds own watermark and therefore two authors exist. Although it might be very hard to prevent an attacker from finding ghost watermarks, many current watermarking techniques claim to achieve such protection. To prevent ghost attacks from occurring, the owner's watermark must be stronger than the attacker's watermark. In [49], a watermark is extracted through the addition of extra logic into the original design. Such logic is claimed to be more credible than any other background ghost watermarks. In [39], watermark generation is associated with generating a pseudo-random number. Therefore, it is impractical for an attacker to reverse the pseudo number generation to find ghost watermarks [39]. Furthermore, embedding watermarks into an owner's design is often hard and requires special tools. Additionally, if an attacker wishes to introduce a second watermark into the system, some reverse engineering is required especially if a design is at the lower level in a design flow. It is time consuming and requires knowledge of an original watermarking scheme to repeat the entire process without damaging the former design. Furthermore, an attacker must follow the area overhead requirements and keep the size of a counterfeit watermark to minimum [39].

DEFINITION 3 *A watermarking scheme is t-resistant to key copy attacks if for any attacker A and any cover work* $\tilde{I} = E(I, W)$ *for some original I and watermark W, it is computationally infeasible for A to compute a work I' such that* $Dist(I, I') > t$, *yet* $D(I', W) = 1$.

A key-copy attack is achieved by using an owner's signature to watermark other, lower quality designs, and effectively diminish the truthfulness of an owner. To prevent an attacker from embedding the owner's watermark in an other solution, it is best to hide a watermark in such way that it is impossible to find. To achieve such protection the area overhead of a watermark circuit must be kept to minimum. The lower the area overhead, the harder it is to find the source of a watermark. Another approach involves encryption of a watermarked design and therefore makes it unreadable to an attacker [44, 48, 49]. However, as discussed in Chapter 2, Section 2.1.2, encryption enforces the use of a particular design platform and may not be acceptable to many SoC designers and integrators, who prefer the flexibility of various design tools during the design flow [10].

3.5 Summary

In this chapter the principles of non-invasive digital watermarks have been discussed. The architecture of the current state-of-the-art digital power watermark was analyzed in Section 3.2 and it was shown that two circuits are implemented: WGC and WPPG. Since the generated watermark power signal is deeply embedded within the device power consumption, the statistical analysis technique such as CPA (Section 3.3), must be applied. However, due to the nature of the CPA algorithm the generated watermark power signal must be strong enough to produce visible and significant correlation peaks in the spread spectrum graph, Figure 2.19. At the same time, watermarking of modern embedded processors is faced with two requirements. First, the area and power overheads of the additional digital watermark circuit must be as small as possible. Second, the superimposed watermarking signal must be relatively easy to detect, hence the signal strength must be as high as possible. There is however a tradeoff between the size of the watermark circuit and therefore area and power overheads and the generated signal strength. Furthermore, due to the high transparency of the RTL design in the soft IP level, there is a tradeoff between the area overhead and the robustness of a digital watermark against third party IP attacks.

Reduction in area overhead increases the robustness of the watermarking technique against removal and key-copy attacks, since it is very hard for an attacker to find an embedded watermark. The area overhead can be minimized in both WGC and WPPG circuits. To reduce the area overhead in WGC short binary sequences can be chosen for watermarking. To minimize the area overhead in WPPG the number of switching

registers must be reduced. This, however, causes the watermarking signal strength to decrease and increases the risk of ambiguity attacks.

In embedded systems power saving is one of the fundamental requirements because many devices are operated solely from the battery. Therefore, any additional power required by the watermark circuit must be minimized to extend the battery life. The power overhead can be reduced by decreasing the area overhead as discussed above. It can also be minimized by reducing the switching activity of the watermark power pattern and therefore the average dynamic power.

3.6 Concluding Remarks

Techniques for embedding power watermarks in the soft IP must be characterized with very low area overheads to reduce the hardware implementation costs and to increase the robustness against removal and key-copy attacks. In the current state-of-the-art soft IP power watermarking technique the implementation costs and the effective robustness are significantly reduced due to large WPPG circuit. Moreover, all recently demonstrated power watermarking techniques (Chapter 2, Section 2.5.1) implement the WGC as the Pseudo-Random Number Generator (PRNG), such as Linear Feedback Shift Register (LFSR). Hence, none of the currently available techniques determines the most cost-efficient and robust sequence for watermarking soft IPs. To reduce the ambiguity attacks, one must be able to provide the evidence that his watermark is stronger than other watermarks found in an IP. Therefore, the chosen watermark sequence must produce the best correlation results with the CPA algorithm, among other watermark sequences for the same size WPPG circuit. Addressing this issue forms part of the next chapter.

Chapter 4

Characterization of Sequences for Cost-Efficient Power Watermark

The first non-invasive IPP techniques embedded power watermark circuits such as the Watermark Generation Circuit (WGC) and the load circuit, also known as the Watermark Power Pattern Generator (WPPG) [8, 58, 74] in a target IP design. The architecture of both circuits has not significantly changed throughout the literature and a Pseudo-Random Number Generator (PRNG), such as the Linear Feedback Shift Register (LFSR) is commonly used in WGC. This ensures a large number of combinations, and high level of robustness against third party IP attacks. As a result a typical WPPG implementation consists of circular shift register of a significant size.

In this chapter[1], binary sequences for the purpose of soft IP power watermarking are characterized and it is shown that several intrinsic parameters can be distinguished, which have the direct impact on correlation results and hardware implementation costs. The potential sequences for watermarking are discussed in Section 4.1. The analysis of the Pearson correlation coefficient, which forms the basis of the Correlation Power Analysis (CPA) technique is presented in Section 4.2. The intrinsic parameters of watermarking sequences are drawn and further simulation results are demonstrated in Section 4.3. The validation of simulation results on FPGA and silicon test chips is shown in Section 4.4 and Section 4.5, respectively. The cost-efficient sequences for watermark implementation are identified and the hardware implementation costs, such as area and power overheads are analyzed in Section 4.6. The susceptibility of the chosen watermark sequences to third party IP attacks is discussed in Section 4.7, while the tradeoff between short and long sequences is analyzed in Section 4.8. The chapter is summarized in Section 4.9 and final conclusions are presented in Section 4.10.

[1]The contents of this chapter have been accepted as "Sequence-Aware Watermark Design for Soft IP Embedded Processors" by Kufel *et al* in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2015*.

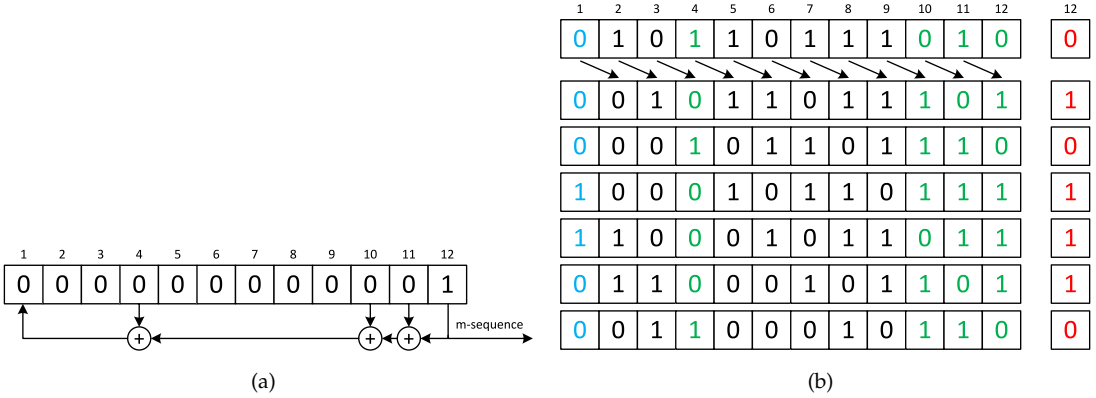


Figure 4.1: (a) Block diagram of the 12-bit LFSR; (b) States of LFSR in consecutive clock cycles.

4.1 Watermark Sequences

This chapter compares two types of binary sequences, such as sequences generated with LFSRs as demonstrated in the current state-of-the-art power watermark architecture [8], and Barker codes. The sequences have been chosen to represent the impact of various intrinsic parameters and lengths on the hardware implementation costs, robustness and detection performance. Throughout this thesis, robustness is described as a measure of the immunity of the watermarking technique against third party IP attacks. Although, the robustness cannot be explicitly measured, it can be described as the hardness of tampering with the watermark in relative terms. In case of detection performance, the commonly used CPA algorithm describes detection as the ability to observe a single and significant correlation peak in a spread spectrum (refer to Figure 2.19, Chapter 3). In this chapter, the detection performance is given as the percentage of detected watermarks when the Null Hypothesis Significance Test (NHST) is applied.

4.1.1 Linear Feedback Shift Register

The binary sequence generated with the LFSR is known as the maximum length sequence (m-sequence). The block diagram of the 12-bit LFSR is shown in Figure 4.1(a). The feedback path can be described by the following polynomial [102]:

$$\mathbf{1}x^{12} + \mathbf{1}x^{11} + \mathbf{1}x^{10} + 0x^9 + 0x^8 + 0x^7 + 0x^6 + 0x^5 + \mathbf{1}x^4 + 0x^3 + 0x^2 + 0x \qquad (4.1)$$

The degree of the polynomial is directly related to the length of the LFSR. The polynomial contains '0' and '1' coefficients, where '1' correspond to the taps of registers which are connected to XOR gates, shown in green in Figure 4.1(b). As can be seen, the feedback path is a result of XOR operation of bits 12, 11, 10 and 4. The last register in the LFSR can be used as an output and the generated sequence is known as the m-sequence (shown

in red in Figure 4.1(b)). For N number of registers, the length of the m-sequence is $2^N - 1$. For example, if a 12-bit m-sequence is used, the implementation requires 12 registers and generates a binary sequence of length $2^{12} - 1 = 4,095$. The number of '1's in any m-sequence is always one more than the number of '0's, since the length of a sequence is always odd. For example, in a 12-bit m-sequence the number of '1's is 2,048 and the number of '0's is 2,047. Such a distribution has a direct impact on the intrinsic parameter described as the *activity factor* and is discussed later in this chapter.

4.1.2 Barker Codes

The Barker codes can be characterized with high auto-correlation and low cross-correlation properties. Such properties are ideal in terms of watermarking, due to high rejection of a background noise signal and good correlation with the expected watermark model, as shown later in this chapter. The Barker codes are bipolar and contain '1' and '-1'. For the purpose of power watermarking all Barker codes must be transformed into the unipolar representation. This is achieved by changing all '-1' to '0'. In Table 4.1, the most commonly used Barker codes are shown and are used throughout this chapter.

Length	Bipolar Sequence	Unipolar Sequence
2	+1 -1	1 0
3	+1 +1 -1	1 1 0
4	+1 +1 -1 +1	1 1 0 1
5	+1 +1 +1 -1 +1	1 1 1 0 1
7	+1 +1 +1 -1 -1 +1 -1	1 1 1 0 0 1 0
11	+1 +1 +1 -1 -1 -1 +1 -1 -1 +1 -1	1 1 1 0 0 0 1 0 0 1 0

Table 4.1: Barker codes [103, 104].

The generation of Barker codes is achieved with circular shift registers. Since no feedback loop exists, the N-bit Barker code requires N registers. In comparison, a 12-bit m-sequence produces a binary sequence of 4,095 clock cycles, while using only 12 registers. The 11-bit Barker code produces a binary sequence of 11 clock cycles and requires 11 registers. Therefore, the m-sequence architecture allows generation of much longer binary sequences with a smaller number of registers. Although it has been demonstrated [105] that longer Barker codes exist, such as 32-bit and 36-bit, these sequences are not used in this work.

4.2 Pearson Correlation Coefficient Analysis

The Correlation Power Analysis utilizes the statistical correlation technique to detect deeply embedded power watermark signal. The correlation is computed using the Pearson correlation coefficient, ρ, as in Equation (4.2). In this section, the modifications to Equation (4.2) are presented, to include the intrinsic parameters of watermark sequences. Moreover, the comparison of detection performance of various sequences is presented. For the conciseness and the ease of understanding the number of steps in the expansion of Equation (4.2) have been limited. For the more in-depth step-by-step explanation please follow Appendix **A**.

$$\rho = \frac{N\sum_{i=1}^{N} X_i Y_i - \sum_{i=1}^{N} X_i \sum_{i=1}^{N} Y_i}{\sqrt{N\sum_{i=1}^{N} X_i^2 - (\sum_{i=1}^{N} X_i)^2}\sqrt{N\sum_{i=1}^{N} Y_i^2 - (\sum_{i=1}^{N} Y_i)^2}} \tag{4.2}$$

In Equation (4.2), X is the watermark model vector and can be represented by a binary sequence, Y is the power vector and contains the sampled power signal, and N is the length of both vectors. The dynamic power of a canonical static CMOS gate is linearly proportional to the *activity factor*, α (P_{trans}), which defines the probability of an output transition taking place (see Chapter 3, Section 3.1) [97]. In digital power watermarking, the *activity factor* is intrinsic to a watermark sequence, and the dynamic power is consumed in clock cycles when such sequence is '1'. Therefore, Equation (4.2) can be modified to incorporate the *activity factor*, α, of a watermark sequence. In Table 4.2, Section 4.3, various watermark sequences are considered and it is demonstrated that the *activity factor*, α, differs between sequences. To compare the detection performance of potential sequences, it is crucial to consider the α parameter. Since vectors X and Y can be out of phase, the hypothetical watermark model, X, is rotated and correlation computation is repeated [8]. Hence, vector X in Equation (4.2) can be substituted with X', which represents the rotated vector X. If both vectors are in phase, then $X' = X$. Additionally, vector Y can be represented as $X + \beta$, where X is the original vector of the hypothetical watermark model and β is the noise present in the system, such as global switching noise of digital IP blocks, environmental and measurement noise. Therefore, ρ can be represented as

$$\rho = \frac{N\sum_{i=1}^{N} X_i'(X_i + \beta_i) - \sum_{i=1}^{N} X_i' \sum_{i=1}^{N} X_i + \beta_i}{\sqrt{N\sum_{i=1}^{N} X_i'^2 - (\sum_{i=1}^{N} X_i')^2}\sqrt{N\sum_{i=1}^{N} (X_i + \beta_i)^2 - (\sum_{i=1}^{N} X_i + \beta_i)^2}} \tag{4.3}$$

Since both vectors X and X' represent a binary sequence, the sum of all terms in a vector ($\sum X_i$ and $\sum X_i'$) is the Hamming weight, H, of a sequence. Moreover, as both X and

X' represent the same but rotated binary sequence, the Hamming weight is the same. Furthermore, if both vectors X' and X are in phase, it is given by

$$\sum_{i=1}^{N} X_i' X_i = \sum_{i=1}^{N} X_i = H \tag{4.4}$$

However, since vectors X' and X can be out of phase, Equation (4.4) is modified by adding the *overlapping factor*, θ as follows

$$\sum_{i=1}^{N} X_i' X_i = \theta H \tag{4.5}$$

To illustrate this point, consider 2 watermark model vectors, where one is the cyclically rotated version of the other.

$$X: \quad 1111000000\ 1111000000 \tag{4.6}$$

$$X': \quad 0111100000\ 0111100000 \tag{4.7}$$

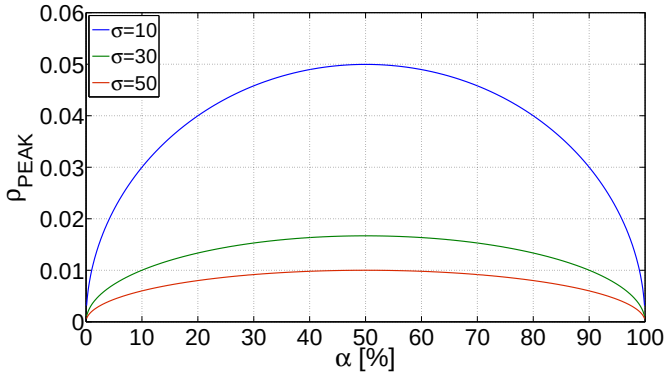
$$\sum_{i=1}^{N} X_i X_i' = 6 = \theta H \quad \theta = 6/8 = 0.75 \tag{4.8}$$

The *overlapping factor*, θ, is 1 when both vectors are in phase, and $\theta < 1$ when both vectors are out of phase, including other rotations of vector X'. In Table 4.2, Section 4.3, θ_{MAX}, is shown and describes the highest *overlapping factor*, θ, under the assumption that X and X' are not in phase. As can be seen, θ_{MAX} varies significantly between sequences. Furthermore, the Hamming Weight, H, can be substituted as the product of the *activity factor*, α, and the length N of vectors, since $\alpha = \frac{H}{N}$. Finally, the Pearson's correlation coefficient, ρ, can be described as a function of *activity*, *overlapping factor*, and both X and X' vectors as follows

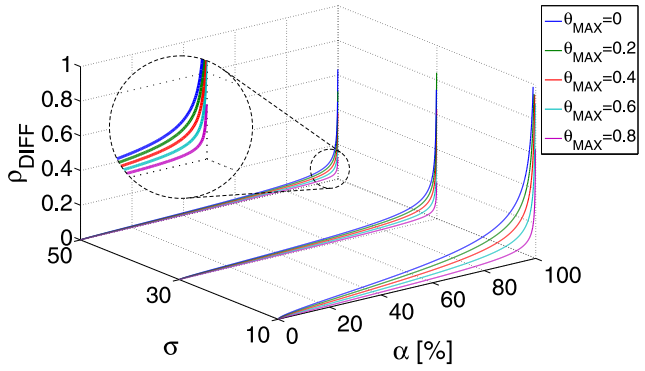
$$\rho(\alpha, \theta, X_i, X_i') = \frac{N\alpha(\theta - \alpha) + \sum_{i=1}^{N}(X_i'\beta_i) - \alpha\sum_{i=1}^{N}\beta_i}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \sum_{i=1}^{N}\beta_i(2\alpha N + \sum_{i=1}^{N}\beta_i)}} \tag{4.9}$$

The terms $\sum_{i=1}^{N}(X_i'\beta_i)$ and $\sum_{i=1}^{N}(X_i\beta_i)$ in Equation (4.9) depend on the position of '1' in a watermark sequence. However, since $N >> 1$, Equation (4.9) can be simplified to

$$\rho(\alpha, \theta, X_i, X_i') = \frac{N\alpha(\theta - \alpha)}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N\sum_{i=1}^{N}\beta_i^2}}, \quad N >> 1 \tag{4.10}$$



(a)



(b)

Figure 4.2: Influence of *activity factor*, α, and *overlapping factor*, θ, on (a) maximum correlation coefficient, ρ_{PEAK}, and (b) correlation coefficient difference, ρ_{DIFF}.

By definition, in a spread spectrum a single correlation peak should be distinguishable to consider a watermark detected [9]. The maximum correlation coefficient, ρ_{PEAK}, is

$$\rho_{PEAK} = \rho(\alpha, 1) = \frac{N\alpha(1-\alpha)}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N\sum_{i=1}^{N}\beta_i^2}}, \quad N >> 1 \tag{4.11}$$

It is expected, as shown in Chapter 3, Figure 2.19, that the highest ρ_{PEAK} occurs when both vectors X and X' are in phase, i.e. $\theta = 1$. In the noiseless environment, ρ_{PEAK} is 1 for all sequences, since $\beta = 0$ and

$$\rho_{PEAK} = \rho(\alpha, 1) = \frac{N\alpha(1-\alpha)}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha)}} = 1 \tag{4.12}$$

In Figure 4.2(a), MATLAB simulations of Equation (4.11) are shown with various watermark sequences and noise levels. The noise vector, β, consists of normally distributed random values. The frequency spectrum is shown in Section 4.3.1, Figure 4.4(a). Therefore, it approximates to white noise, to represent the global switching noise of digital

Table 4.2: Parameters of Watermarking Sequences.

Watermark Sequence	**Bit Length**	**Period (clock cycles)**	***activity factor*** α [%]	**Maximum** θ_{MAX}
Barker codes	2-bit	2	50%	0
	3-bit	3	66.6%	0.5
	4-bit	4	75%	0.667
	5-bit	5	80%	0.75
	7-bit	7	57.15%	0.5
	11-bit	11	45.45%	0.4
m-sequence	6-bit	63	50.8%	0.5
	8-bit	255	50.2%	0.5
	12-bit	4095	50.01%	0.5

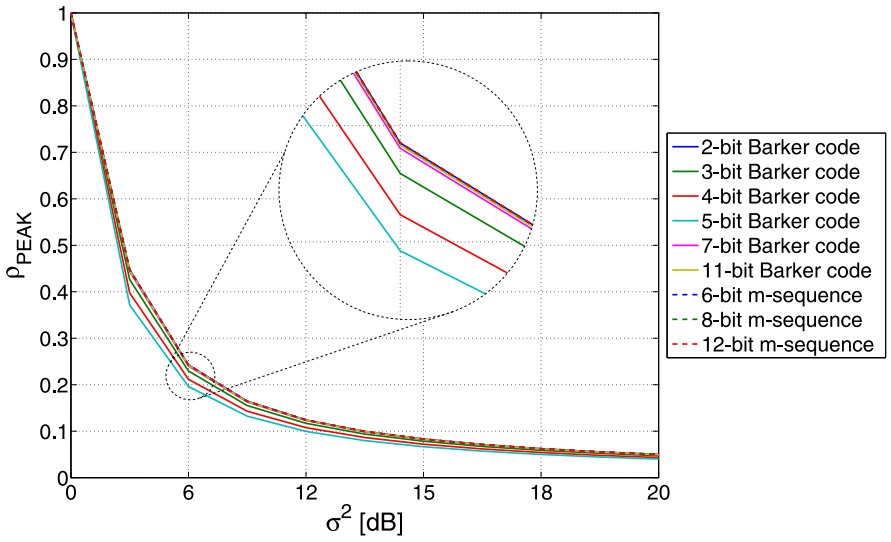
IP blocks, environmental and measurement noise. Since the mean value of β is 0, the power follows the variance, σ^2. To increase the power of β (Figure 4.2(a)), the standard deviation, σ, of the generated random values is increased. From Equation (4.11), ρ_{PEAK} is principally influenced by the *activity factor*, α. Due to the parabolic shape of the graph, watermark sequences with $\alpha \approx 50\%$ produce the highest ρ_{PEAK}. As the noise increases, the term $N \sum_{i=1}^{N} \beta_i^2$ becomes dominant and the graph becomes flatter. The ρ_{PEAK} decreases and watermark sequences produce similar results.

In practice, a power supply noise and measurement error give rise to undesirable spurious correlation coefficients. If such spurious coefficients are considered as the system noise floor, the correlation coefficient difference, ρ_{DIFF}, can be described as the distance from ρ_{PEAK} to the noise floor, as in Equation (4.13).

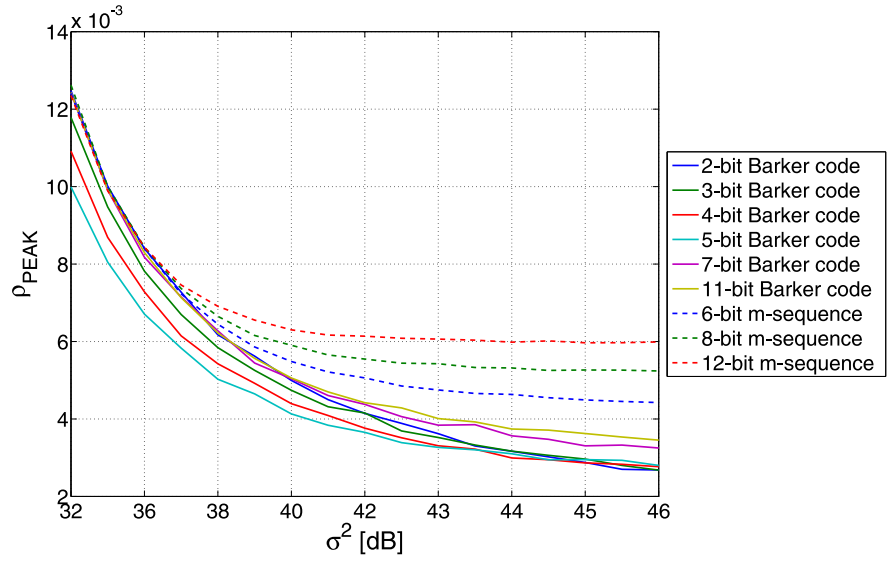
$$\begin{aligned} \rho_{DIFF} &= \rho_{PEAK} - \rho(\alpha, \theta, X_i, X_i') \\ &= \frac{N\alpha(1-\theta)}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N\sum_{i=1}^{N}\beta_i^2}}, \quad \theta < 1, N >> 1 \end{aligned} \tag{4.13}$$

The simulation of Equation (4.13) with various watermark sequences is shown in Figure 4.2(b). As expected, ρ_{DIFF} is influenced by both α and θ_{MAX} parameters. If the noise standard deviation, σ, is increased, ρ_{DIFF} approaches 0. This means that there is no distinctive correlation peak in a spread spectrum, and a watermark cannot be found.

In this section, the intrinsic parameters of sequences, such as the *activity factor*, α, and the *overlapping factor*, θ, have been introduced, and their impact on correlation coefficient values has been demonstrated. The significance of Equation (4.11) and Equation (4.13) is the ability to design embedded power watermarks with low overheads. In the following sections, the sequences for power watermarks are presented and the discussion, supported by simulation results, of differences between ρ_{DIFF} and ρ_{PEAK} is provided. Furthermore, the detection performance of watermark sequences is established.



(a)



(b)

Figure 4.3: Simulation results of watermark sequences comparison, based on maximum correlation coefficient, ρ_{PEAK}.

4.3 Simulation Results

To validate the theory of Section 4.2, various watermark sequences have been simulated and compared with the commonly used maximum length sequence (m-sequence) [8]. The summary of sequences and parameters is shown in Table 4.2. In Figure 4.2(b), ρ_{DIFF} is highest for lowest values of θ, when X and X' are not in phase. Therefore, the maximum values of the *overlapping factor*, θ_{MAX}, are described for any given sequence in Table 4.2, to account for other spurious correlation values which are highest for θ_{MAX}. As can be seen, the α, and the θ_{MAX}, vary for all Barker codes. However, for

m-sequences α decreases and tends to 50%, and θ_{MAX} is constant and equals 0.5, when the length of the sequence increases.

4.3.1 Maximum Correlation Coefficient

The maximum correlation coefficient, ρ_{PEAK}, of watermark sequences at various noise levels is shown in Figure 4.3(a) and Figure 4.3(b). For noise signals with relatively low power, Figure 4.3(a), watermark sequences with $\alpha \approx 50\%$ produce the highest ρ_{PEAK}. This is as expected based on results in Section 4.2, Figure 4.2(a). As the system noise level increases, the watermark signal-to-noise ratio (SNR) decreases. Finally, it reaches the point where the watermark power signal is too low to be reliably detected. In the marginal case, results of ρ_{PEAK} are dictated by the robustness of a watermark sequence against the correlation to the noise present in the system. Therefore, the results can be considered as the *noise-to-sequence* correlation, ρ_{NOISE}. In Figure 4.3(b), it can be seen that when the noise power approaches $38dB$, ρ_{PEAK} of m-sequences are higher than other sequences. As the noise level is further increased, the length of a sequence determines the *noise-to-sequence* correlation, with longer sequences producing higher ρ_{PEAK}. To understand the reason of such behaviour, consider ρ in Equation (4.2) when no watermark is present. Vector Y, which originally represents the measured power signal and contains the watermark model X and system noise β, is replaced with β, since no watermark exists. If the substitution of the Hamming Weight (Section 4.2) is followed, ρ_{NOISE} can be represented as

$$\rho_{NOISE} = \frac{\sum_{i=1}^{N} X'_i\beta_i - \alpha \sum_{i=1}^{N} \beta_i}{\sqrt{\alpha(1-\alpha)}\sqrt{N\sum_{i=1}^{N}\beta_i^2 - (\sum_{i=1}^{N}\beta_i)^2}} \tag{4.14}$$

Equation (4.14) was analyzed with watermark sequences of various lengths and α. It was found that α had no effect on the *noise-to-sequence* correlation, ρ_{NOISE}, for very low SNR between the watermark power and noise signals. Therefore, the diminishing effect of α on correlation coefficients can be observed, as the noise power is increased.

The period of a watermark sequence (M) determines the number of frequency components in the watermark model. Short sequences contain only a few frequency components, Figure 4.4(b). As the length of a sequence increases, more frequency components appear in the frequency spectrum of the watermark model, Figure 4.4(c). In Figure 4.4(a), the frequency spectrum of the noise signal obtained from simulations is shown. If the convolution of the watermark model and noise signal is considered, Figure 4.4(d) and Figure 4.4(e), the overlapping area between the two signals increases with the length of a sequence. Therefore, more information contained within the noise signal is

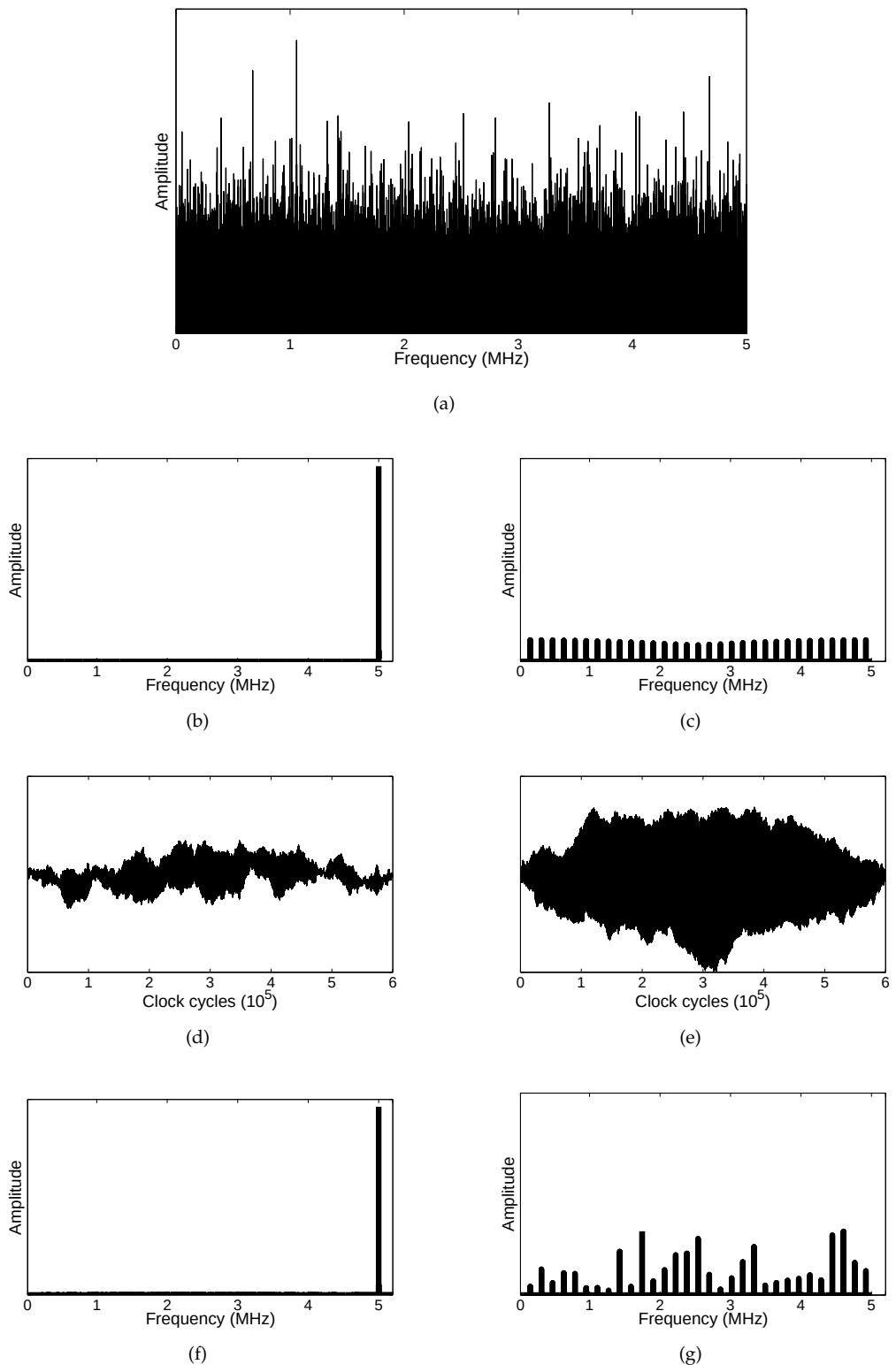
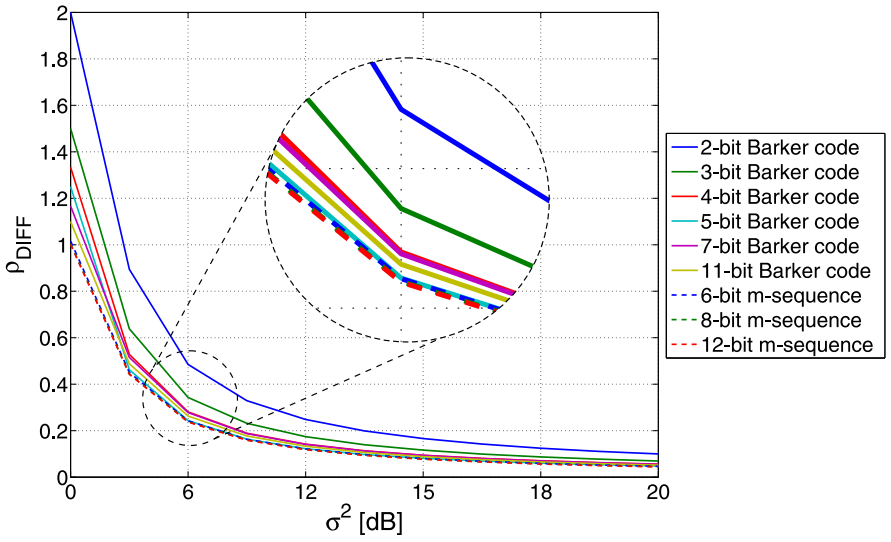
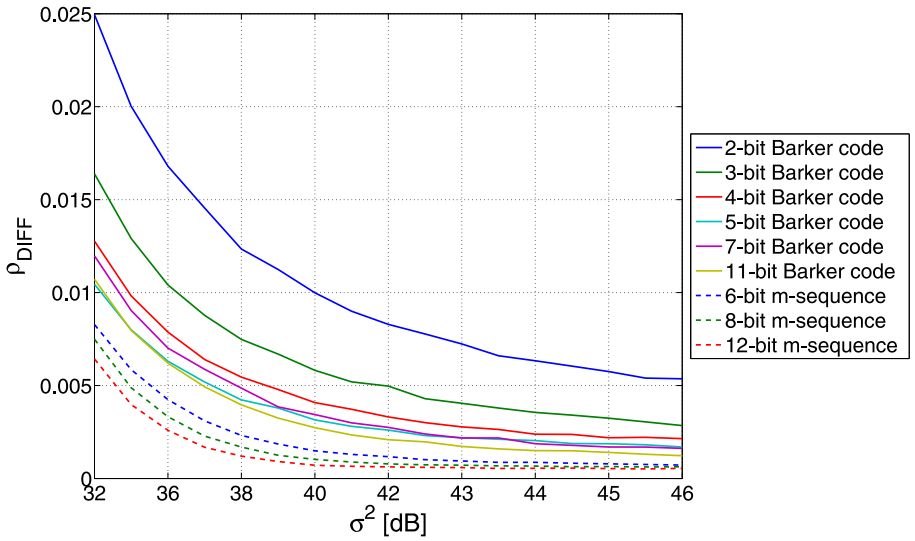


Figure 4.4: Frequency spectra of (a) noise β, (b) 2-bit Barker code and (c) 6-bit m-sequence. Convolutions of (d) 2-bit Barker code and (e) 6-bit m-sequence with noise β. Frequency spectra of convolved (f) 2-bit Barker code and (g) 6-bit m-sequence with noise β.



(a)



(b)

Figure 4.5: Simulation results of watermark sequences comparison based on coefficient difference, ρ_{DIFF}.

retained, Figure 4.4(f) and Figure 4.4(g). At the same time, the correlation between the two signals increases, which causes ρ_{PEAK} to be higher for longer m-sequences.

4.3.2 Correlation Coefficient Difference

Equation (4.13) demonstrates that ρ_{DIFF} is determined by α and θ (θ_{MAX}) parameters. It should be noted that the highest ρ_{DIFF} occur for watermark sequences with the highest α and lowest θ_{MAX}, Figure 4.2(b). In Figure 4.5(a), the 2-bit Barker code produces

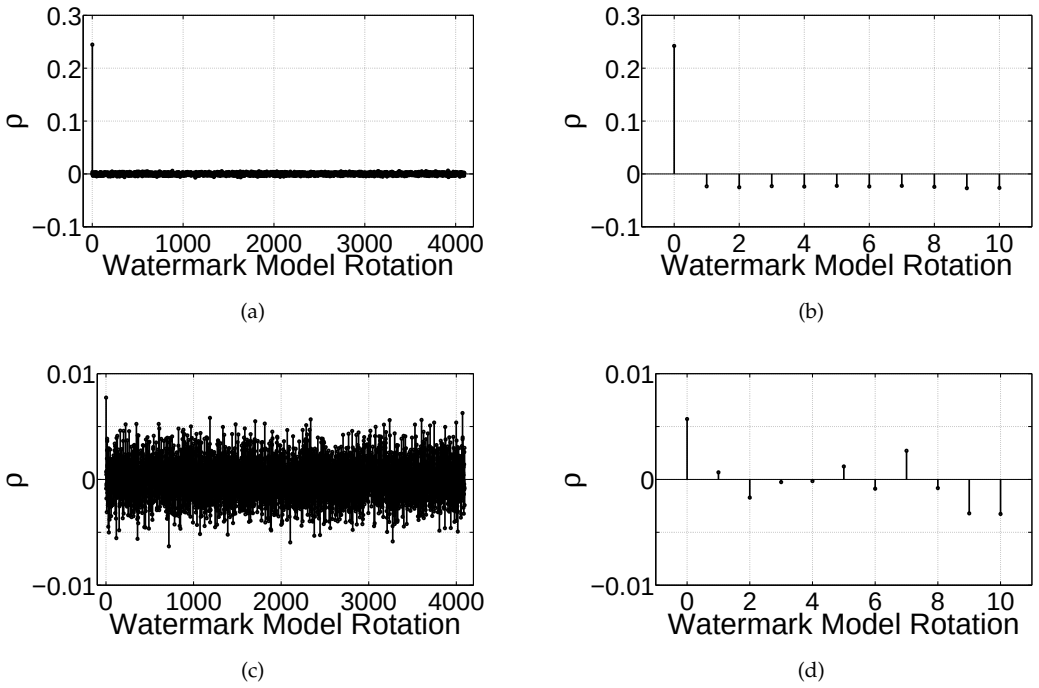


Figure 4.6: Spread spectra of (a, c) 12-bit m-sequence and (b, d) 11-bit Barker code for noise with power of $6dB$ and $40dB$, respectively.

the highest ρ_{DIFF}, since $\alpha = 50\%$, and $\theta_{MAX} = 0$. Maximum length sequences (m-sequences) produce much lower ρ_{DIFF} than most of the Barker codes, since α reduces when M is increased and tends to 50%, while θ_{MAX} remains at 0.5. The difference between subsequent m-sequences is minimal due to the same θ_{MAX} and similar α. However, since longer watermark sequences contain more frequency components that correspond with the noise signal, there are multiple correlation coefficients with values close to ρ_{PEAK}. This causes ρ_{DIFF} to be much lower as the watermark sequence length increases, Figure 4.5(b).

In Figure 4.3 and Figure 4.5, the relationship between ρ_{DIFF} and ρ_{PEAK} varies with power of the generated noise signal and watermark sequences. In Figure 4.6(a) and Figure 4.6(b), the 12-bit m-sequence and 11-bit Barker code are shown, for $6dB$ noise signal. As can be seen for 12-bit m-sequence (Figure 4.6(a)), ρ_{DIFF} and ρ_{PEAK} have similar values (0.25), since the noise floor in the spread spectrum is close to 0. However, for 11-bit Barker code (Figure 4.6(b)) ρ_{DIFF} (0.27) is higher than ρ_{PEAK} (0.25). Increasing the noise power to $40dB$ (Figure 4.6(c) and Figure 4.6(d)), causes ρ_{DIFF} to be of much lower amplitude than ρ_{PEAK}, since the noise floor increases and gets closer to ρ_{PEAK}. In Figure 4.6(c), the noise floor is of similar amplitude as ρ_{PEAK} in Figure 4.6(d). This is as expected, since ρ_{PEAK} is higher for longer sequences, for very low SNR (Figure 4.3(b)). However, these are separate test cases and as shown in Section 4.3.3, the threshold levels differ based on a sequence. Therefore, as shown in Figure 4.6(d), at $40dB$ 11-bit

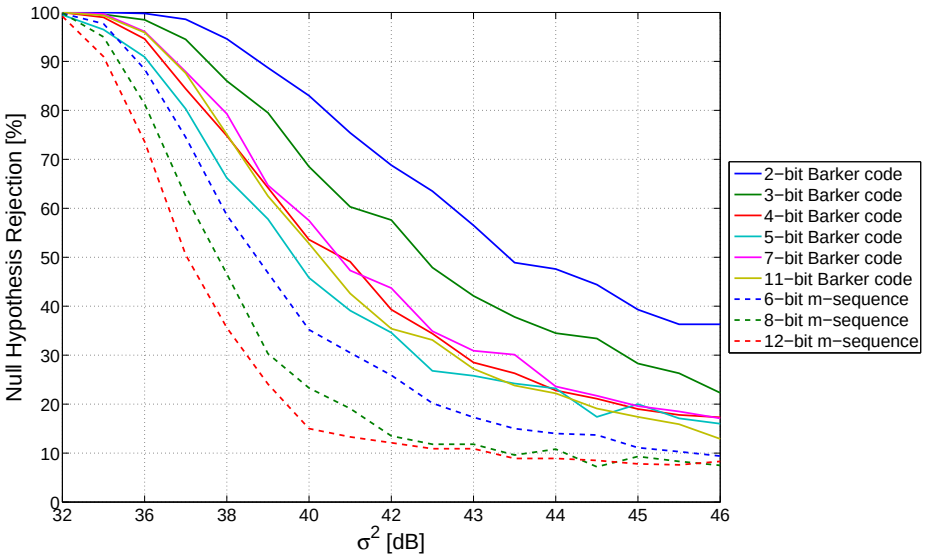


Figure 4.7: Null Hypothesis Significance Test of simulated watermark sequences with 5% significance level.

Barker code is clearly detectable. However, in Figure 4.6(c) the noise floor in the spread spectrum is too high to detect the 12-bit m-sequence.

4.3.3 Null Hypothesis Significance Test

In Section 4.3.1 and Section 4.3.2, the influence of watermark sequence length on *noise-to-sequence* correlations was demonstrated. In Figure 4.3(b), results of ρ_{PEAK} are higher for longer m-sequences as the noise level increases. However, based on results of ρ_{DIFF} in Figure 4.5(b), other correlation coefficients exist which make the spread spectrum more even, and no significant peaks can be distinguished, at high background noise levels. To compare the detection performance of watermark sequences, the Null Hypothesis Significance Test (NHST) [75] was performed for each sequence. The percentage of rejected null hypotheses was found by applying a 5% threshold to results, where the null hypothesis states that the watermark does not exists. The ρ_{DIFF} was chosen to describe the detection performance of a watermark sequence, since it considers multiple correlation values in a spread spectrum. If ρ_{DIFF} is above the threshold, the null hypothesis can be rejected with 5% possibility of a false alarm. This means that there is a 5% possibility of detecting a watermark which does not exist. To minimize the possibility of false alarms higher threshold levels can be used. To determine the threshold for each watermark sequence separately, the simulation was repeated 100 times with no watermark present in the system. The null hypothesis was found by plotting a distribution of ρ_{DIFF} from which a 5% threshold level was determined [75]. The process was repeated 10 times and the average threshold level was found for each watermark sequence. Next, the

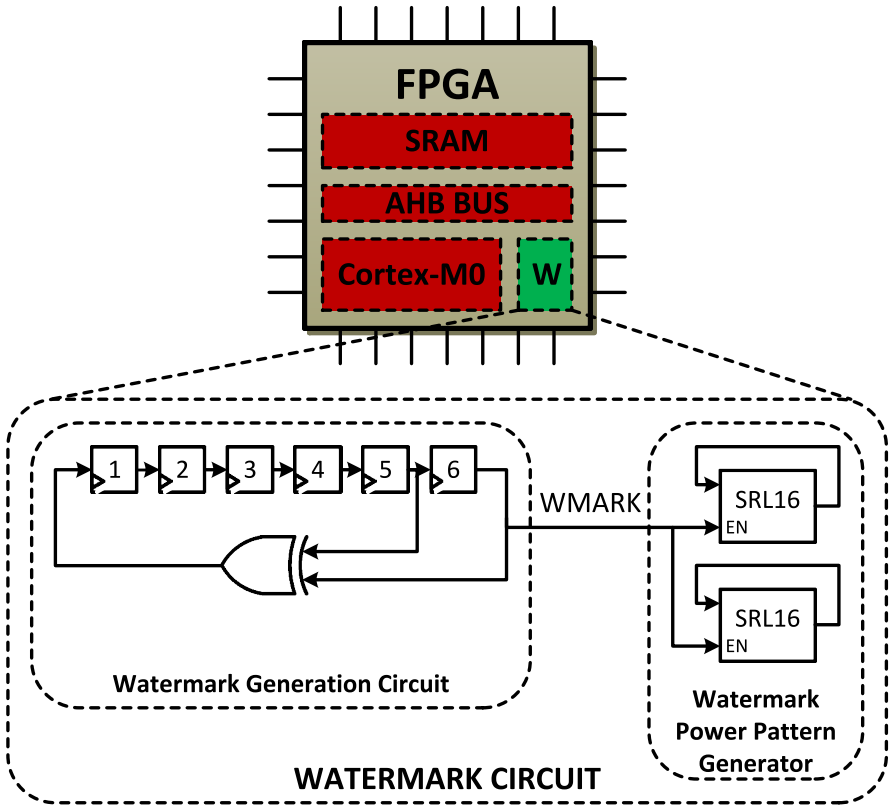


Figure 4.8: Architecture of a 6-bit m-sequence power watermark circuit implemented on FPGA.

watermark was added and the simulations were repeated in the same way. The threshold levels were applied to each sequence, to determine the detection performance, as in Figure 4.7. The difference between Barker codes is clearly distinguishable, however it is not as distinctive as in Figure 4.5(b). This demonstrates the higher *noise-to-sequence* correlations of shorter sequences. Nevertheless, as the length of sequences increases, the null hypothesis rejection ratio decreases most quickly for longer m-sequences. When noise reaches $46dB$, most sequences approach the 5% threshold.

4.4 FPGA Validation

To verify the theory and simulation results of Section 4.2 and Section 4.3, an ARM® Cortex®-M0 microprocessor IP core was implemented on a Xilinx Virtex-II Pro XC2VP30 FPGA, Figure 4.8. The FPGA was used for illustration purposes to demonstrate the relationship between the WPPG size and detection performance. Furthermore, it is used to determine the ratio of the WPPG size between watermarking sequences. To determine if such ratio would change if a newer device was used, consider Equation (3.7). As discussed in Chapter 3, Section 3.1, the dynamic power consumption depends on the *activity factor*, α, load capacitance, supply voltage and frequency. The α parameter is intrinsic to a watermark sequence, hence it does not change when using other devices.

Table 4.3: Area of Watermark Circuit Implemented on FPGA.

Watermark Sequence	**WPPG Registers (SRL16)**	**FPGA Slices**	**Area Overhead**
ARM® Cortex®-M0 IP core	-	2,696	-
7-bit Barker code	-	4	-
	8	**12**	**0.45%**
	16	20	0.74%
	32	36	1.34%
11-bit Barker code	-	6	-
	8	**14**	**0.52%**
	16	22	0.82%
	32	38	1.41%
6-bit m-sequence	-	5	-
	8	13	0.48%
	16	21	0.78%
	32	**37**	**1.37%**
8-bit m-sequence	-	5	
	8	13	0.48%
	16	21	0.78%
	32	**37**	**1.37%**
12-bit m-sequence	-	8	
	8	16	0.59%
	16	24	0.89%
	32	**40**	**1.48%**

The supply voltage is most likely expected to decrease and frequency to increase, when moving to a newer, smaller process. As the gate capacitance decreases in newer processes, the change in supply voltage and frequency would directly impact the amount of WPPG registers. Nevertheless, the change of parameters would apply equally to different watermarking sequences and the WPPG ratio would remain constant. Therefore, the performance of an FPGA does not directly influence such a relationship.

4.4.1 Watermark Circuit Architecture

The architecture of the watermark generation circuit (WGC) depends on the watermark sequence. The LFSRs (Section 4.1.1) were used for m-sequences, and simple circular shift register were used for Barker codes (Section 4.1.2). The output from the last register ('*WMARK*') serves as the clock enable signal for the watermark power pattern generator (WPPG). The WPPG dissipates power due to shifting data in the flip-flops, when enabled by the *WMARK* signal. In the Xilinx FPGA, a single look-up table (LUT) can be configured as a 16-bit shift register (SRL16) [106]. To increase the SNR between the watermark power signal and the system noise signal, the number of SRL16 blocks is increased. To generate the maximum power in clock cycles when watermark sequence is '1', each SRL16 block is pre-initialized with '1010...' sequence. In Table 4.3, the size

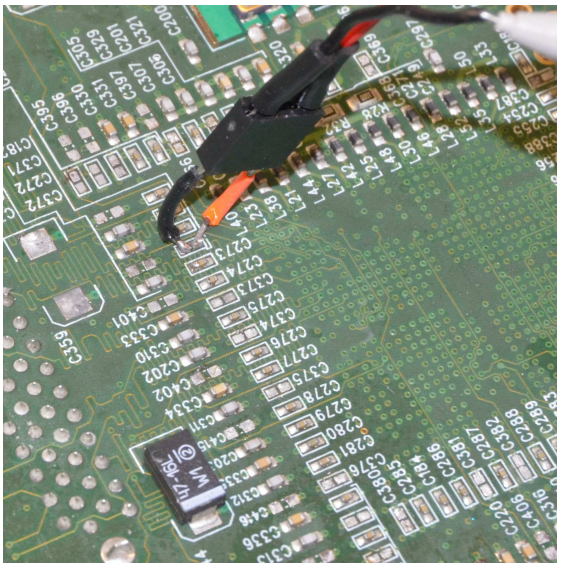


Figure 4.9: Voltage measurement at the back of the FPGA board.

of the watermark circuit implemented on FPGA is shown for deterministic sequences, discussed in Section 4.4.2.1 and various WPPG sizes.

4.4.2 Experimental Results

The operating frequency of the entire system implemented on FPGA was $10MHz$. The voltage was measured very close to the FPGA power supply pins, using an Agilent Technologies MSO6032A oscilloscope, at a sampling frequency of $500MHz$, Figure 4.9. Therefore 50 samples per single clock cycle were averaged, to obtain the power vector, Y. The watermark sequences discussed in Section 4.1 were evaluated while running the Dhrystone benchmark, which is a compact and widely available benchmark in the public domain to measure the performance of a processor. It reflects the activities of the integer IP processor core, such as integer arithmetic, string operations, logic decisions and memory accesses in a general computing application [107] and it is one of the most common benchmarks used in the industry. To avoid the influence of the design placement variation on detection performance, the design was constrained such that the ARM® Cortex®-M0 microprocessor IP core and the WPPG circuit were mapped to the same LUTs throughout the experiment. The length of both X and Y vectors was approximately $300k$ clock cycles. The trigger signal was generated on FPGA with its period of $765,765$ clock cycles. Such specific period satisfies all tested sequences (Chapter 3, Equation (3.8)).

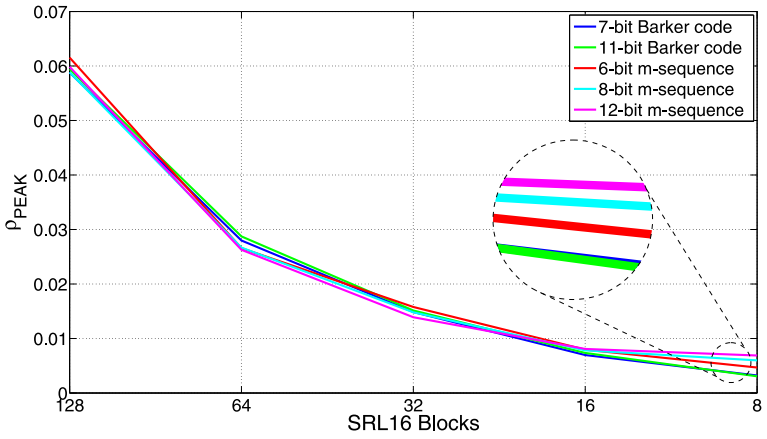


Figure 4.10: FPGA experimental results of ρ_{PEAK}.

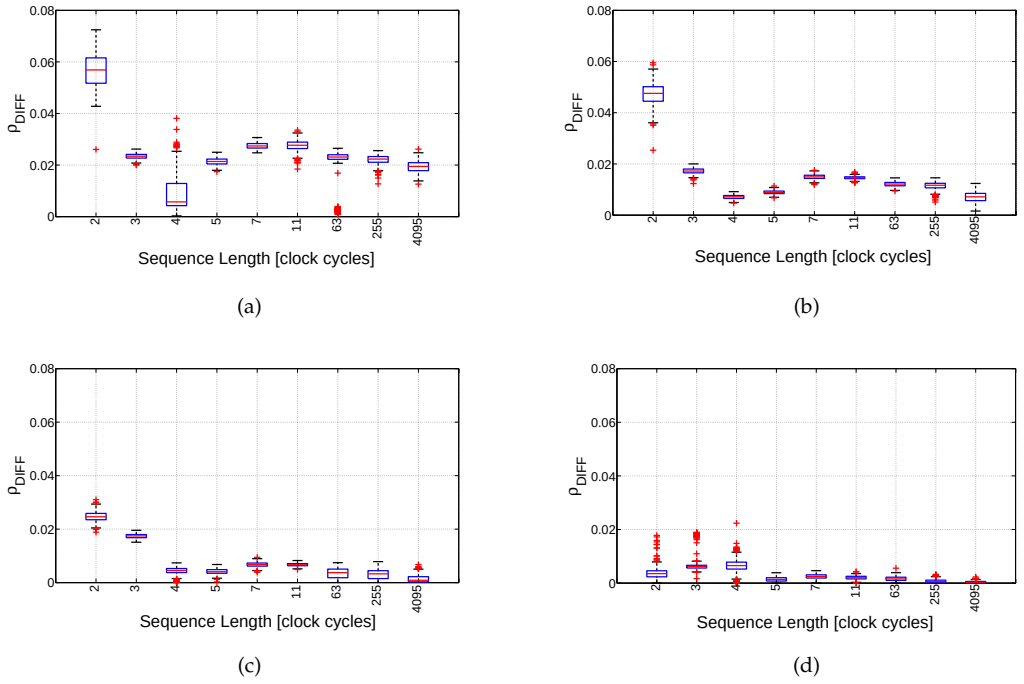


Figure 4.11: Box plots of ρ_{DIFF} at various sizes of WPPG circuit on FPGA: (a) 64 SRL16, (b) 32 SRL16, (c) 16 SRL16, (d) 8 SRL16.

4.4.2.1 Repeatability

In Figure 4.10, FPGA measurements of ρ_{PEAK} are shown. The results match the simulation results of Section 4.3, Figure 4.3(b). As the SNR between the watermark power signal and the system noise signal is high (128 to 16, SRL16), all watermark sequences generate similar ρ_{PEAK}. As the SNR decreases, the impact of α diminishes and the length of the watermark sequence becomes the major factor. Therefore, longer sequences produce higher ρ_{PEAK}.

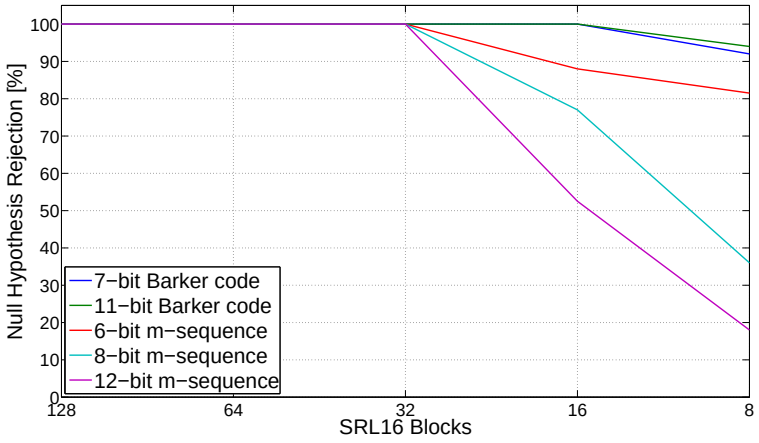


Figure 4.12: Null Hypothesis Significance Test of watermark sequences implemented on FPGA, with 5% threshold level.

The simulation results in Section 4.3 have shown a clear differentiation between short and long watermark sequences. Moreover, no significant variations have been found between results, when simulated multiple times. However, experimental FPGA results indicate that some watermark sequences are less repeatable than others. This means that when the experiment is repeated multiple times, distributions of ρ_{PEAK} or ρ_{DIFF} vary from test to test. In Figure 4.11, the variance of results is shown in terms of box plots, for various sizes of WPPG circuit. Each box represents the combined distributions of ρ_{DIFF}, obtained from multiple tests. As in Section 4.3.3, the ρ_{DIFF} is used, since it considers other correlation coefficients in the spread spectrum. Nevertheless, the same variance occurs for ρ_{PEAK}. The test was repeated 3 times for each watermark sequence and the 100 point distributions were found for each test. The FPGA was re-configured between each test, and the delay between the start of the program and the start of the watermark circuit was modified. This causes the noise characteristics to vary between consecutive tests and correlate differently for some sequences. As can be seen in Figure 4.11, short watermark sequences correlate with much higher variance for most WPPG circuit sizes. Additionally in Figure 4.11(d), medians of very short watermark sequences do not match the simulation results discussed in Section 4.3, Figure 4.3 and Figure 4.5. According to the simulation results shorter watermark sequences produce higher ρ_{DIFF}, which are not observed for 2, 3, 4, and 5 bits Barker codes. As the period of a watermark sequence increases, the variance of results is significantly lower. Results become deterministic and the expected behaviour can be predicted.

4.4.2.2 Detectability

To determine the detection performance, the Null Hypothesis Significance Test (NHST) [75] was performed on FPGA measurements. The 5% threshold levels were found for

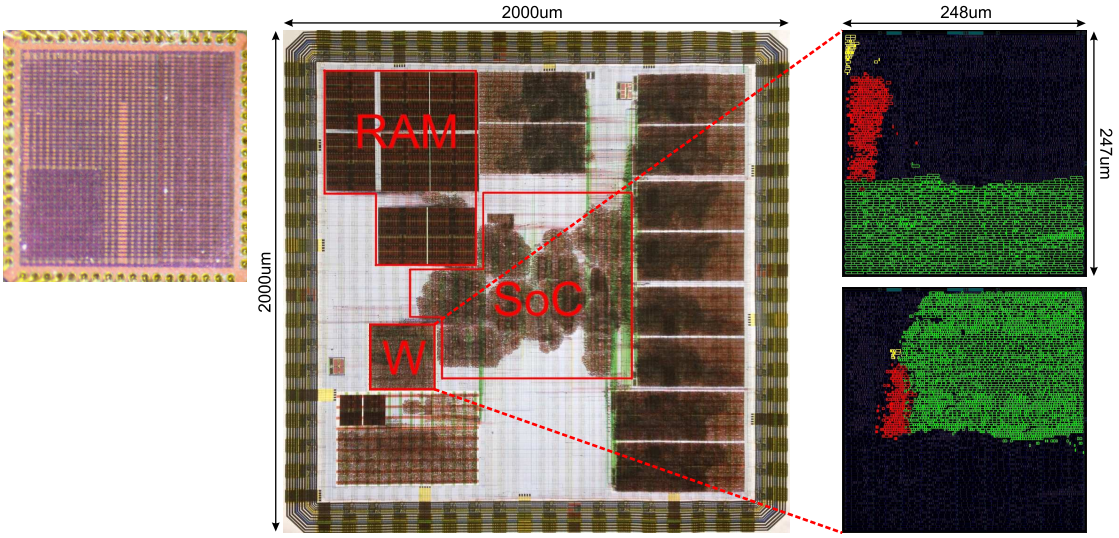


Figure 4.13: The layout (middle), die photo (left) and the hard macro watermark module ('W'), highlighting the watermark circuit (top right) and noise generator (bottom right), integrated in chip I.

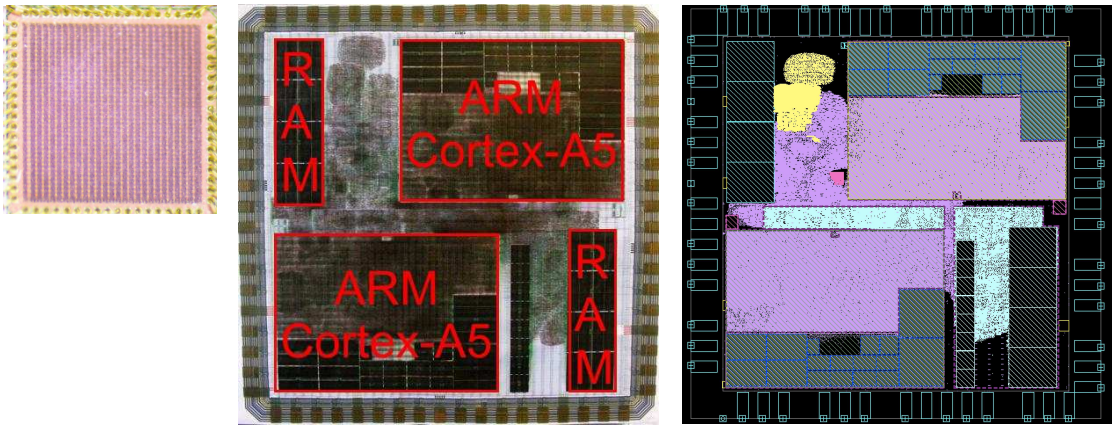


Figure 4.14: Layout (middle) and die photo (left) of test chip II, with the soft macro watermark module embedded in the SoC, highlighted in yellow (right).

each watermark sequence, when a watermark signal was not present. Furthermore, the thresholds were applied to the results in Figure 4.11 and the average null hypothesis rejection ratio was found, Figure 4.12. The deterministic watermark sequences discussed in Section 4.4.2.1 are considered. Results in Figure 4.12, match the simulation results of Section 4.3, Figure 4.7. Longer period watermark sequences such as 12-bit m-sequence approach the threshold level much faster than shorter sequences, such as 7 and 11 bits Barker codes. Therefore, it is possible to reduce the area and power overheads with shorter watermark sequences, through reduction of WPPG registers (see Section 4.6).

4.5 Silicon Validation

To aid with experimental results and investigate the impact of process variation (PV) on watermark sequence detection results, two ASIC designs were fabricated.

4.5.1 Test Chips Overview

The test chips were fabricated through the multi-project 'Mini@sic' scheme by Europractice using the $65nm$ low leakage CMOS technology, with the nominal operating voltage of $1.2V$. The designs were completed using industry standard EDA tools. The Synopsys tool suite version E-2010.12 was used during the fabrication, including Design Compiler for synthesis, IC Compiler for place and route and PrimeTime and VCS for verification. Design rule checking (DRC) and layout versus schematic (LVS) sign-off were completed with Calibre from Mentor Graphics. The available die size for the entire project was $2mm\ x\ 2mm$, Figure 4.13.

The architecture of the embedded watermark module is the same on both test chips. In the first design (chip I), the watermark module was integrated as a hard macro block, on a separate power domain ('W' in Figure 4.13). The final layout was $248\ x\ 247\mu m$, with the design signed off at the maximum operational frequency of $200MHz$ [2]. The SoC consists of the ARM® Cortex®-M0 microprocessor IP core, along with an on-chip bus and numerous commercial IP blocks including the ASCII Debug Protocol (ADP). The Cortex®-M0 is embedded in the SoC with the data and instruction space mapped to the RAM memory. The ADP unit is used as a means of communication and configuration and allows to download the program onto the on-chip memory. It communicates with an off chip USB protocol converter over an 8-bit bus and provides read and write access to the entire memory map of the SoC via USB. All other unmarked blocks are a part of a separate experiment and are not relevant to this thesis. In the second design (chip II), the watermark module was integrated from an RTL description, Figure 4.14. Therefore, the design was propagated through the entire design flow, which is closer to the intended usage scenario, when embedding watermarked soft IP. The chip consists of dual core ARM® Cortex®-A5 microprocessor IP core and caches. The SoC, shown as the unmarked circuitry, consists of the ARM® Cortex®-M0 microprocessor IP core, ADP unit and the watermark module.

4.5.2 Watermark Module Architecture

The watermark module architecture is the same in both chips, Figure 4.15. It is a fully compatible Advanced High Performance bus (AHB) slave, which means that it can be

[2] The worst case characterization corner with a critical path length of $4.777ns$ was estimated.

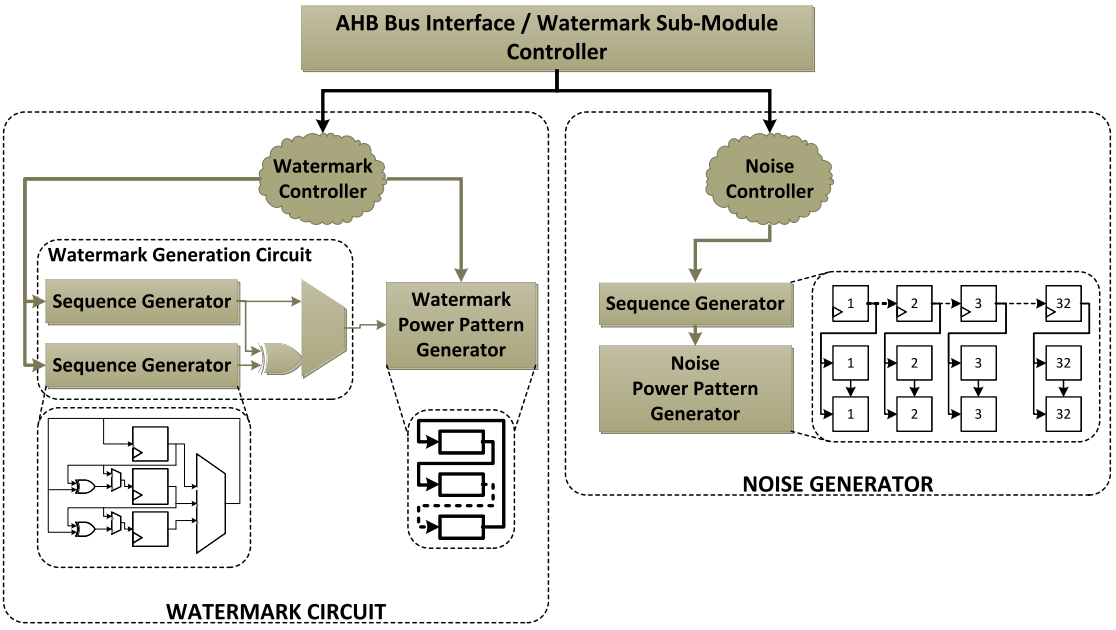


Figure 4.15: Schematic diagram of the watermark module embedded in test chips.

directly connected to the bus without any modifications. The hardened watermark module integrated in chip I is shown in Figure 4.13. The watermark circuit is highlighted in the top right corner and implements the watermark controller (yellow), WGC (red) and WPPG (green). As can be seen, the watermark circuit accounts for an approximately half of the area of the entire module. The configuration registers to control the watermark circuit are mapped to the SoC RAM memory. To accommodate the possibility of generating various watermark sequences, the watermark circuit contains two sequence generators, which can be configured as either 32-bit LFSR or a simple 32-bit circular shift registers. The WPPG architecture contains 1,024 registers, divided into 32 words. Each 32-bit word can be configured separately increasing or decreasing the strength of the generated watermark power pattern. Upon watermark sequence bit '1', all words are rotated. To generate the maximum switching power, words must be configured in an alternating fashion such as given by:

$$
\begin{aligned}
1^{st}\ word: &\quad FFFFFFFF \quad (hexadecimal)\\
2^{nd}\ word: &\quad 00000000\\
&\ .\\
&\ .\\
&\ .\\
31^{st}\ word: &\quad FFFFFFFF\\
32^{nd}\ word: &\quad 00000000
\end{aligned}
$$

For an in-depth explanation about the watermark architecture please refer to Appendix

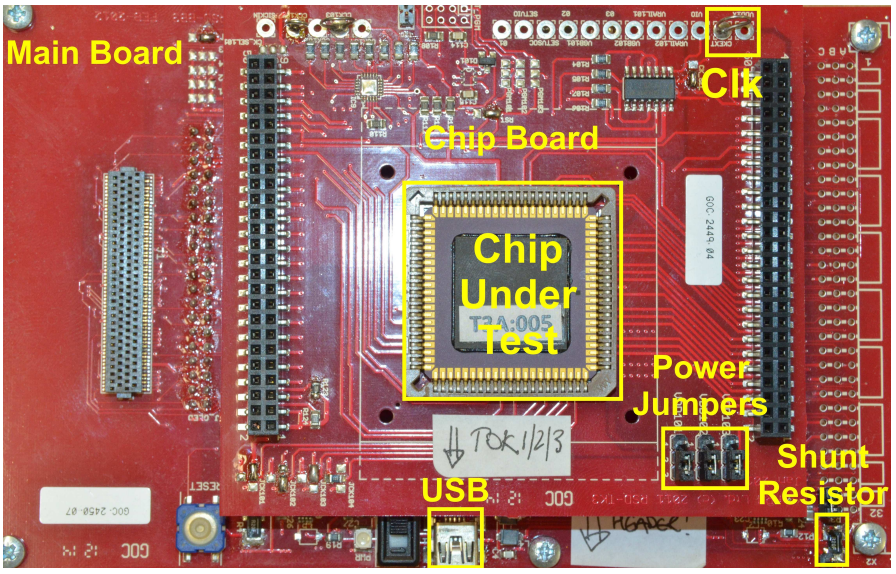
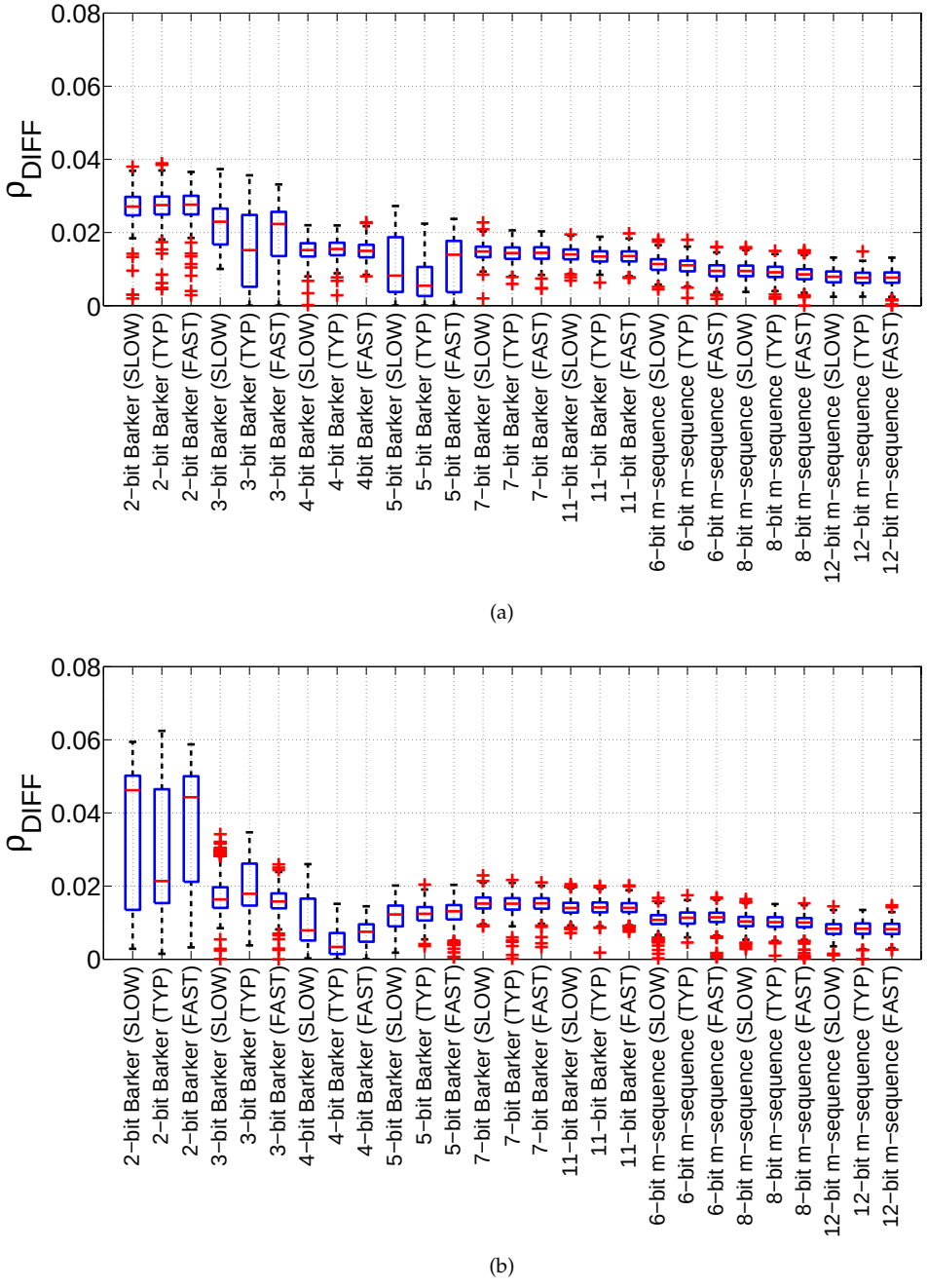


Figure 4.16: Silicon chips test board.

B.1. Furthermore, the watermark module implements a noise generator (Figure 4.15), shown in Figure 4.13. The architecture is very similar to the watermark circuit and contains a large power pattern generator (green) and a random number generator (red). The amplitude of the generated noise can be controlled in the similar manner as the WPPG circuit. Therefore, the noise strength varies with the pre-initialized data. The noise generator was used in the preliminary experiments to confirm its functionality and detection capabilities at various SNR. However, it was not used in further experiments due to unnoticeable differences in correlation results.

4.5.3 Experimental Setup

The test board is shown in Figure 4.16. All power domains were connected using the power jumpers and the total current consumed by the chip was measured, using the shunt $270m\Omega$ resistor. The operating frequency of both chips was $10MHz$. The current signal was measured as in Section 4.4.2. The aim of the experiment was the detection of the watermark while running the Dhrystone benchmark. The Dhrystone benchmark was executed on ARM® Cortex®-M0 on the SoC, on both chips. Although, on chip II Cortex®-A5 did not execute any program both cores, along with the on-chip bus were active, which accounted for a significant portion of background noise in the system. The experimental process (Section 4.4.2.1) was repeated on both test chips. The number of test repetitions was increased to 5, to test the susceptibility of watermark sequences to run-to-run noise variations. Additionally, 30 chips were characterized and 3 corners were chosen: fast, slow, and typical. The impact of PV, which occurs in the foundry during the chip fabrication was investigated. It should be noted that the measured



(a)

(b)

Figure 4.17: Box plots of correlation coefficient difference, ρ_{DIFF}, representing the influence of run-to-run and and process variation on watermark sequence correlation results in test chips: (a) chip I, (b) chip II.

current consumption included the noise of the system caused by the SoC and RAM on both chips, and the clock tree of the dual core Cortex®-A5 on chip II.

Table 4.4: Area and Power Reduction in ASIC

Watermark Sequence	Number of Registers in WPPG	Area Reduction	65nm		
			P_{DYN} Reduction	P_{STATIC} Reduction	P_{TOTAL} Reduction
7-bit Barker code	512	74.9%	73.2%	74.8%	73.3%
11-bit Barker code	512	74.8%	75.3%	74.8%	75.2%
12-bit m-sequence	2048	-	-	-	-
7-bit Barker code	256	74.8%	74.5%	75.7%	74.5%
11-bit Barker code	256	74.5%	75.5%	73.9%	75.5%
12-bit m-sequence	1024	-	-	-	-
7-bit Barker code	128	74.7%	75.8%	75.7%	75.8%
11-bit Barker code	128	74.3%	77.7%	75.3%	77.6%
12-bit m-sequence	512	-	-	-	-
7-bit Barker code	64	74.3%	76%	75.8%	75.9%
11-bit Barker code	64	73.4%	77.4%	75%	77.3%
12-bit m-sequence	256	-	-	-	-
7-bit Barker code	32	73.4%	72.7%	73.6%	72.7%
11-bit Barker code	32	71.7%	73.1%	71.8%	73.1%
12-bit m-sequence	128	-	-	-	-

4.5.4 Experimental Results

Results of ρ_{DIFF} obtained from both test chips are shown in Figure 4.17. First, consider the impact of run-to-run variations, when the test is repeated many times and the chip is re-configured between consecutive tests. This is shown by the size of the boxes in Figure 4.17. If the area of a box is large, the variance of results is high. Otherwise, results are consistent and a watermark sequence is deterministic. Results confirm the findings from the FPGA platform (Section 4.4.2). Watermark sequences with short periods cause much higher variance in results than longer period sequences. Next, consider the impact of PV on ρ_{DIFF}. It should be noted, that the size of box plots representing the variance of results does not differ much for most of the sequences. However, medians differ considerably for short period sequences on both test chips. Therefore, short period sequences are susceptible to PV, as shown on both test chips.

Experimental results demonstrate that short period sequences are not suitable for embedded power watermarking, due to high variance of results and strong sensitivity to PV. Therefore, results are non-deterministic and the expected detection performance cannot be estimated.

4.6 Area and Power Overheads

Minimization of area and power overheads is one of the major factors of all power watermarks implemented on embedded processors. In Section 4.3.3, various watermark sequences were simulated and it was shown that shorter sequences produce higher null

hypothesis rejection ratio than longer sequences, for the same noise power. The theory in Section 4.2 and simulation results of Section 4.3 have been validated on FPGA and test chips. Experimental results from the FPGA in Section 4.4, demonstrated that shorter period watermark sequences, such as 7 and 11 bits Barker codes, achieve the null hypothesis rejection ratio close to 95%, when the number of SRL16 blocks for WPPG is 8. To achieve the similar detectability with the 12-bit m-sequence, 32 SRL16 blocks must be used. Therefore, shorter Barker codes enable area overhead reduction of approximately 75%, by reducing the number of WPPG registers. To estimate the power reduction, the watermark circuits were synthesized using $65nm$[3] technology library. The fully placed and routed watermark circuit netlist, embedded in chip I, was simulated using Synopsys VCS, and a value change dump (VCD) file was created from the switching activity of the circuit. The estimate of the power consumption was obtained with Synopsys Primetime-PX, using the VCD file obtained from simulations. Results are shown in Table 4.4. The size of the WPPG circuit was varied, while keeping the 75% ratio between sequences. As the size of the WPPG is reduced, the 7-bit Barker code enables greater area and static power minimization, when compared to the 11-bit Barker code. The 7-bit Barker code requires 7 registers, while 11-bit Barker code requires 11 registers. However, since the *activity factor*, α, of the 11-bit Barker code is lower by 12% (Table 4.2), it consumes less total power for all WPPG sizes. Furthermore, as can be seen the total power reduction of at least 73% is achieved when using short watermark sequences, such as 7 or 11 bits Barker codes, instead of longer m-sequences, due to lower implementation requirements of the WPPG circuit. The reason for this is Equation (4.11) and Equation (4.13) shown in Section 4.2.

4.7 IP Attacks and Robustness

The watermarks discussed in previous sections transmit a single bit of information, to determine the presence of an IP. The watermark implementation followed [8, 58, 59], to establish the influence of sequence parameters on hardware implementation costs and detection performance. However, as the watermark can only be regarded as found or not, the IP candidates must be short listed for more thorough investigation. Therefore, the digital signature, such as author of a core, serial number or license agreement is not conveyed and anyone can claim an ownership, once he detects a watermark in a system [108]. In this section, the security of watermarks against various types of third party IP attacks is analyzed. The security, commonly known as the robustness, is determined by the attacks a watermark is able to withstand and the effort of an attacker. As it is difficult to quantify the robustness of side-channel watermarks [108], this section discusses it in relative terms [8]. In the classical cryptographic scenario an attacker aims to retrieve a secret key. If such occurs, the security of a system is breached and allows

[3]TSMC $65nm$ low leakage technology library.

an attacker to extract sensitive information. In case of IP watermarks, the system's security is not the aim of an attack, but the legal rights to an IP. This section discusses attacks against watermarks and focuses on the most prominent approaches, such as tampering, finding ghosts and forging. These are illustrated using the commonly used "Alice and Bob" scenario, often used in Cryptography, where "Alice" and "Bob" denote two individuals at either end of a communications channel, with cryptographic techniques applied to ensure their conversation is secret.

4.7.1 Tampering

Bob (attacker) can tamper with Alice's (IP supplier) solution, by removing Alice's signature (watermark) and adding own signature. Due to the transparent nature of the RTL description and unprotected design files provided to the SoC integrators, Bob has virtually unlimited access to a design. Therefore, it is not possible to prevent Bob from adding own watermark. However, it is crucial that Alice's watermark circuit is hidden, such that Bob cannot easily find it. In Section 4.6, the area overhead was reduced with short sequences. In such case, the watermark circuit is harder to find and remove, when compared to the current m-sequence scenario in [8]. Nevertheless, short sequences reduce the number of combinations and brute force attacks become feasible. Hence, the robustness against finding ghosts and forging attacks is impaired.

4.7.2 Finding Ghosts and Forging

Bob can attempt to find a ghost signature, such as specific power pattern, and claim that an IP contains his own watermark. Furthermore, Bob can forge Alice's implementation and watermark other solutions, which do not belong to Alice. In such case, Bob demonstrates that Alice's signature is not genuine since it can be found in another IP.

The robustness of sequences against such attacks is limited to their intrinsic characteristics. In case of commonly used m-sequence, the robustness increases with the number of registers used for WGC. For example, 32-bit m-sequence is more robust than 12-bit m-sequence, since it contains more frequency components (Figure 4.4(c)). Therefore, it increases the number of watermark combinations. Nevertheless, based on Figure 4.12, its detection performance must be complemented by increasing the number of WPPG registers. However, this reduces the watermark robustness against tampering attacks and the tradeoff occurs. The short period sequences, such as Barker codes enable the significant hardware implementation costs reduction, for the same level of detection performance as m-sequences, but are not as robust against finding ghosts and forging attacks. The use of short sequences allows attackers to introduce own watermarks (signatures), being difficult to determine which of the two watermarks is the authentic

one. As shown in Figure 4.4(b) and Figure 4.4(c), the amount of information transmitted in short sequences, such as Barker codes, is greatly reduced when compared to m-sequences. Therefore, the number of combinations of signatures based on the use of m-sequences is significantly larger, and it is much harder for an attacker to find ghost watermarks and demonstrate a stronger evidence or forge author's signature.

To overcome the limitations of short sequences, such as Barker codes, the private/public key encryption and the cryptographic hash functions, such as *MD*5 [109], can be used as in [19, 26, 35, 36, 49, 110]. However, as the encryption and the cryptographic hash functions are used, the encoded signatures vary with conveyed messages. Hence, the power pattern parameters, such as α and θ_{MAX}, change along with the implementation costs, to provide a high detection performance, Figure 4.7. To ensure the most cost-efficient parameters are utilized, the encoded signature can be generated as in [108]. In Figure 4.18(a), the implementation algorithm is shown. The digital signature ("Cortex®-M0") is encrypted with a private key, known only to the IP vendor. To reduce the length of the output bitstream, the encrypted message is later encoded using the cryptographic hash function (MD5). Furthermore, the hash encrypted bit sequence is used to modulate the cost-efficient sequence. In Figure 4.18(a), the 7-bit Barker code is used for illustration purposes. To generate bit '1', a full period of a 7-bit Barker code is used. To generate bit '0', the inverse of a sequence is used. The inverted sequence demonstrates different α and θ_{MAX} parameters. However, the parameters complement each other (Figure 4.2(b)) and similar ρ_{PEAK} and ρ_{DIFF} results are expected, when compared with the non-inverted sequence. In such way, the highly robust digital signature is generated. The detection algorithm is shown in Figure 4.18(b). The device power signal (trace) is measured with an oscilloscope. The power matrix is created by dividing a power trace, such that each signature bit corresponds to a specific trace. The Correlation Power Analysis is applied to each trace separately and the correlation spectra, ρ_{PEAK} and ρ_{DIFF} are found. To demonstrate the use of such algorithm, the digital signature of Figure 4.18(a) was simulated. The normally distributed noise of $32dB$ was introduced, as in Section 4.2. The size of the obtained power matrix was $128\ x\ 300,000$ clock cycles. When a watermark model for a particular signature bit is correct, a high positive correlation peak can be noticed, as shown in Figure 4.18(c). Otherwise, when a model is not correct and represents the inverted sequence, a high negative correlation peak is seen, as shown in Figure 4.18(d). Furthermore, if a model of another sequence was used or the data was not properly arranged, the correlation value would be close to 0. Finally, ρ_{DIFF} corresponding to all signature bits are plotted in Figure 4.18(e). In the case that ρ_{DIFF} have similar positive values, the correct hash encoded sequence is considered as found, then it can be further decoded and decrypted using the public key. In Figure 4.18(f), an error bit was introduced at the 4^{th} bit of a hashed sequence. As can be seen, the negative ρ_{DIFF} peak occurs and the error bit is detected. This means that the detected signature does not match the expected signature and the IP differs from the expected IP.

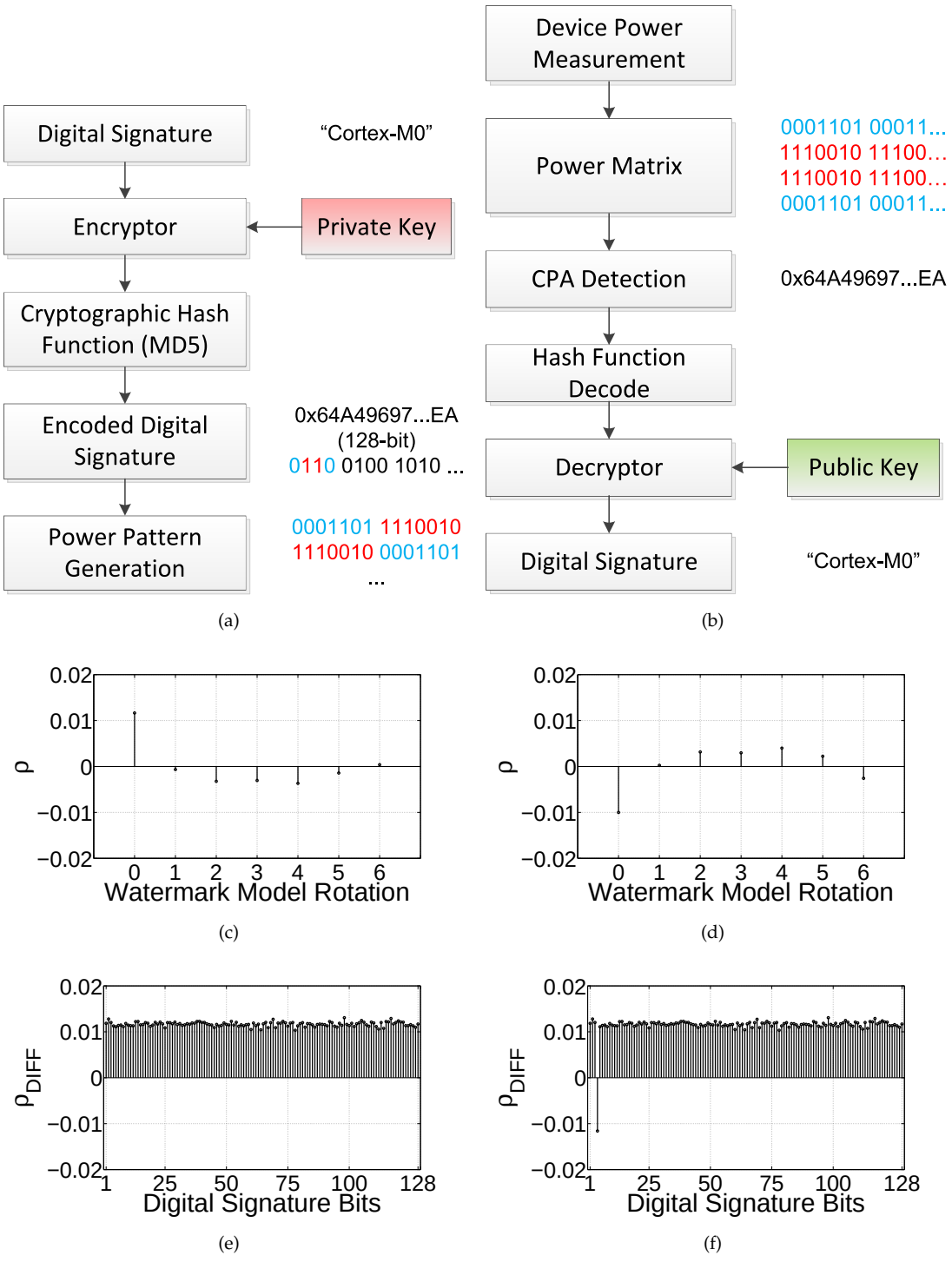


Figure 4.18: Implementation (a) and detection (b) diagrams of secure digital signature. Detection of a correct (c) and an incorrect (d) signature bit. Correct (e) and an incorrect (f) detection of a digital signature.

The use of the private/public key and the cryptographic hash functions ensures that the embedded power signature is highly robust. Although the attacker may not know the generation scheme of the digital signature, due to the limited number of signature combinations when short sequences, such as Barker codes are considered, it is feasible

for an attacker to record all power data and use a brute force attack, to reverse engineer the hash encrypted message. Nevertheless, if this occurs, the IP supplier's private key remains uncompromised [19], and it can still be used to prove the IP infringement. Additionally, if an attacker obtains the RTL a digital simulation of the design can be performed to check if any registers follow a Barker code. Since, the Barker code is public and there are only a few codes that can be used, this approach is also feasible. However, as all Barker codes are short and require only few clock cycles for the generation, the number of false alarms caused by other registers switching in the similar pattern increases with the system size. Moreover, the attacker would have to inspect most of such occurrences which may become time consuming for large designs. This is certainly the disadvantage of using Barker codes with the watermark generation approach as in [8], where the watermark circuit is active at all times. This issue can be addressed by reducing the frequency of the watermark circuit active time. For example, the watermark can be triggered with a specific system instruction, to increase the attacker's effort and computational time of the simulation. This is addressed in Chapter 6.

The proposed methodology, Figure 4.18, is however impractical in case of the longer m-sequences, due to the modulation of the watermark sequence with the hash encoded bit sequence. Nevertheless, the proposed approach requires an additional circuitry to implement the key encrypted and hash reduced Barker code (Figure 4.18). In a typical implementation, an extra 128-bit shift register would be required to hold the hash value and a state machine would have to be implemented to achieve the desired modulation. In an FPGA, such a shift register requires 8 LUTs, configured as 16-bit shift registers (SRL16). Although the basic state machine with only few states would be sufficient, the final hardware implementation would approximate the m-sequence approach (Table 4.3). Furthermore, an ASIC implementation would require the entire 128 registers to be implemented.

In this section, the robustness of watermark sequences against third party IP attacks was discussed. It was shown that m-sequences are robust against forging attacks but require bigger area to implement significant size WPPG circuit. Shorter sequences, such as Barker codes offer a reduced area and power overheads but are not as robust as longer m-sequences. The robustness of shorter sequences can be improved with the secure approach demonstrated in Figure 4.18 but the area overhead gains vanish. The tradeoff between longer m-sequences and shorter sequences, such as Barker codes, occurs and the watermark implementation must be reconsidered for various types and sizes of systems.

4.8 Application Specific Watermark Implementation

In small processors, such as microcontrollers (e.g. ARM® Cortex®-M0), the area overhead of the secure short sequence (Figure 4.18) implementation may be excessive. Since, a small WPPG is sufficient to generate a strong enough watermark power consumption, the m-sequence approach [8] may be a better solution. In bigger processors, such as application processors (i.e. ARM® Cortex®-A9), the WGC circuit has negligible impact and the WPPG size is the main factor. Since the WPPG size increases relatively linearly with the system size, the area overhead of the WPPG circuit for m-sequence would certainly be larger than the area overhead of the secure short sequence implementation. Therefore, the use of encoded Barker codes is expected to be more suitable, since it allows both area and power overhead minimization through a reduction of the WPPG circuit implementation.

Furthermore, in embedded systems the area and power overheads are often prioritized and it is not viable to generate the watermark power signal at all times. In such systems, the watermark is required to be active non-deterministically and for a short period of time. Short sequences, such as Barker codes, offer an ideal solution to such approach. Since an attacker must know when a watermark sequence is active and only for a very short period, finding activation time without a full knowledge of a system architecture is impossible. Moreover, if an attacker obtains a power signal without any architectural knowledge, the watermark signal may be too weak to be found, due to incorrect assumptions of the implemented architecture. Additionally, an erroneous correlation peaks may be generated, when a watermark is not present. Furthermore, to detect a watermark, multiple acquisitions of a power signal must be obtained and a long trace must be created, where a watermark signal is present continuously. When an IP owner tries to extract the embedded watermark pattern, it uses a special trigger to combine multiple power acquisitions into a single trace, where a watermark pattern is continuous. Such trigger is however not known to an attacker and will significantly increase the effort required for a successful attack. Additionally, short sequences can be implemented as in Chapter 5, to significantly reduce the area and power overheads. The visibility of an overridden clock enable signal due to watermark circuit can be kept to minimum since a simple XOR gate would ensure the clock gate is modulated according to a watermark sequence. This also ensures that if an attacker embeds his own ”always-ON” watermark they may violate the original area and power specification, which is easily detectable. Furthermore, the watermark embedded by an attacker can be of much lower amplitude, since they would use the WPPG circuit to achieve the desired watermark power consumption. The IP owner instead would use the original processor to emulate the WPPG circuit. This minimizes the occurrence of error bits with implementation in Figure 4.18. If an attacker wishes to understand the watermark implementation they would need to re-simulate the entire RTL to understand the watermark activation scheme, which is not a trivial task.

4.9 Summary

The influence of watermark sequences' intrinsic characteristics on detection performance, area and power overheads has not been addressed in the literature. In this chapter, two types of sequences have been analyzed (Section 4.1). First, sequences generated using the LFSR, as commonly found in the literature [8, 58, 59, 74]. Second, sequences generated with a simple circular shift register, also known as Barker codes. The in-depth analysis of the Pearson correlation coefficient, which is the fundamental of the Correlation Power Analysis technique for the detection of deeply embedded watermark power signals, has been been presented in Section 4.2. The intrinsic parameters of sequences used for watermarking have been demonstrated and their impact on detection performance was analyzed. To estimate the detection performance the Null Hypothesis Significance Test was applied to the CPA results. The expected behaviour was drawn with the support of simulations and it was shown that despite higher correlation peaks, longer sequences such as m-sequences generate other spurious correlation values, which may cause uncertain detection results. Therefore, shorter sequences such as Barker codes were demonstrated to produce much clearer spread spectra, with significant correlation peaks and much lower values of spurious correlations. The validation of simulation results was provided on an embedded Cortex®-M0 microprocessor core, executing the Dhrystone benchmark. Watermark detection performance of various binary sequences was investigated on FPGA and silicon test chips. Experimental results obtained from FPGA matched the simulation results and confirmed that shorter sequences, such as Barker codes achieve much better detection results when compared with longer sequences, such as m-sequence. Silicon results have demonstrated that very short sequences, such as 2, 3, 4 and 5 bits Barker codes are sensitive to both process variation and run-to-run variations, while longer sequences such as 7 and 11 bits Barker codes and m-sequences are immune to either of such variations.

Furthermore, with the aid of the power estimation, based on the fully placed and routed ASIC implementation, the area and power overheads of various watermark architectures were compared in Section 4.6. It was shown that a significant area and power reduction is achievable with shorter sequences, such as Barker codes, due to much lower WPPG requirements, to achieve a comparable detection results. However, in Section 4.7, the analysis of most prominent third party IP attacks has shown that short sequences are far more susceptible to attacks than longer sequences, due to fewer number of possible combinations. This makes shorter sequences vulnerable to brute force attacks. The solution to enhance the robustness of shorter sequences was proposed in a way of encryption. Nevertheless, this has caused the architectural gains to vanish and the short sequence implementation to result in a similar or larger circuit, when compared with longer sequences. Such tradeoff was analyzed in Section 4.8, and it was concluded that longer m-sequences are more suitable in smaller circuits, such

as microcontrollers, whereas shorter sequences are an ideal solution for much bigger devices, such as microprocessors.

4.10 Concluding Remarks

In this chapter, watermarking sequences have been characterized based on their intrinsic parameters, and have been compared in terms of detection performance, area and power overheads. It was shown how sequence's parameters, such as length, *activity*, α, and the *overlapping factor*, θ, influence the correlation coefficients in a spread spectrum graph. Using a new theoretical definition, the relationship between the watermark sequence parameters and detection performance has been illustrated and validated with simulations and experimental results of FPGA and ASIC designs of embedded processors. It has been shown that the tradeoffs occur between shorter sequences, such as Barker codes, and longer sequences, such as m-sequence, in terms of hardware implementation costs and robustness against third party attacks. The tradeoffs have been analyzed and it has been concluded that for smaller systems the commonly used m-sequence approach is a better solution due to its robustness against third party attacks. However, for bigger systems shorter sequences achieve better hardware implementation costs without sacrificing the robustness performance (especially when used in conjunction with other techniques such as clock modulation, discussed in the next chapter).

Chapter 5

Clock-Modulation Based Watermark Power Pattern Generation

The current state-of-the-art watermark circuits [8] require a significant size WPPG load circuit, to be detected with the current state-of-the-art Correlation Power Analysis [8,9] detection technique. However, embedded processors are increasingly constrained in terms of area overhead. Therefore, a technique is required which allows the minimization of the WPPG load circuit, without a corresponding reduction in detection performance. In this chapter[1], such a technique is proposed and enables complete removal of the WPPG circuit through the modulation of the clock signal in sequential logic. Therefore, a significant area overhead reduction is achieved.

The rest of the chapter is organized as follows. The fundamentals of the proposed technique and the watermark circuit implementation are provided in Section 5.1. The validation of watermark implemented on silicon test chips is given in Section 5.2. The reduction of area overhead through the application of the proposed technique is analyzed in Section 5.3 and the improved robustness is discussed in Section 5.4. The chapter is summarized in Section 5.5 and the final conclusions are given in Section 5.6.

[1]The contents of this chapter have been published in [111], as "Clock-Modulation Based Watermark for Protection of Embedded Processors" by Kufel *et al* in *Design, Automation Test in Europe (DATE) Conference, 2014*.

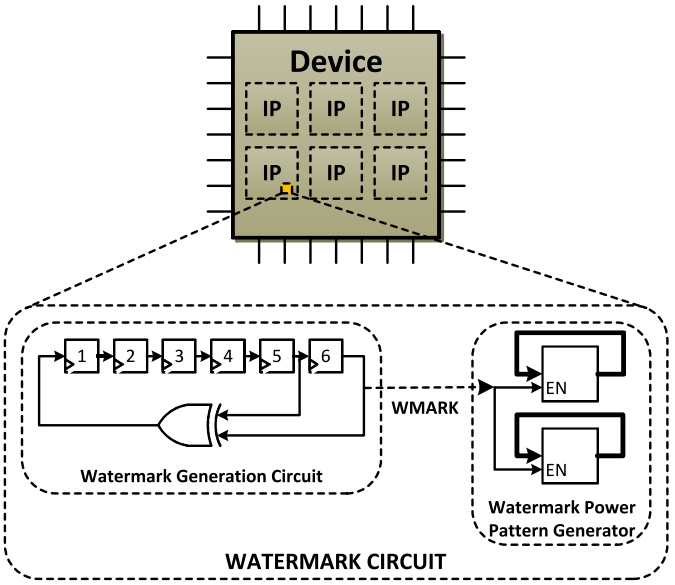


Figure 5.1: Architecture of the current state-of-the-art power watermark circuit.

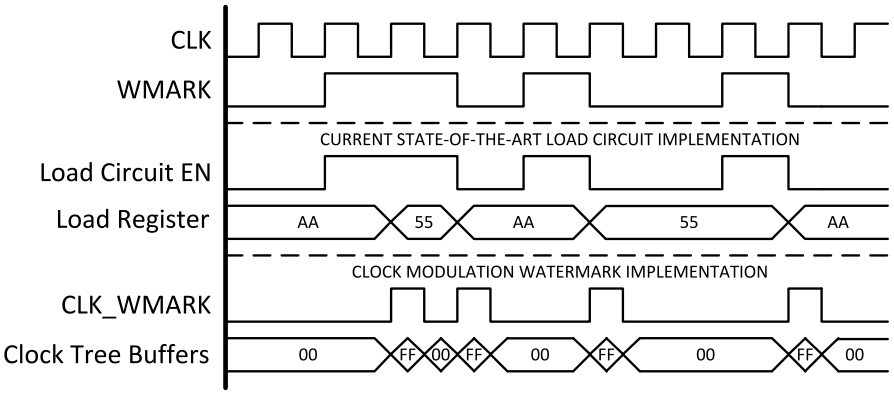


Figure 5.2: Timing diagram of current state-of-the-art (middle) and clock modulation (bottom) watermark architectures.

5.1 Clock Modulation Watermarking Technique

The current state-of-the-art power watermark circuit [8, 58] implements WGC and WPPG circuits, Figure 5.1. The WGC generates the watermark sequence, *WMARK*, which controls the shift enable input of the WPPG. In Figure 5.2, the timing diagram of the watermark circuit is shown. The load register consists of an 8-bit shift register initialized with '1010...' pattern ($0xAA$), to maximize dynamic power consumption when *WMARK* is '1'. It can be seen, that when *WMARK* is '1' the shift enable signal is '1'. The dynamic power is consumed, due to shift operation during which all registers change their states. Analytical techniques such as Correlation Power Analysis (Chapter 3, Section 3.3), have been reported to detect deeply embedded watermark signals [8]. However, the area overhead of the watermark circuit in Figure 5.1 is significant, when

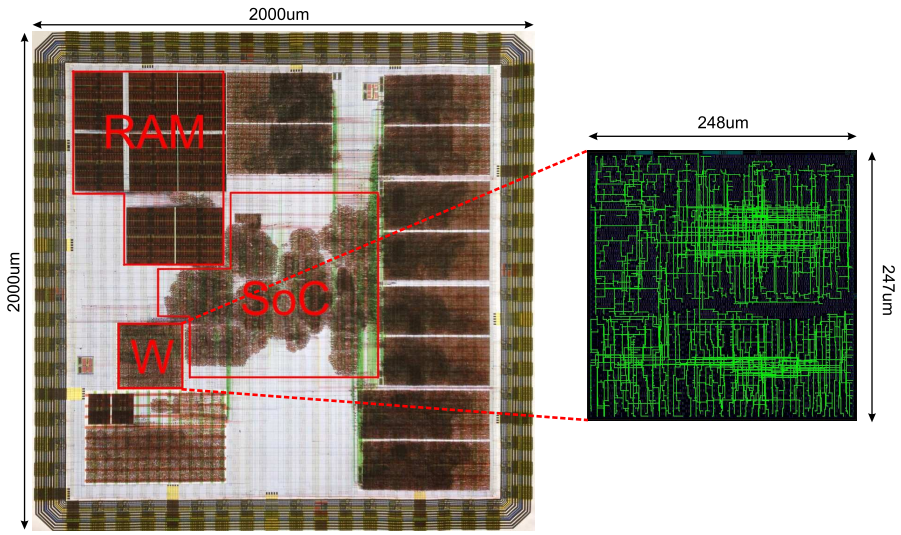


Figure 5.3: Clock tree of hard macro watermark module integrated on chip I.

compared to area overhead as low as 0% with techniques [34,35,39] discussed in Chapter 2, Section 2.4.3.

In Chapter 4, Section 4.4.2, results of the Null Hypothesis Significance Test have shown that watermark detection of approximately 100% can be achieved with shorter watermarking sequences, such as 7 and 11 bit Barker codes. At the same time, the area overhead reduction of 75% is possible in comparison to the commonly used m-sequence, while maintaining the same detection performance. Nevertheless, the WPPG already occupies a significant area in a system. To reduce area overhead, a watermark implementation technique is proposed which controls a watermarked IP sub-module with the watermark modulated clock signal, as shown in Figure 5.6. The size of the watermark circuit is constant, independent on the system size and therefore the area overhead is negligible. The technique is expected to scale with the system size and achieve similar detection performance, while maintaining a low area overhead. Furthermore, the technique produces a watermark circuit with an improved robustness against third party IP attacks, since the watermark logic is embedded in such way that it becomes an integral part of a processor. Therefore, tampering with an IP, such as removal, will cause erroneous functionality of a system. To fully understand the applicability of the technique and its advantages, it is necessary to consider the dynamic power reduction technique, known as clock gating.

5.1.1 Clock Gating

In digital circuits, the distribution of a clock signal on a chip, also known as a clock tree [113], contributes a significant amount of dynamic power consumption. In Figure 5.3, the clock tree of watermark module integrated on chip I (Chapter 4, Section 4.5) is shown. As can be seen, the clock signal is routed throughout the watermark module.

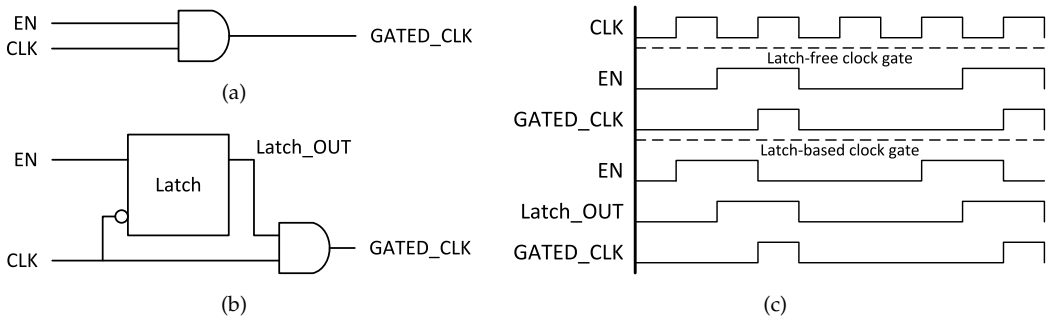


Figure 5.4: Architecture of the (a) latch-free and (b) latch-based clock gates and (c) timing diagrams [112].

In [114], authors report that typically up to 50% of the total dynamic power is consumed by the system's clock signal. Since most of the processor design is sequential, with a large number of registers being clocked, the fan-out[2] of the clock tree causes high dynamic power consumption. However, not all registers must be switched every clock cycle and their states are retained for many cycles before the next update. During the clock cycles where the data is only retained and does not change, an additional and redundant dynamic power is consumed, due to the clock tree switching signal. To reduce the dynamic power consumption, the technique known as clock gating was introduced to switch off the parts of the design when it did not need to be updated. This is achieved through the use of special clock gating logic cell added to the clock tree [115]. The schematic diagrams of a typical clock gating cells, such as latch-free and latch-based clock gates are shown in Figure 5.4(a) and Figure 5.4(b), respectively. The timing diagram of both architectures is depicted in Figure 5.4(c). The latch-free architecture uses a simple AND or OR gates, but all enable signals must be held constant from the active (rising) edge of the clock until the inactive (falling) edge of the clock [112]. Therefore, the system design must meet the setup and hold times to avoid truncating the generated clock pulse prematurely or generating multiple pulses where a single pulse is required. Otherwise metastability may occur, where the logic level of a signal is neither '0', nor '1', and erroneous execution is expected. The latch-based architecture avoids such restrictions and prevents any signal change during the active edge of the clock. This is achieved by locking the state of the enable signal during the falling edge of the clock. Therefore, the output from the latch, *Latch_OUT*, is retained until the next falling edge of the clock. Hence, the signal meets both setup and hold times. The clock gates are introduced by designers during the system and block design phases [116]. In [115], the dynamic power reduction of 33% and total power reduction of 15% are reported, when clock gating was used in a general purpose microprocessor. Furthermore in [117], it is shown that the dynamic power reduction of over 50% is achieved, when clock gating technique is implemented on FPGA.

[2]The fan-out of a logic gate describes the number of logic gates it can drive/connect to their inputs.

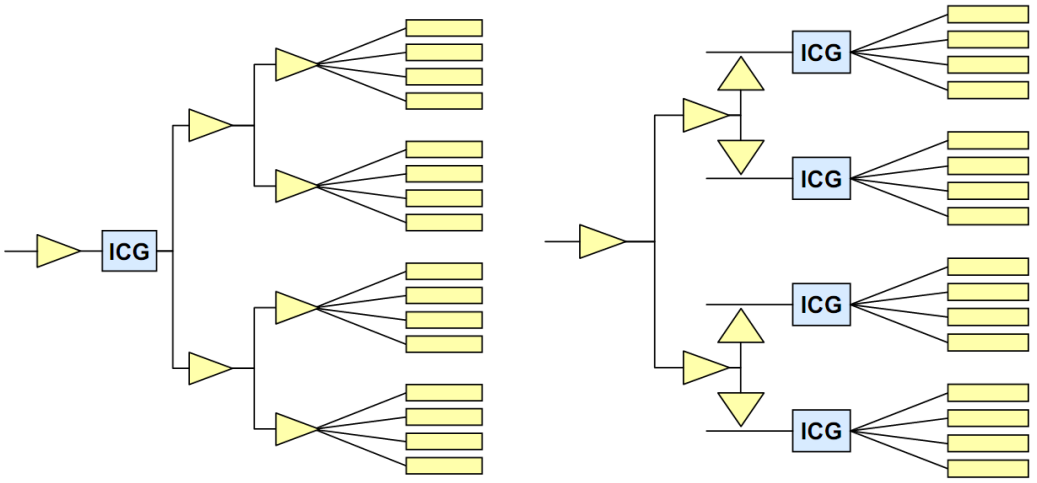


Figure 5.5: Clock tree connections for (a) high, and (b) low fan-out settings [118].

5.1.2 Clock Modulation Watermark Architecture

The clock gating technique is the mainstream of the design flow for dynamic power reduction in modern integrated circuits, such as embedded processors. The design is divided into sections with each section clock gated separately. Since, the clock network delivers the clock signal to most parts in a design (Figure 5.3), the number of interconnections is large and differs with the design. If a high fan-out is allowed, the lower level clock gates in the same section are not required [118], Figure 5.5(a). Therefore, the significant power reduction is achieved, but the enable signal is constrained. In case of a low fan-out, the additional clock gates cells (ICG) are embedded in the lower logic levels, Figure 5.5(b). The enable signal is easier to implement but the power savings are reduced. The clock tree distributes the clock signal throughout the chip to all sequential logic. However, the differences in clock net lengths introduce the variance in delay between connections. To improve the timing of the clock signal, known as the clock skew, the buffers are added [119, 120]. Nevertheless, such approach increases the capacitive load of the clock network and contributes a significant portion of dynamic power.

The current watermarking scheme [108] requires an additional circuit to generate such portion (or less) of power. Therefore, the clock tree can be utilized and configured to emulate the WPPG in a watermarked design. Nevertheless, to ensure a sound robustness of the proposed technique, the significant transparencies in the RTL description, such as interconnections between various circuits, must be kept at minimum. One such solution, which implements a watermark circuit in the top level clock gate provides two advantages. First, the number of interconnections is reduced, and second, the generated watermark power pattern is the strongest, since the modulated clock is propagated to the entire processor core. However, due to the modulation of the top level clock signal the synchronization issues with other IP blocks may occur. This could

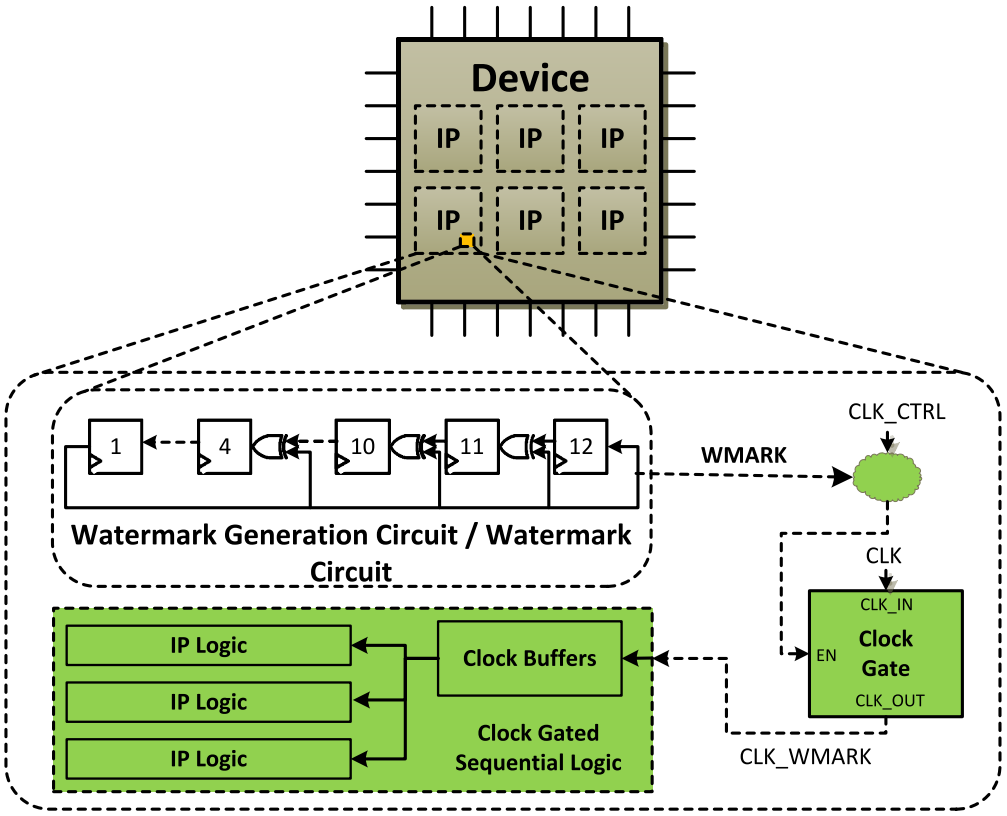


Figure 5.6: Architecture of the proposed clock modulation power watermark circuit.

be avoided by not modulating the blocks which interface with the rest of the system, such as bus controllers and I/O controllers. Nevertheless, the performance issues occur and the throughput of a watermarked processor is significantly reduced. Both issues are further addressed in Chapter 6.

The proposed clock modulation based technique is shown in Figure 5.6. The significant size WPPG circuit reported in previous publications [8,58] is thus removed. The architecture of the WGC is unmodified and generates a watermark sequence, *WMARK*. The original clock gate control signal, *CLK_CTRL*, and *WMARK* further control the enable signal of a clock gate of an IP block, as shown in Figure 5.6. Analogically, the original clock signal, *CLK*, to the IP block is modified and replaced with the modulated clock signal, *CLK_WMARK*. When *WMARK* is '1', the clock gate enable is '1' and *CLK* is propagated (*CLK_WMARK* = *CLK*). When *WMARK* is '0', *CLK* is stopped at the clock gate (*CLK_WMARK* = 0). In case the watermark circuit is active during processor execution, the entire IP block generates significant dynamic power in clock cycles when *WMARK* is '1'. However, this may require an additional synchronization between the watermark modulated and other IP blocks, to ensure data is not corrupted, or decrease the throughput. Moreover, the size of the IP module must be significant to generate a strong enough watermark power signal, due to the background noise produced by the rest of the system. In case the watermark circuit is active while the entire system

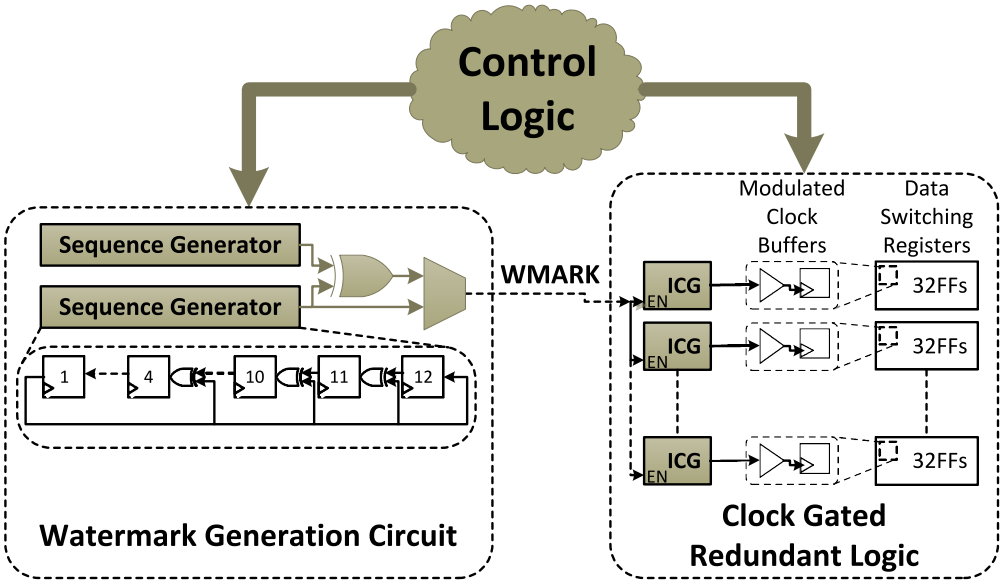


Figure 5.7: Schematic diagram of the clock modulated watermark circuit embedded in test chips.

is inactive, the watermark power is consumed entirely by clock tree buffers. As can be seen in Figure 5.2, the clock modulation technique produces higher switching activity in comparison to watermark architecture implementing a WPPG circuit of Figure 5.1. This means that clock buffers switch twice in a single clock cycle during the rising and falling edges of a clock signal. Therefore, the dynamic power consumed in a single register by clock tree buffers is higher than the dynamic power consumed by data switching in the same register, as shown in Section 5.3. In Section 5.2, experiments on silicon test chips are performed, to analyze if watermark power generated in such way produces high enough amplitude, to be detected with the Correlation Power Analysis.

5.2 Silicon Validation

To validate the technique discussed in Section 5.1, two ASIC designs (Chapter 4, Section 4.5), fabricated in 65*nm* low leakage CMOS technology, were tested.

5.2.1 Watermark Sub-Module Circuit Architecture

The architecture of the watermark module is the same on both chips, Figure 5.7. The WGC contains two sequence generators which can be configured as either 32-bit Linear Feedback Shift Registers (LFSR) or simple 32-bit circular shift registers. In experiments presented in this section, only a single sequence generator was used, configured as 12-bit LFSR and generated 12-bit maximum length sequence at *WMARK* output signal. The redundant logic circuit contains 1,024 registers, divided into 32 words. The clock

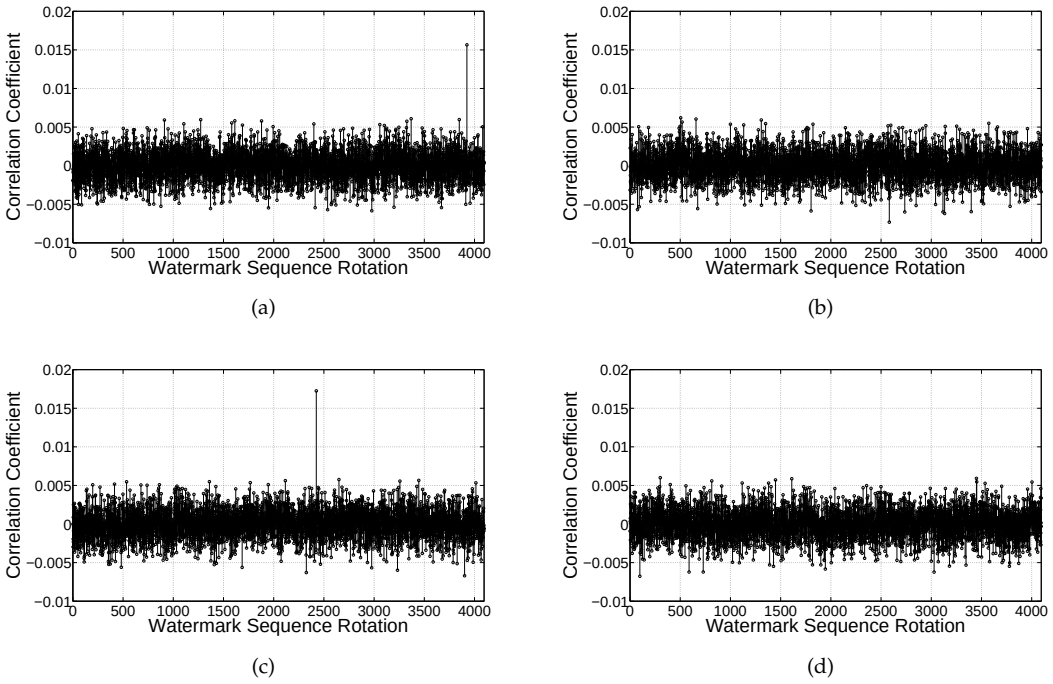


Figure 5.8: Spread spectra of correlation results from test chips. Chip I with active (a), and inactive (b), watermark circuit. Chip II with active (c), and inactive (d) watermark circuit.

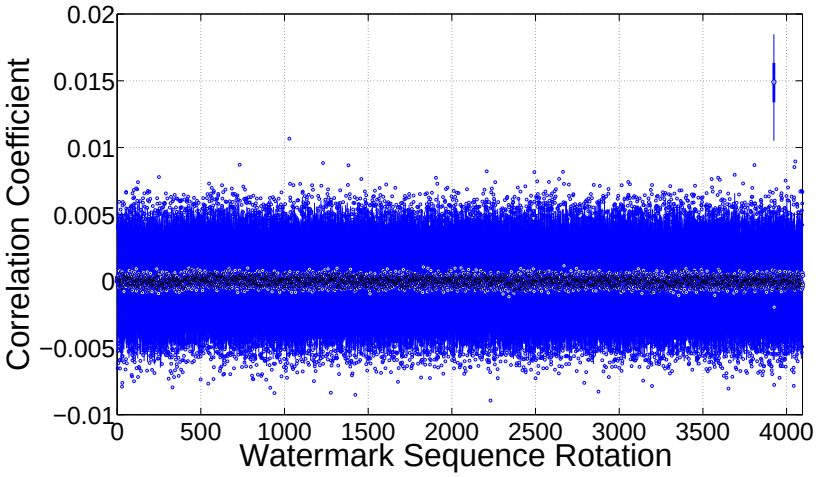
signal to each 32-bit word is clock gated using the clock gate cell (ICG). The clock enable signal of each clock gate cell is controlled by *WMARK*. The clock signal is propagated through all 32 clock gates, when *WMARK* is '1'. When *WMARK* is '0', the clock signal is stopped at ICGs and no dynamic power is consumed. All registers are pre-initialized to '0', hence no data switching occurs. As can be seen in Figure 5.7, there is a large number of clock buffers embedded in the WPPG registers. Such buffers consume a significant amount of dynamic power. Therefore, the clock modulation technique from Section 5.1, can be validated with such a watermark circuit configuration.

5.2.2 Methodology

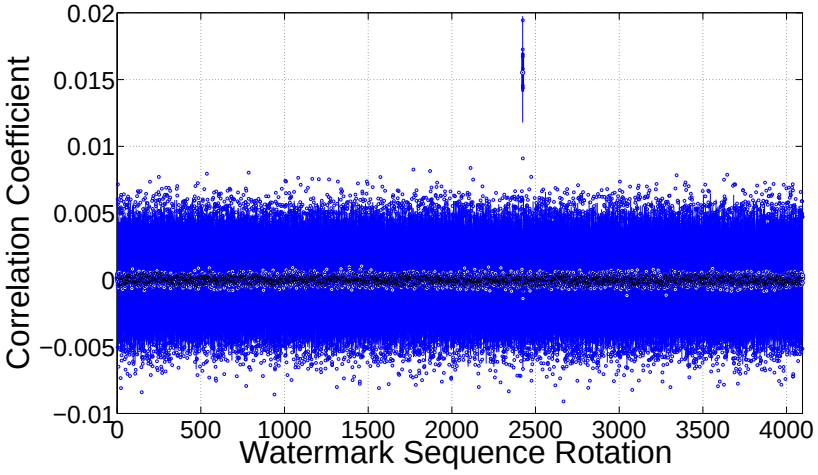
The experimental process discussed in Chapter 3, Section 3.3 and Chapter 4, Section 4.5, was repeated on both test chips. The Dhrystone benchmark was executed on ARM® Cortex®-M0 on the SoC, on both chips.

5.2.3 Experimental Results

The spectra of correlation results are shown in Figure 5.8. Since the period of the 12-bit maximum length sequence ($2^{12} - 1$) is shorter than $300k$ clock cycles, the watermark sequence was repeated multiple times within a vector, X. It can be seen, that the



(a)



(b)

Figure 5.9: Box plots of correlation coefficient results from chip I (a) and chip II (b), when the experiment was repeated 100 times.

correlation peak for chip I occurs at approximately the $3,800^{th}$ rotation of the watermark sequence (Figure 5.8(a)), and at approximately the $2,400^{th}$ rotation of the watermark sequence for chip II (Figure 5.8(c)). Since no other correlation peaks exist, the watermark can be regarded as detected. To confirm that correlation peaks were not the result of the correlated system noise, the watermark circuit was disabled and experiments were repeated on both chips. As can be seen in Figure 5.8(b) and Figure 5.8(d), no correlation peaks occurred when the watermark power pattern was not present. To investigate the repeatability of detection results, the experiments were performed 100 times on both chips. In Figure 5.9 correlation coefficients are shown in a form of a box plot[3]. It can be seen, that medians when X and Y were not in phase is close to 0. However, when both vectors were in phase medians are much higher and distinctive correlation peak

[3]The description of a box plot can be found in Chapter 4, Section 4.4.

Table 5.1: Power Consumption of Placed and Routed WPPG Circuit

WPPG Circuit Implementation	**Power Consumption**			Total Watermark Dynamic Power
	Dynamic	**Static**	**Total**	
Clock Buffers Modulation No Data Switching	1.51 mW	0.404uW	1.51mW	95.6%
Clock Buffers Modulation 256 Switching Registers	1.80 mW	0.407uW	1.80mW	96.8%
Clock Buffers Modulation 512 Switching Registers	2.09 mW	0.407uW	2.09mW	97.2%
Clock Buffers Modulation 1024 Switching Registers	2.66 mW	0.408uW	2.66mW	98%

can be distinguished. The variance of all results represented by the box in the figure accounts for 95% of all correlation coefficients, with extreme values shown as dots. As can be seen, the correlation coefficient peak is present in all experiments on both chips. Therefore, an embedded watermark was successfully detected in all repetitions. The redundant logic in the demonstrated experiments is a stand-alone circuit, however, in the end application a commercial IP sub-module can be reused with similar results, reducing the area overhead, as will be shown in Chapter 6.

5.3 Area Overhead Reduction

The area of the current state-of-the-art watermark circuit is largely occupied by the significant size watermark power pattern generator (WPPG) load circuit, Figure 5.1. In case of system scaling, the size of the watermark generation circuit (WGC) does not change, while the size of the WPPG varies and increases with the system size. This effect is caused by the Correlation Power Analysis detection technique, which requires substantial watermark power signal to detect an embedded watermark circuit. Various novel detection techniques have been demonstrated in recent publications [81, 88], which detect an embedded circuit of negligible size. However, similarly to many soft IP protection techniques [34–36, 38, 39], an access to watermarked design throughout an entire design flow, or access to post-fabricated design internals are necessary for successful detection. This is not possible for many IP suppliers. The proposed watermark clock modulation technique enables watermark implementation at RTL description level, with negligible area overhead, and allows post-fabrication watermark detection with the CPA detection technique. Furthermore, the size of the watermark circuit is the same for all systems, hence does not need scaling, since the watermark architecture only requires implementation of the negligible size WGC.

To determine the area overhead reduction of the proposed clock modulation technique to the current state-of-the-art watermark implementation demonstrated in Section 5.1,

Table 5.2: WPPG Circuit Implementation Costs

Detectable WPPG Circuit Dynamic Power Consumption P_{Load}	**Number of WPPG Circuit Registers** $N = P_{Load}/(1.126uW + 1.476uW)$	**Area Overhead Increase**
0.25 mW	96	89%
0.5 mW	192	94%
1 mW	384	97%
1.5 mW	576	98%
5 mW	1921	99%
10 mW	3843	100%

Figure 5.1, the power consumption of the fully placed and routed watermark circuit was estimated with Synopsys Primetime-PX, using 65nm low leakage CMOS technology. The power consumption of the clock modulated redundant WPPG circuit is shown in Table 5.1. In the top of the table, the WPPG is implemented as shown in Figure 5.7. Hence, the dynamic power consumption is caused entirely by clock buffers. This implementation was found to account for 95.6% of total watermark circuit dynamic power. The number of switching registers is further increased, until all 1,024 registers switch when *WMARK* is '1'. Therefore, the dynamic power consumption is caused by both data switching and clock buffers modulation. It can be seen, that the dynamic power consumed by clock buffers is higher than the dynamic power caused by data switching. On average, the dynamic power consumption of a single clock buffer is 1.476uW, and data switching in a single register is 1.126uW. In Table 5.2, the number of switching registers, N, required to implement the WPPG of Section 5.1, Figure 5.1, is shown for various system sizes. It is based on the required WPPG dynamic power consumption (in relative terms), to be easily detected with CPA technique. As can be seen, approximately 580 registers are required to implement the WPPG with the current state-of-the-art watermark architecture, to consume the same dynamic power as the clock gated redundant circuit in Section 5.2, Figure 5.7. With the proposed clock modulation technique, the dynamic power consumed by clock buffers can be obtained through the modulation of clock tree buffers of existing logic, Section 5.1.2, Figure 5.6. The WGC requires only 12 registers, hence an area overhead reduction of 98% can be achieved. Nevertheless, the area overhead reduction depends on the system size, as shown in Table 5.2. As can be seen, the area overhead reduction is less in smaller systems, but still significant when compared to the WPPG based watermark implementation. In bigger systems, the area overhead reduction is close to 100%. Moreover, watermark implementation can be system specific. Therefore, top level IP modules or lower level sub-modules can be modulated with the proposed technique and the power overhead of the watermark implementation can be tailored to the system.

5.4 Improved Robustness

One of the major cornerstones of all IP watermarking techniques is the robustness against third party removal attacks. It is performed with the aim of removing watermark circuit from the design. In the case of soft IP, a removal attack can be performed at the RTL description level, due to high visibility of the system [1]. Since the current state-of-the-art watermark circuit implements a significant WPPG load circuit, the removal is easily performed. Moreover, as the watermark is a stand-alone circuit, removal has no impact on the system performance.

The clock modulation technique proposed in this chapter significantly reduces area overhead of the watermark circuit, leading to an enhanced robustness to removal attacks. Furthermore, the WGC circuit can be embedded in various sub-modules and detection capabilities of an attacker are significantly reduced. As can be seen in Figure 5.7, the proposed watermark implementation does not produce a stand-alone circuit, and therefore the system's functionality is greatly impaired when watermark is removed.

5.5 Summary

The significant size of the WPPG circuit, which is the integral part of the current state-of-the-art watermark circuit [8] (Chapter 3, Section 3.2) is required to be detected with the CPA technique [8, 9]. In this chapter, the technique was proposed for the minimization of the watermark circuit, through removal of the entire WPPG circuit and modulation of the clock signal in sequential logic. It was shown that the clock tree network occupies a considerable amount of the circuit, Figure 5.3, and consumes up to 50% of the total dynamic power [114]. Hence, the generation of the watermark power pattern through the modulation of clock gates is a suitable technique for IP protection. If such sequential logic circuits are considered as a single sequential logic block, it can be regarded as the WPPG load circuit with zero-area overhead. The technique was validated on silicon tests chips (Chapter 4, Section 4.5.1), fabricated in 65*nm* low leakage CMOS technology. Although the architecture of the watermark implemented a separate stand-alone circuit, experimental setup and configuration of the watermark circuit represent the generic case of modulating the IP processor core, with the watermark sequence. Experimental results have confirmed the significant area overhead reduction with the proposed technique. Furthermore, the robustness against third party IP attacks has been improved. However, when it is implemented in an embedded processor, the performance is impaired, due to the '1' and '0' pattern of a watermark sequence. Moreover, synchronization issues may occur.

5.6 Concluding Remarks

A novel clock modulation watermark technique for embedded processors has been proposed in this chapter. The technique was validated with two ASIC designs. The significant area reduction has been demonstrated when compared to the current state-of-the-art watermark circuit architecture, reported in previous publications, with 98% area reduction obtained from experimental results. The proposed clock modulation technique has also achieved the significant increase in robustness against third party IP attacks, such as removal attacks. However, performance and synchronization issues occur. These issues are addressed in Chapter 6.

Chapter 6

Instruction Based Activation of Watermark Power Pattern

In Chapter 5, the technique for the emulation of the WPPG circuit through the modulation of clock gates in sequential logic was proposed. A significant area overhead reduction was achieved by complete removal of the WPPG load circuit, typically found in the literature [8]. To compensate such a considerable power consumption, the modulated sequential logic was considered as the zero-area overhead load circuit. Nevertheless, the proposed clock modulation technique assumes a continuous generation of a watermark power pattern and power overhead costs are significant. The technique proposed in this chapter activates a watermark circuit during specific processor instruction, to minimize the dynamic power consumption.

The rest of the chapter is organized as follows. In Section 6.1, a brief motivation for the work presented in this chapter is given. It is followed by the introduction of the proposed technique in Section 6.2. The technique is validated on two ASICs, and it is divided to two separate test cases investigated in Section 6.3 and Section 6.4, respectively. The hardware implementation costs are discussed in Section 6.5 and the robustness is analyzed in Section 6.6. The chapter is concluded in Section 6.7.

6.1 Motivation

The clock modulation technique demonstrated in Chapter 5 allows a significant area overhead reduction, when compared to the current state-of-the-art technique [8]. However, the dynamic power consumption is significant and synchronization between watermarked and non-watermarked circuits is necessary. Furthermore, the processor's performance is greatly impaired. Hence, a new technique is required to avoid such synchronization issues and minimize the power and performance overheads. In this chapter, technique of Chapter 5 is improved by activation of the watermark circuit during the specific processor instruction, known as Wait-for-Interrupt (WFI). This allows a significant reduction of the power overhead with negligible performance, and synchronization issues do not occur.

6.2 Proposed Watermark Circuit Instruction Based Activation Technique

Techniques for minimization of power consumption in embedded processors, such as clock gating and power gating are commonly used throughout the industry. The clock gating technique (refer to Chapter 5, Section 5.1.1) allows minimization of dynamic power consumption, by disabling the clock signal to the parts of the circuit which do not need to be updated [115]. However, the static power consumption (short circuit power[1]) is unchanged. To aid with further reduction of the static power consumption the power gating technique is applied, where the power to the processor is removed [121]. During the normal operating mode the processor executes the application and both dynamic and static power is consumed. However, as it is commonly known most applications require a processor to wait for a specific signal from other devices for extended periods. Therefore, until such a signal is received the device remains in the low power mode using power gating techniques. In a typical processor design flow both clock gating and power gating techniques are combined to reduce dynamic and static power. The commonly used instruction to enter the low power mode is known as Wait-for-Interrupt (WFI). Upon execution of such instruction a processor enters a low power mode [122]. However, before removing the power to the processor other active tasks must be executed to prevent the loss of data or erroneous operation, after the circuit has been powered up. This is commonly known as the pipeline flush.

To minimize the area overhead the watermark is implemented as in Figure 6.1. The WPPG circuit is removed as in Chapter 5. The watermark circuit is embedded in the architectural top level clock gate circuit and generates a watermark sequence (*WMARK*),

[1]For the explanation of the source of short circuit power consumption, see Chapter 3, Section 3.1.

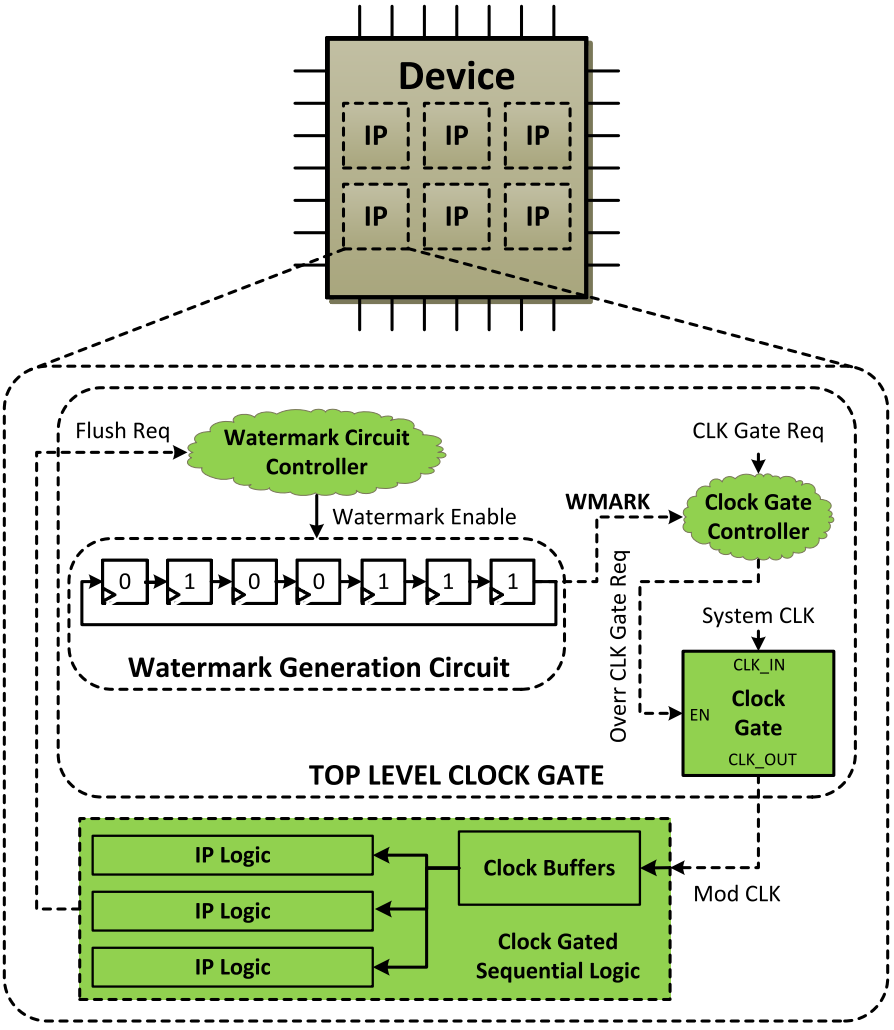


Figure 6.1: Architecture of the proposed technique for clock modulation based watermark power pattern generation with instruction based activation.

to control the enable signal of the processor clock gate. As can be seen, the original clock gate request signal (*CLK Gate Req*) is overridden and replaced with the *Overr CLK Gate Req* signal. The modulated clock signal, *Mod CLK*, drives the sequential parts of a processor. Therefore, it emulates the WPPG load circuit.

In this chapter, two watermark activation approaches are proposed, Figure 6.2. In the first approach (mode I), the watermark circuit is active during the processor pipeline flush operation. Upon the execution of the WFI instruction the pipeline flush request signal, *Flush Req*, is sent to multiple parts of a processor. This ensures that active instructions are executed before the processor is power gated. In Figure 6.1, such signal is connected to the watermark circuit controller and determines the activation of the watermark power pattern. Since the entire system, including top and lower level clock gates are active, the amplitude of the watermark power signal is the highest. However, the processor is active and generates a considerable background noise. Therefore, the signal-to-noise ratio (SNR) is expected to be much lower and may cause the watermark

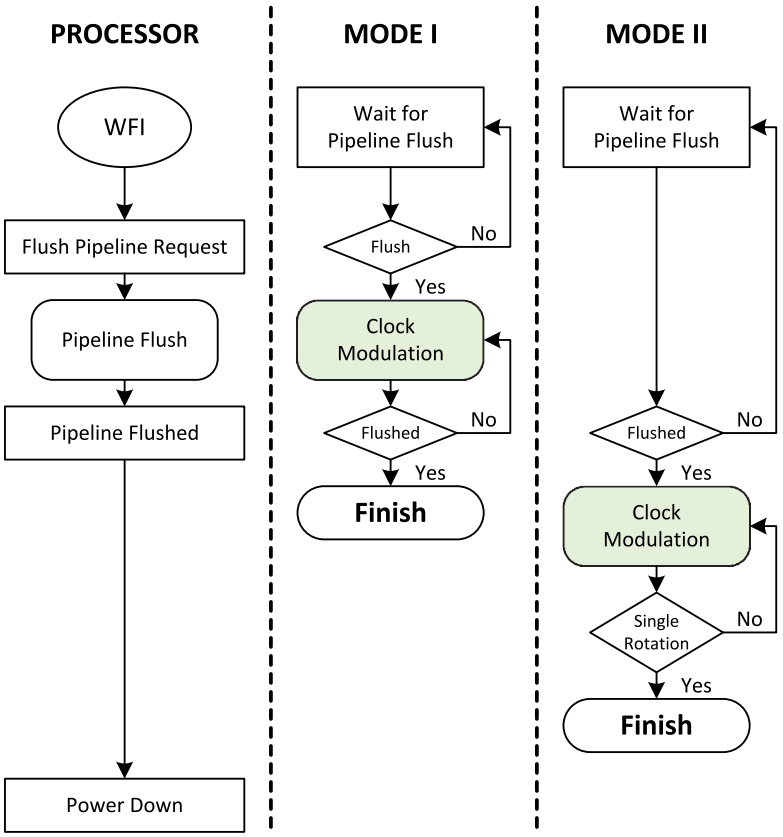


Figure 6.2: Watermark activation techniques.

detection hard to perform. In the second approach (mode II), the generation of the watermark power pattern is performed immediately after the pipeline flush and before the processor is power gated. As all lower level clock gates have already been switched off, the data is retained and the clock modulation does not impact the functionality of a processor. The generated watermark power signal is expected to be of much lower amplitude, when compared with mode I. The number of buffers between the top level clock gate and lower level clock gates and their capacitive load is expected to be significant, to be sufficient for successful watermark detection. Furthermore, as the processor is inactive the SNR is higher.

The timing diagram of both design approaches for 7-bit Barker code is shown in Figure 6.3. Additional operations between clock gating and power gating are not shown. The timing diagram of the non-watermarked processor is shown in the top of Figure 6.3. The flush request signal (*Flush Req*) changes to '1' upon the execution of the WFI instruction and it is registered on the next clock cycle. This indicates the beginning of the pipeline flush operation (green). The duration of the pipeline flush depends on the number of active instructions that must be executed, prior to processor entering the low power mode. In Figure 6.3, the pipeline in the non-watermarked processor is flushed within 14 clock cycles. Next, the clock gate request signal (*CLK Gate Req*) changes to '1' and the clock signal to the processor is stopped. The entire operation from the

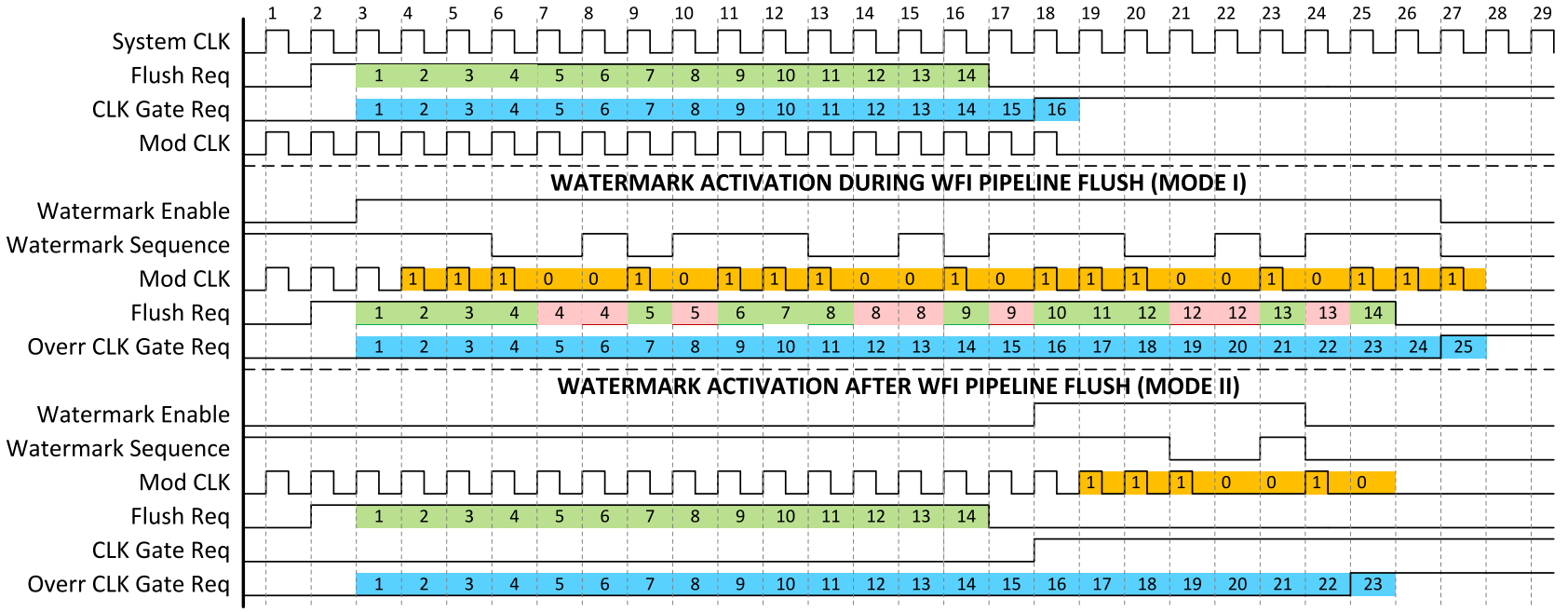


Figure 6.3: Timing diagram of the proposed technique for clock modulation based watermark power pattern generation with instruction based activation during the WFI pipeline flush (middle) and immediately after WFI pipeline flush (bottom).

active edge of the pipeline flush request to the clock cycle where the processor's clock is stopped requires 16 clock cycles (blue). In the first approach, the watermark circuit is active during the pipeline flush. On the active edge of the *Flush Req* signal the pipeline flush operation is requested. The watermark circuit is activated in the next clock cycle, and the generation of the watermark power pattern begins. The generated watermark sequence modulates the top level clock gate, indicated by the *Mod CLK* signal (orange). The clock signal is propagated (*Mod CLK* = *System CLK*) when watermark bit is '1' and is stopped (*Mod CLK* = 0) when watermark bit is '0'. As can be seen, when watermark is '1' the pipeline flush operation is active (green), and when watermark sequence is '0' the pipeline flush operation is stalled (red). The entire pipeline flush operation requires 23 clock cycles, which is as expected due to approximately 50% of '1' in the 7-bit Barker code, such as

$$7 - bit\ Barker\ code\ Hamming\ Weight = 4/7 = 0.57 \tag{6.1}$$

$$Non - Watermarked\ Pipeline\ Flush\ Interval = 14\ clock\ cycles \tag{6.2}$$

$$Expected\ Watermarked\ Pipeline\ Flush\ Interval = (14 - 1)/0.57 \tag{6.3}$$

$$= 22.8 \approx 23\ clock\ cycles \tag{6.4}$$

In Equation (6.3), 1 clock cycle is subtracted $(14 - 1)$, since the watermark circuit is enabled with 1 clock cycle delay. In the second approach, the watermark circuit is activated after the pipeline has been flushed. The watermark circuit generates a single period of a watermark sequence (orange). As can be seen, the first approach has the greater impact on the processor's performance for long intervals of a pipeline flush. Nevertheless, it is expected that the performance impact is negligible for both approaches in the real application. This is further discussed in Section 6.5.

To validate the proposed technique, two ASIC designs were fabricated in a $65nm$ low leakage CMOS technology, with a nominal operating voltage of 1.2V, and were tested. The designs were completed using industry standard EDA tools. The architecture of both designs differs, hence they are discussed as two separate test cases.

6.3 Case I: Modulation of ARM® Cortex®-A5 Clock Signal With ARM® Cortex®-M0

In the first test case, chip II introduced in Chapter 4, Section 4.5 was used. The clock signal of the embedded ARM® Cortex®-A5 microprocessor core was modulated with the software executed on ARM® Cortex®-M0 microprocessor core.

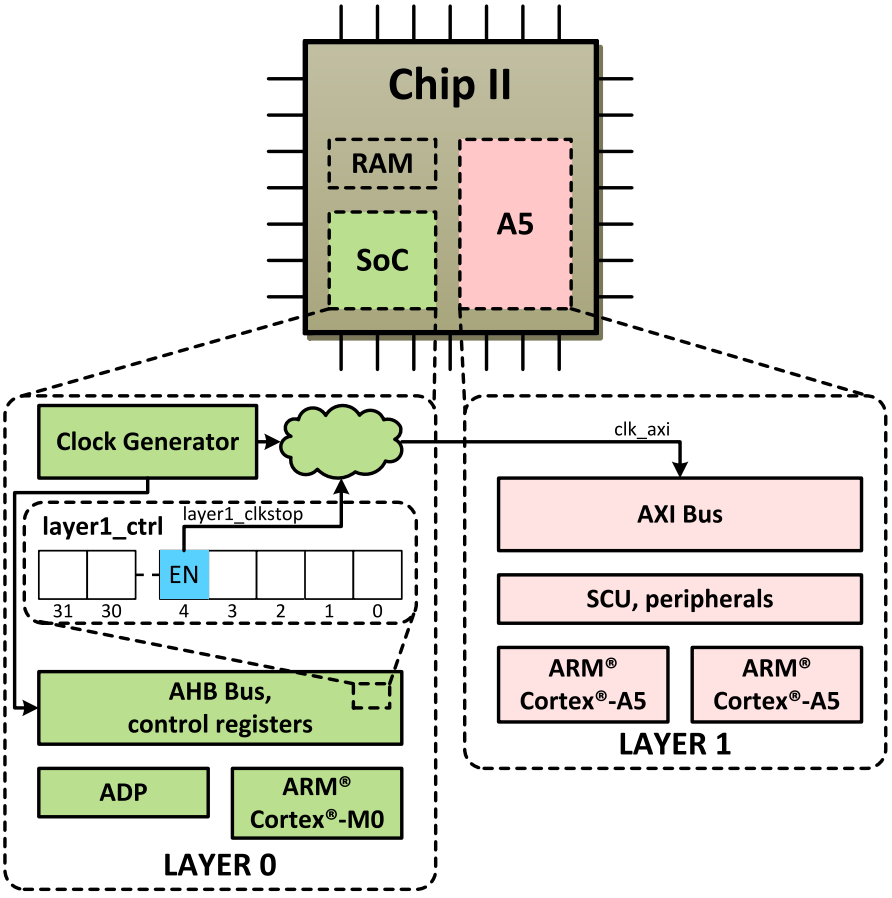


Figure 6.4: Block diagram of chip II implementation for the modulation of Cortex®-A5 with Cortex®-M0.

6.3.1 System Architecture

The architecture of the system is shown in Figure 6.4. As can be seen, the system is divided into 2 separate abstraction layers. The SoC in layer 0 consists of Cortex®-M0, Advanced High-performance Bus (AHB), control registers and generates a clock signal for both layers. The dual core Cortex®-A5, Snoop Control Unit (SCU) and Advanced eXtensible Interface (AXI) bus are located in layer 1. The memory, not shown in the figure, is shared by both layers. The on-chip ring oscillator is used to generate the clock signal. Through the control of *layer*1_*ctrl* control register, the modulation of layer 1 clock signal (*clk_axi*) is possible. As can be seen in Figure 6.4, bit 4 of *layer*1_*ctrl* serves as the enable signal (*layer*1_*clkstop*) for *clk_axi*. Therefore, if *layer*1_*clkstop* is controlled in a specific way the deterministic modulation of ARM® Cortex®-A5 can be achieved and the proposed technique can be validated.

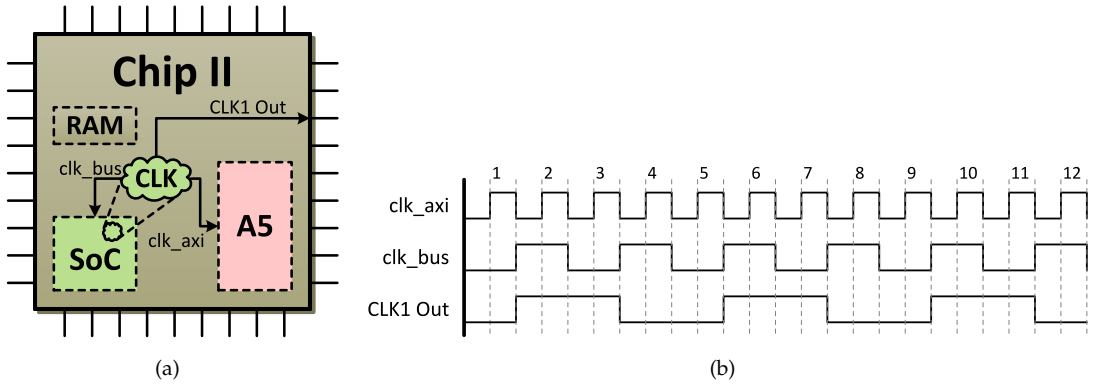


Figure 6.5: a) Block diagram of clock signal distribution on chip II. b) Timing diagram of clock signals on chip II.

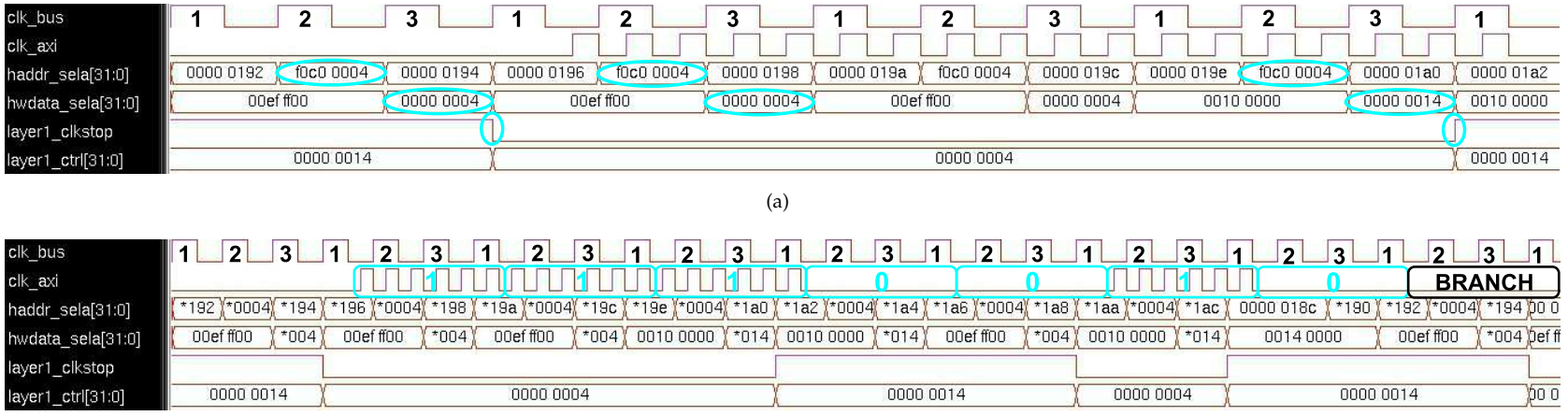
6.3.2 Experimental Setup

The test board shown in Chapter 4, Section 4.5.3 was used. All the power domains were connected using power jumpers and the total current consumed by the chip was measured, using a shunt $270m\Omega$ resistor. The generation of the clock signal using the on-chip ring oscillator is shown in Figure 6.5. The distribution of clock signals on chip II is shown in Figure 6.5(a). The clock generation circuit (*CLK*) embedded in SoC provides the clock signal for Cortex®-M0 (*clk_bus*) and dual core Cortex®-A5 (*clk_axi*). Additionally, the clock signal (*CLK*1 *Out*) is connected to one of the output pins and can be used for measurement and synchronization with the internal logic. As can be seen in Figure 6.5(b), *CLK*1 *Out* is asynchronous when compared with *clk_axi* and its frequency is 4 times less than the frequency of *clk_axi* and 2 times less than the frequency of *clk_bus*. Therefore, the experimental process discussed in Chapter 3, Section 3.3 differs and it is modified in Section 6.4.4. The operating frequency of Cortex®-M0 was $72.6MHz$, and the operating frequency of Cortex®-A5 was $145.2MHz$. The current signal was measured using an Agilent DSO80204B [123] oscilloscope with Agilent 1130A active differential probe, at a sampling frequency of $5GHz$. Due to the detection methodology (see Section 6.3.3), the average number of samples per *CLK*1 *Out* clock was 137. Therefore, 34 samples per single cycle of *clk_axi* were averaged, to find the power vector, Y. The number of clock cycles obtained for a single ρ was approximately $300k$.

The proposed technique activates the WGC circuit with the WFI instruction. To modulate the clock signal of one of Cortex®-A5 microprocessor cores and emulate the effect of the embedded WGC in the top level clock gate, the program executed by Cortex®-M0 controls the switching activity of the *clk_axi* signal. To validate mode II, the system noise level was significantly reduced by keeping Cortex®-A5 in the idle mode. Therefore, the processor did not execute any program. To achieve the modulation of Cortex®-A5 clock signal, register *layer*1_*ctrl* was controlled with the program executed on Cortex®-M0. The function for the modulation of *layer*1_*clkstop* is given below.

```
1  __asm void modulate_clock(void)
2  {
3       LDR  R0,=0xF0C00004  // layer1_ctrl register address
4       LDR  R1,[R0]         // load current layer1_ctrl register content
5       LDR  R2,=0x00000010  // layer1 stop clock mask
6       LDR  R3,=0xFFFFFFEF  // layer1 enable clock mask
7
8  barker7                   // --- barker 7 sequence
9       ANDS R1,R1,R3        // enable layer1 clock  (watermark = 1)
10      STR  R1,[R0]         // store content to register
11      ANDS R1,R1,R3        // enable layer1 clock  (watermark = 1)
12      STR  R1,[R0]         // store content to register
13      ANDS R1,R1,R3        // enable layer1 clock  (watermark = 1)
14      STR  R1,[R0]         // store content to register
15      ORRS R1,R1,R2        // disable layer1 clock (watermark = 0)
16      STR  R1,[R0]         // store content to register
17      ORRS R1,R1,R2        // disable layer1 clock (watermark = 0)
18      STR  R1,[R0]         // store content to register
19      ANDS R1,R1,R3        // enable layer1 clock  (watermark = 1)
20      STR  R1,[R0]         // store content to register
21      ORRS R1,R1,R2        // disable layer1 clock (watermark = 0)
22      STR  R1,[R0]         // store content to register
23      B barker7
24 }
```

The function starts by defining Cortex®-M0 working registers. As can be seen, the function is an infinite loop and generates a 7-bit Barker code. The generation of each watermark bit is comprised of 1 logical operation (*ANDS* or *ORRS*) and a store instruction (*STR*). Since store instruction requires 2 clock cycles a single watermark sequence bit requires 3 clock cycles (*clk_bus*). After the generation of a single watermark period, the branch (*B*) is performed and the watermark generation is re-executed. Similarly, the branch instruction requires 3 clock cycles. In Figure 6.6(a), the initial part of watermark generation is shown. As can be seen, the logical operation takes place in the 1^{st} clock cycle of *clk_bus*, while store instruction is executed in the 2^{nd} and 3^{rd} clock cycles. First, the address of *layer*1_*ctrl* (0*xF*0*C*00004) is pushed onto the address bus of the AHB bus (*haddr_sela*). On the next clock cycle, the result of the logical operation is pushed onto the data bus of the AHB bus (*hwdata_sela*). The result is stored in *layer*1_*ctrl* on the next active edge of *clk_bus*. It can be seen, that *layer*1_*clkstop* changes from 1 to 0, and the *axi_clk* is enabled shortly after. In Figure 6.6(b), the generation of a full period of a watermark sequence is shown, followed by a branch instruction. As previously discussed, the generation of a single watermark sequence bit requires 3 clock cycles. However, since the clock signal of Cortex®-A5 is of higher frequency, the same watermark generation period takes 6 clock cycles of *axi_clk*.



(a)

(b)

Figure 6.6: Timing diagrams of the modulated Cortex®-A5 clock signal, depicting the beginning of the program (a) and a single period of a watermark sequence (b).

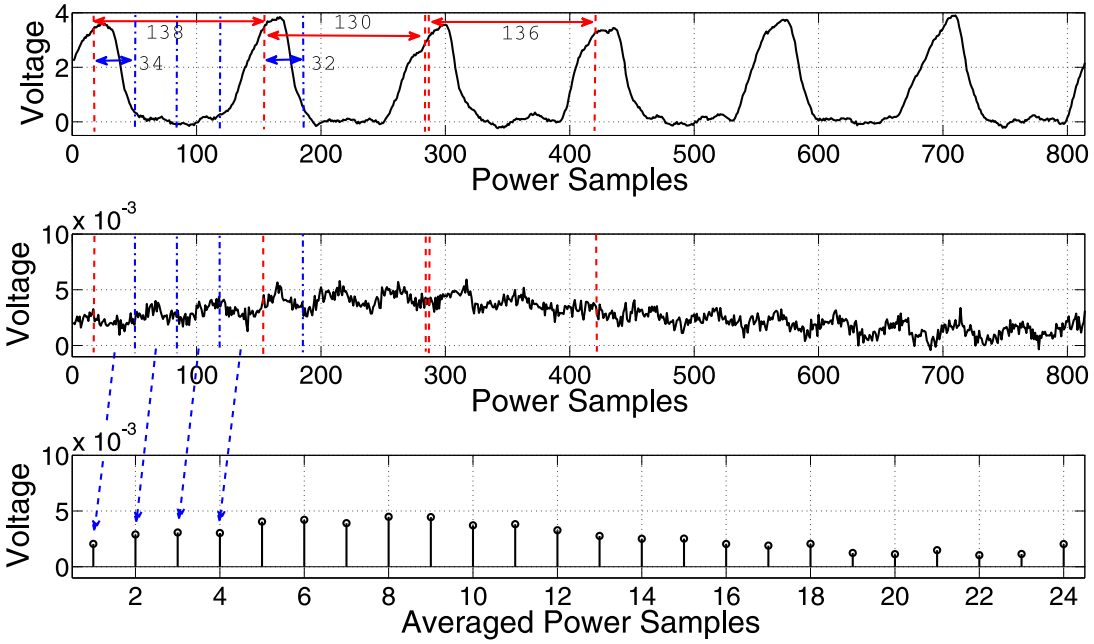


Figure 6.7: Representation of the averaged power signal (bottom), from the captured clock (top) and power (middle) signals in chip II.

6.3.3 Detection Methodology

In Chapter 3, Section 3.3, the power vector Y is obtained by averaging all power samples in a single clock cycle. In Section 6.3.2, the ring oscillator is used to generate the clock signal to various parts of a system. The output frequency of a ring oscillator depends on a propagation delay, which is influenced by temperature, supply voltage, load capacitance, threshold voltage, channel length, oxide thickness and transistor channel width [81]. Although the effect of temperature variations can be minimized through the use of the temperature chamber, other parameters are susceptible to power supply noise and process variations. Therefore, the frequency of the generated clock signal varies in consecutive clock cycles. To reduce such effect on detection performance each clock cycle is considered individually. The measured with the oscilloscope power signal is shown in Figure 6.7. To obtain the power vector Y and calculate the Pearson's correlation coefficient, ρ, the power measurements are post-processed as follows:

1. The synchronization point is found.
2. The number of samples, N_i, for a particular *CLK*1 *Out* clock cycle is computed.
3. N_i is divided by 4, to find the number of samples, N_{axi}, per *clk_axi* clock cycle.
4. The shift ratio between *CLK*1 *Out* and *clk_axi* is determined.
5. The power vector component, y_i, for *clk_axi* clock cycle is found and the process is repeated for the duration of *CLK*1 *Out* clock cycle.

6. The next clock cycle is found by adding N_i to the synchronization point.

7. Steps 3 – 6 are repeated for *CLK*1 *Out*, to find Y.

First, the synchronization point is found based on the threshold voltage in the measured *CLK*1 *Out* signal. For example, in Figure 6.7, $2V$ determines the beginning of the clock signal. Next, the number of samples, N_i, is determined for a particular clock cycle (red dotted line). As can be seen, the first clock cycle consists of 138 samples and the following clock cycles consist of 130 and 136 samples, respectively. Furthermore, as the frequency of *clk_axi* is 4 times higher than the frequency of *CLK*1 *Out*, N_i is divided by 4, to find the number of samples, N_{axi}, per single *clk_axi* clock cycle (blue dotted line). In case result is not an integer, the fractional components are not included, hence:

$$138/4 = 34.5 \approx 34$$

The shift ratio between *clk_axi* and *CLK*1 *Out* must be found, to determine the starting point of *clk_axi* clock cycle. In Figure 6.7, the shift ratio equals 0.5, which is as expected when Figure 6.5(b) is considered. The shift ratio is given as the ratio of the number of samples between the starting point of *clk_axi* and synchronization point and the length of the *clk_axi* clock cycle. Finally, the power vector component, y_i, for the particular *clk_axi* clock cycle is found by averaging all power samples within the *clk_axi* clock cycle. In Figure 6.7, 4 power vector components are found for a single *CLK*1 *Out* clock cycle. To find the following components, N_i is added to the synchronization point. At times, the synchronization points may not overlap, due to removal of fractional components and various number of samples in consecutive clock cycles. As can be seen in Figure 6.7, a few samples exist between the end of the 2^{nd} and the beginning of the 3^{rd} clock cycle, which are not included in the calculation of ρ. Nevertheless, the percentage of lost data is negligible and commonly only up to a few samples may be lost per single *CLK*1 *Out* clock cycle. The operation is repeated multiple times until the power vector Y is found.

6.3.4 Detection Algorithm

As shown in Figure 6.6, the modulation of a Cortex®-A5 with a Cortex®-M0, requires 3 *clk_bus* clock cycles and 6 *axi_clk* clock cycles, to emulate the WPPG circuit. Since the power vector Y represents the consumed power with respect to *clk_axi*, 6 power vector components (y_i) represent 1 clock cycle of a watermark sequence. However, as the phase of the *clk_axi* with respect to a watermark sequence is not known the detection algorithm is as follows:

1. Create a mask vector X_M for the power vector Y.

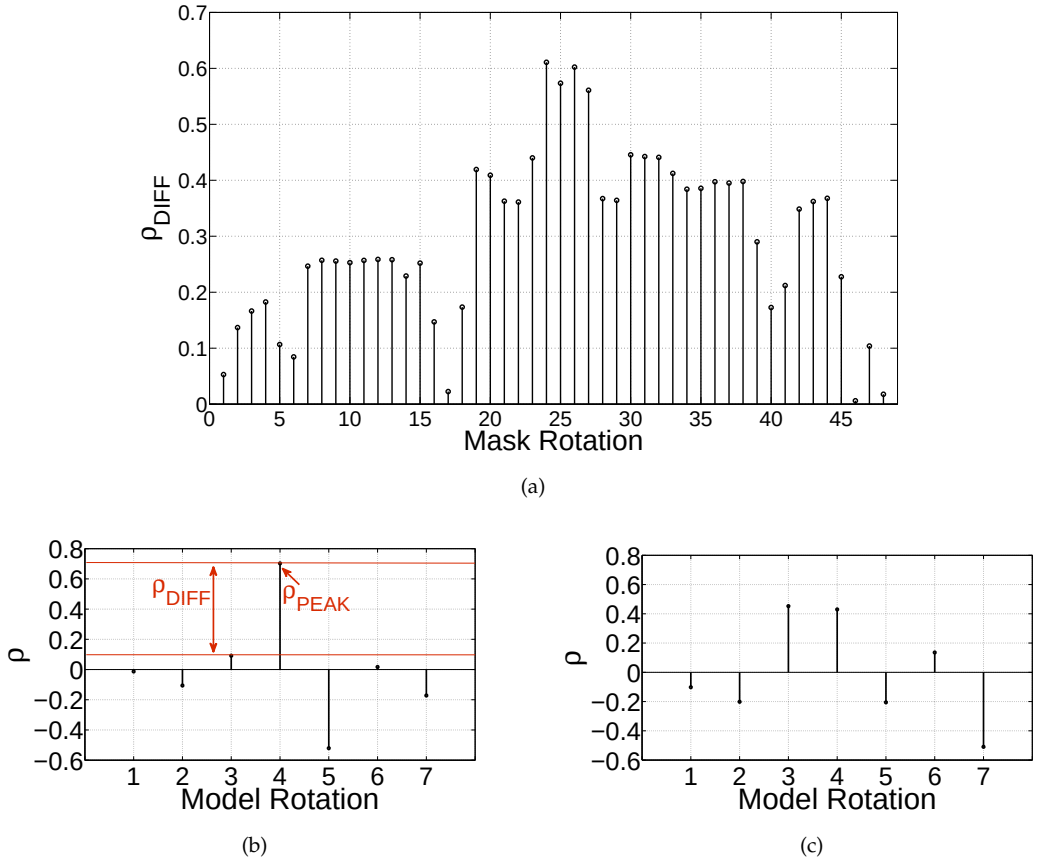


Figure 6.8: (a) Results of ρ_{DIFF} while varying the mask rotation (chip I). Spread spectra when X_M and Y are in phase (b) and are not in phase (c).

2. Mask Y to obtain the masked power vector, Y_M.
3. Average 6 consecutive elements in Y_M, to find the average power per watermark sequence bit.
4. Find spread spectrum graph.
5. Find a single correlation coefficient difference, ρ_{DIFF} and plot as in Figure 6.8(a).
6. Rotate X_M and repeat steps 2-5.

The detection algorithm requires the mask vector, X_M, to be created to remove the information about the branch instruction. Since each watermark bit requires 6 *clk_axi* clock cycles, X_M is created as follows (for 7-bit Barker code):

$$\textit{Mask for single watermark period} = 1111111$$
$$\textit{Mask for single watermark period with branch} = 1111111\mathbf{0}$$
$$\textit{Mask vector } (X_M) :$$
$$111111\ 111111\ 111111\ 111111\ 111111\ 111111\ 111111\ \mathbf{000000}$$

After the branch information has been removed from Y, using X_M, the masked power vector Y_M is obtained. Next, Y_M, is correlated with the watermark model, X, to find the spread spectrum graph. The correlation difference, ρ_{DIFF}, is found and plotted on a graph, Figure 6.8(a). Since the phase between X_M and Y is not known X_M is rotated by a single clock cycle and the new Y_M is found. The process is repeated until a full rotation of X_M has been performed and all ρ_{DIFF} have been plotted on the graph, Figure 6.8(a).

6.3.5 Experimental Results

The experimental results are shown in Figure 6.8(a). As can be seen, due to rotation of X_M, ρ_{DIFF} varies and it is highest when rotation is 24. Furthermore, it can be noticed that other high value ρ_{DIFF} elements exist. To understand this, consider the following situation. When X_M and Y are in phase the branch instruction information is removed and the highest ρ_{DIFF} occurs. When X_M and Y are not in phase the watermark information is partially removed and it is replaced with the information of a branch instruction. Furthermore, the rotation of X_M causes the starting point to move and it can be approximated to the effect of the convolution of the two signals. In Figure 6.8(c) and Figure 6.8(b) the spread spectra are shown for two X_M rotations. As can be seen, when X_M and Y are not in phase (Figure 6.8(c)) ρ_{DIFF} approaches 0, due to spurious correlation coefficients. Otherwise, when X_M and Y are in phase (Figure 6.8(b)), ρ_{DIFF} is significantly higher and demonstrates the successful watermark detection. This validates mode II of the proposed technique.

6.4 Case II: Modulation of ARM® Cortex®-A5 Top Level Clock Gate

In the second test case, the watermark circuit was embedded in the ARM® Cortex®-A5 microprocessor core, integrated as a part of a system.

6.4.1 Test Chip Overview

The test chip was fabricated using the 65nm low leakage CMOS technology, with the nominal operating voltage of 1.2V. The design was completed using industry standard EDA tools. The Synopsys tool suite version E-2012.12 was used, including Design Compiler for synthesis and VCS for verification. For the ease of comparison with previous test chips, the test chip demonstrated in this section will be considered as 'chip III' in the rest of this thesis. The test chip is shown in Figure 6.9. It consists of a single core ARM® Cortex®-A5 microprocessor IP core and caches. The SoC shown as the unmarked circuitry in the top half of the chip layout, consists of ARM® Cortex®-M0

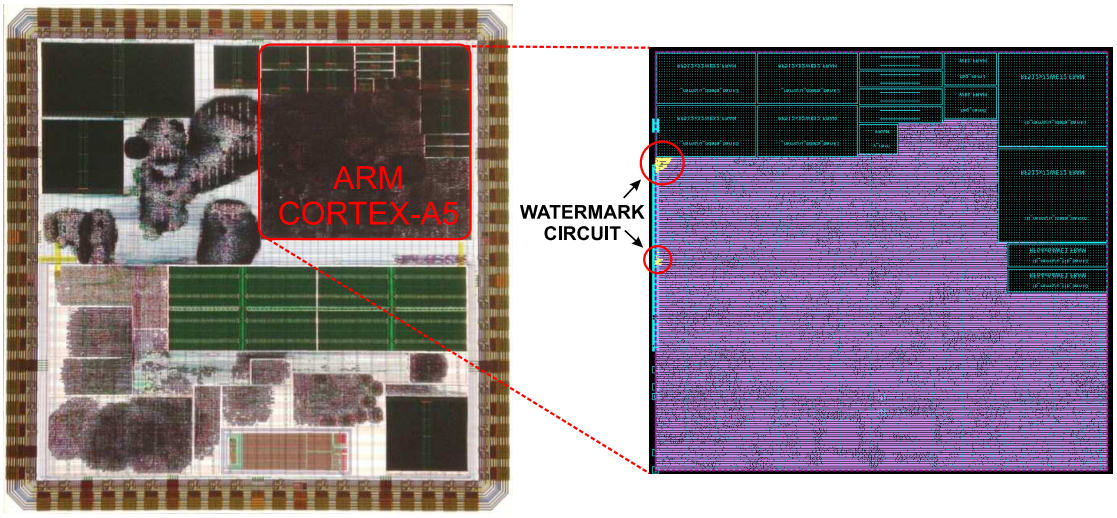


Figure 6.9: The layout (left) of chip III and ARM® Cortex®-A5 integrated on the chip (right), with the highlighted watermark circuit.

microprocessor IP core, ADP unit and numerous IP blocks. The bottom part of the chip formed part of another experiment and it is not considered in this thesis. The watermark circuit (see Figure 6.1) was integrated from an RTL description. Therefore, it was propagated through the entire design flow, which is closer to the intended usage scenario, when embedding watermarked soft IP.

6.4.2 Watermark Circuit Architecture

The architecture of the watermark circuit integrated in chip III is shown in Figure 6.1. For an in-depth explanation please refer to Appendix B.2.

6.4.3 Experimental Setup

The test board shown in Chapter 4, Section 4.5.3 was used. The operating frequency of ARM® Cortex®-A5 was 100MHz. The total current consumed by the chip was measured, using the shunt 5.6Ω resistor and an Agilent Technologies DSO80204B [123] oscilloscope with Agilent 1130A active differential probe, at a sampling frequency of 2*GHz*. The Dhrystone benchmark was executed on ARM® Cortex®-A5. The watermark activation differs from previous experiments, hence it requires new detection methodology, discussed in Section 6.4.4.

6.4.4 Detection Methodology

The detection methodologies discussed in previous chapters consider the watermark as active at all times throughout the duration of the experiment. Therefore, to acquire a

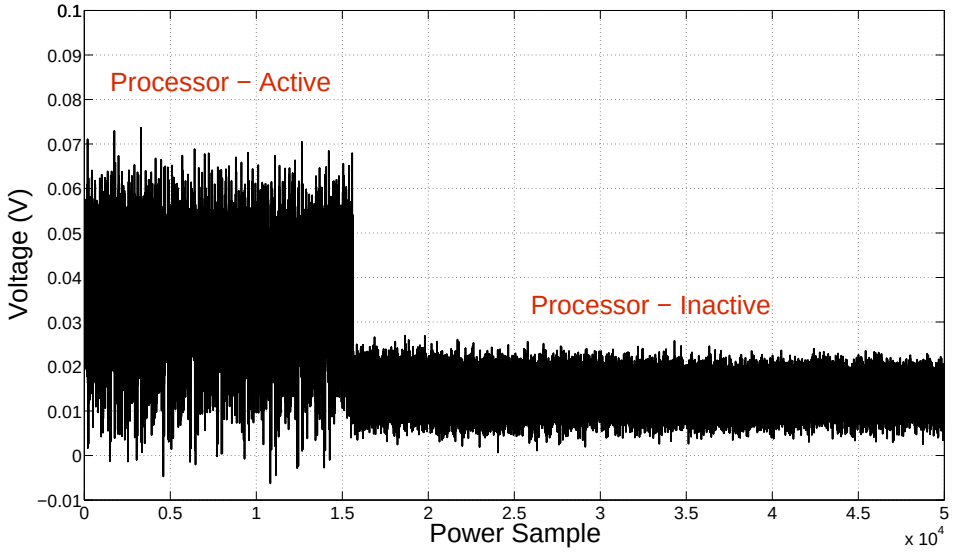


Figure 6.10: The effect of WFI instruction on a processor dynamic power consumption.

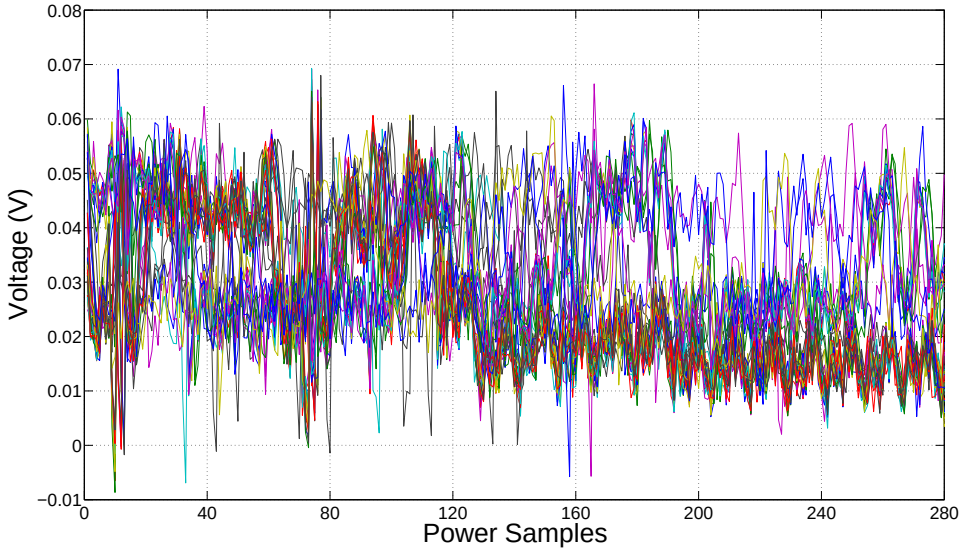


Figure 6.11: Power signals obtained from oscilloscope (reduced to processor switch point from an active mode to sleep mode).

power vector Y, a simple external trigger signal can be generated using FPGA (refer to Chapter 3, Section 3.3.1). To minimize the dynamic power overhead, the proposed technique activates the watermark circuit with the WFI instruction executed on a processor. Therefore, the activation periods are non-deterministic and hard to predict. However, the WFI instruction is commonly used before the processor enters a low power mode. It is known that the dynamic power consumption is much lower after the pipeline flush

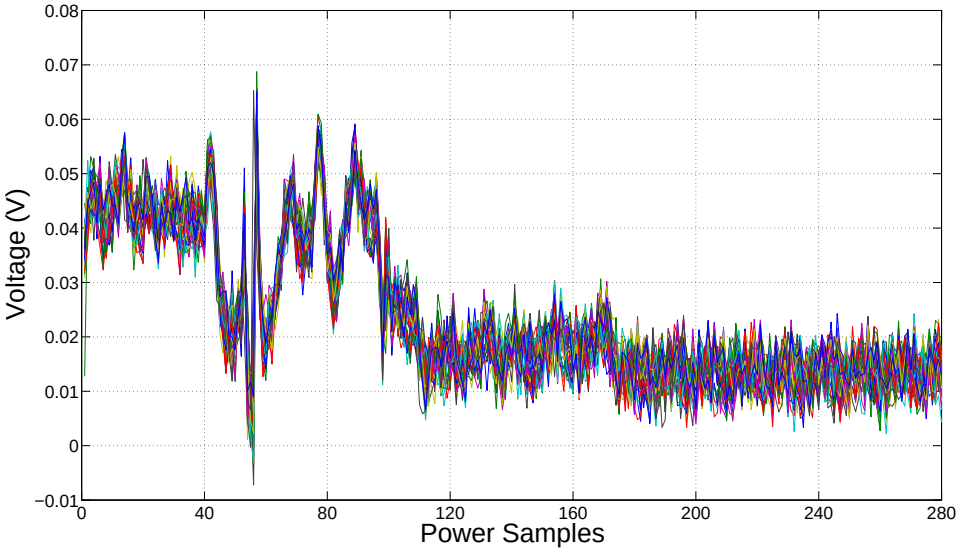
has been performed. In Figure 6.10, the dynamic power consumption during the processor active and sleep (inactive) modes is shown. As can be seen, the dynamic power consumption is significantly reduced when WFI instruction is executed and the pipeline has been flushed. Therefore, it is used as a trigger signal to detect the watermark power pattern.

6.4.5 Detection Algorithm

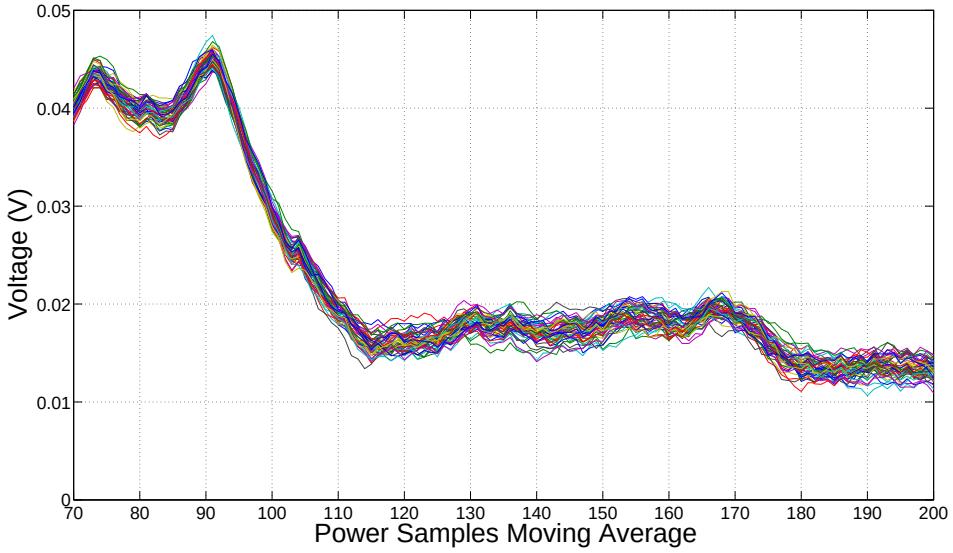
In Figure 6.10, the substantial reduction in the power consumption can be noticed, when the processor switches from the active mode to sleep mode. Although it is used to trigger the oscilloscope measurement, the obtained power traces are not in phase (Figure 6.11), due to background noise present in the signal, which causes the oscilloscope to trigger at various times. However, to find power vector Y, multiple power traces must be merged (refer to Chapter 3, Section 3.3.1). Furthermore, such signals must be in-phase to ensure the continuity of a watermark sequence. To synchronize power traces, each signal is processed as follows:

1. The length of each signal is reduced, to retain the power data when processor switches from the active mode to sleep mode.
2. The reference power trace is chosen.
3. A power trace is correlated with the reference power trace.
4. A power trace is rotated and the correlation is repeated.
5. All correlation values are plotted on a spread spectrum graph.
6. A power trace is synchronized with the reference power trace, based on the highest correlation value in a spread spectrum graph.
7. Steps 3 – 6 are repeated for other power traces.

To obtain the synchronized power traces and form a longer power vector Y, the size of each power trace is minimized to reduce the post-processing time. To reduce the noise present in the power signal and ease the detection of the processor mode switch point, the moving average is applied, where several consecutive power samples are averaged and represent a single point on a graph. The higher the number of averaged power samples the lower the noise present in the obtained signal. However, the less accurate the switch mode point. In this section, 10 consecutive samples have been averaged. It was found to be sufficient to reduce noise present in the power signal. To synchronize power traces, the reference power trace is chosen. In this section, the first power trace was used. Next, each trace is correlated with the reference trace. Since the phase shift



(a)



(b)

Figure 6.12: (a) Synchronized power traces. (b) Synchronized power traces after application of moving average technique.

is not known, the currently processed signal is rotated and the correlation is repeated. The spread spectrum graph is obtained for each trace. The maximum correlation peak represents the synchronization point and the distance by which the power trace must be rotated to be synchronized with the reference signal. In Figure 6.12, the synchronized (Figure 6.12(a)) and the moving average (Figure 6.12(b)) power traces are shown. As can be seen in Figure 6.12(b), the processor switch from the active to sleep mode is clearly distinguishable.

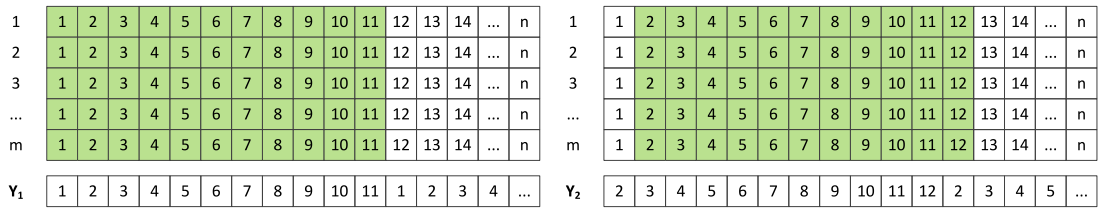
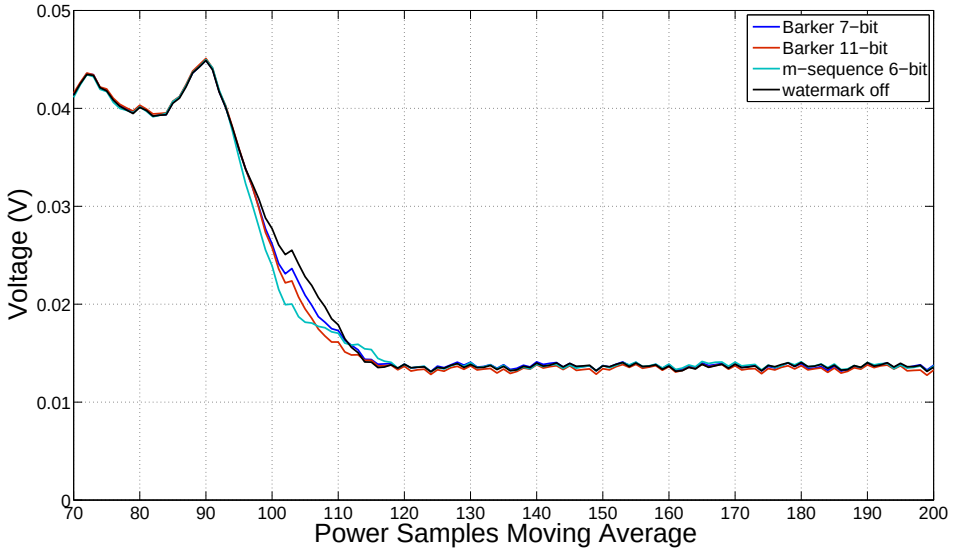


Figure 6.13: Power vectors extracted from consecutive columns in a power trace matrix. (a) Power vector Y_1. (b) Power vector Y_2.

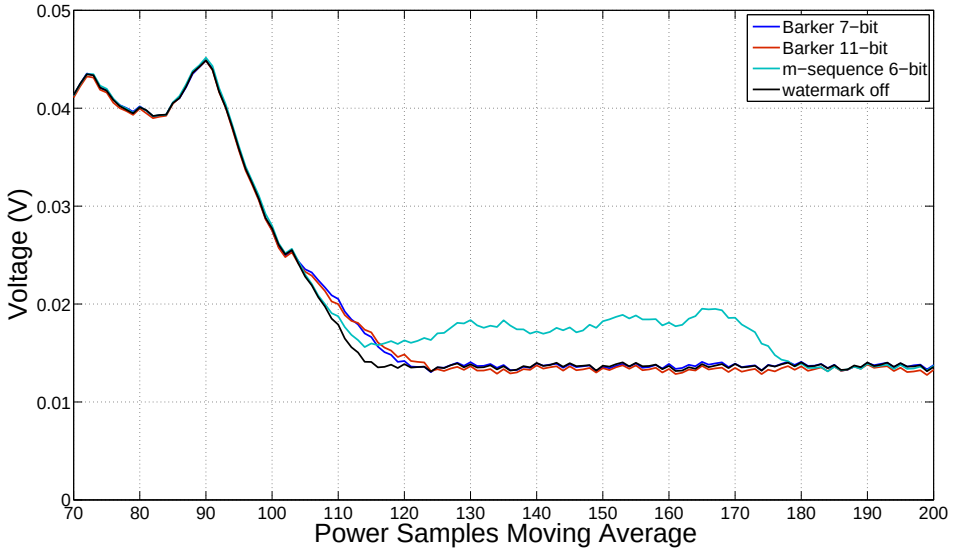
To perform the watermark detection the synchronized power traces are formed as a matrix (Figure 6.13), with rows (m) representing the number of power traces and columns (n) representing the power samples. To obtain the power vector Y, several power samples from each power trace are merged. For example, for 11-bit Barker code, 11 power samples from all m rows form a longer power vector Y (Figure 6.13(a)). Similarly, for 6-bit m-sequence 63 power samples are merged. Therefore, to obtain Y of approximately $300k$ samples, $27k$ power traces are required for 11-bit Barker code and $5k$ for 6-bit m-sequence. However, since the active period of a watermark circuit is not known, the consecutive columns in a matrix form new power vectors (Figure 6.13(b)). The correlation is computed for each vector and plotted on a spread spectrum graph. The process is repeated for Y_{n-1} power vectors.

6.4.6 Experimental Results

The experiments have been performed for both mode I and mode II implementation. In Figure 6.14(a), the moving average graph is shown for deterministic sequences discussed in Chapter 4, for mode I. The moving average is shown, since it removes a significant portion of noise present in the original trace. It is obtained by averaging the traces of Figure 6.12(b). When the watermark is off, the top level clock gate is enabled until the pipeline has been flushed. Therefore, the pipeline flush requires the least number of clock cycles to finish, but the power consumption is highest. As can be seen in Figure 6.14(a), when the watermark is off the amplitude is highest, but the processor enters the sleep mode most quickly. When watermark is active, the pipeline flush requires more clock cycles to finish, due to the modulation of the top level clock gate. Based on Table 4.2 (Chapter 4, Section 4.3), the *activity factor*, α, of 7-bit Barker code is higher (57.15%), than α of 11-bit Barker code (45.45%). Therefore, the amplitude of the signal in Figure 6.14(a) for 7-bit Barker code is higher, than the amplitude of the signal for 11-bit Barker code. In case of 6-bit m-sequence, the sequence starts with a few leading zeroes. This can be noticed in the figure by a sudden drop in the signal's amplitude and the longest pipeline flush of all sequences. However, the pipeline flush



(a)



(b)

Figure 6.14: Power signal moving average for (a) mode I, (b) mode II.

is too short and the processor enters the sleep mode before the watermark sequence reaches the full period of a 6-bit m-sequence.

In Figure 6.14(b), the moving average graph is shown for mode II. When the watermark is off, the signal is the same as in Figure 6.14(a). However, when the watermark is active a single period is generated. As can be seen, the signals' amplitudes are much higher after the pipeline has been flushed. When Barker codes are compared, the amplitude for the 7-bit Barker code is marginally higher in the first part of a signal. However, as the 11-bit Barker code is a longer sequence, the signal is flatter and the amplitude is

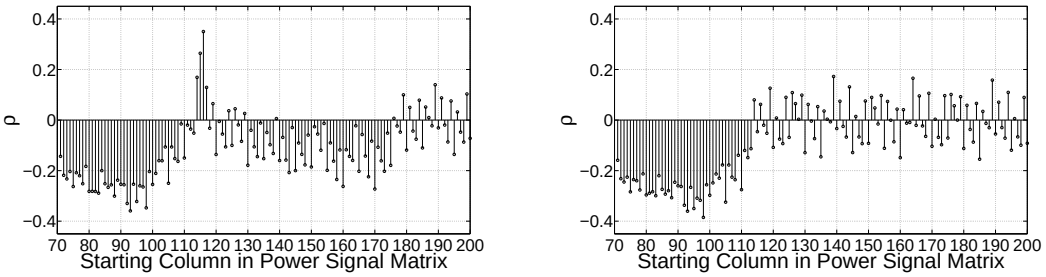


Figure 6.15: Spread spectra of correlation results for 6-bit m-sequence when watermark circuit is active (a) and inactive (b), after the pipeline has been flushed.

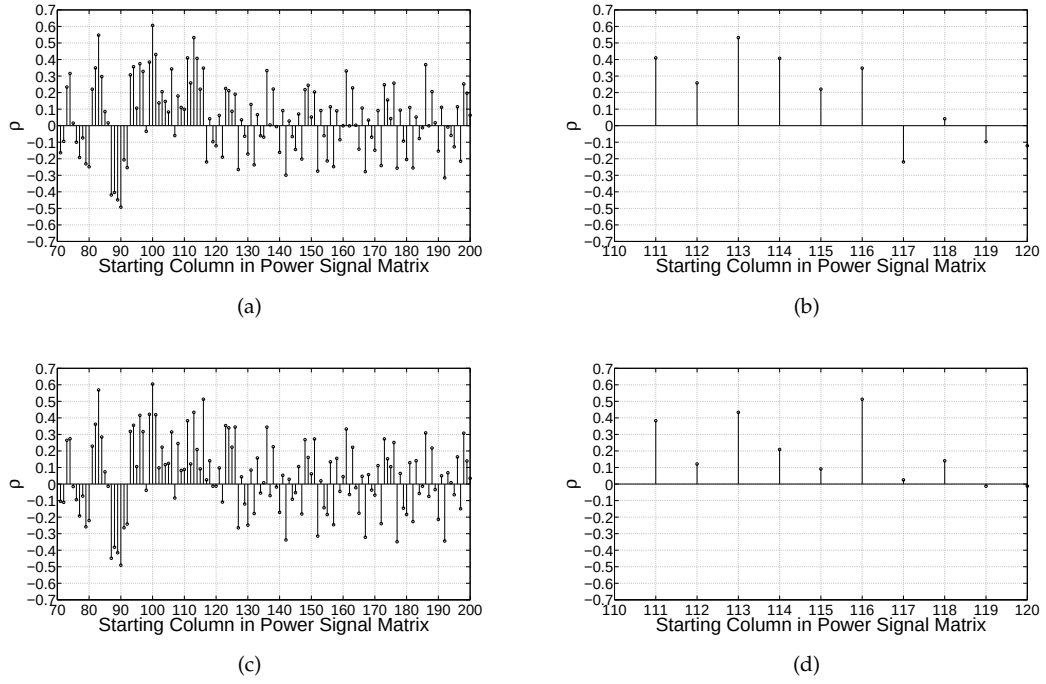


Figure 6.16: Spread spectra of correlation results for 11-bit Barker code active after the pipeline has been flushed. (a),(b) Watermark inactive. (c),(d) Watermark active.

higher in the last part of a signal. If a 6-bit m-sequence is considered, it can be seen that due to much larger sequence length (Table 4.2, Chapter 4, Section 4.3), the watermark circuit is active for longer and the watermark is clearly distinguishable. To determine if the watermark has been found, the spread spectrum is analyzed (refer to Chapter 3, Section 3.3). In Figure 6.15, the correlation results for 6-bit m-sequence are shown for mode II. As can be seen in Figure 6.15(a), the significant correlation peak occurs when watermark is active, when compared with results in Figure 6.15(b), where watermark is off. Therefore, the watermark can be considered as detected.

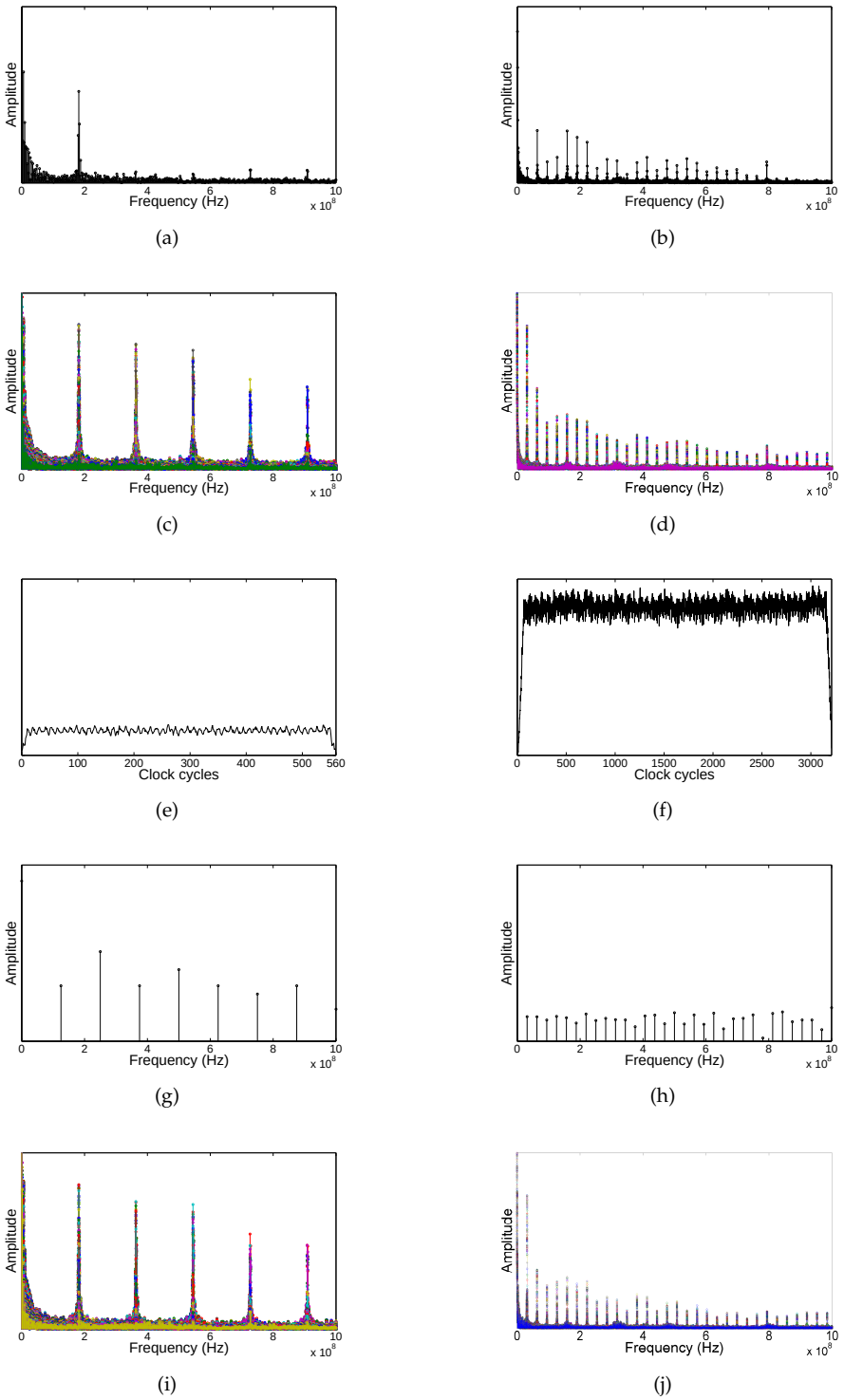


Figure 6.17: Frequency spectra of power vectors when watermark was off for (a) 11-bit Barker code (single vector), (b) 6-bit m-sequence (single vector), (c) 11-bit Barker code (all vectors), (d) 6-bit m-sequence (all vectors). Convolution of power signal with watermark for (e) 11-bit Barker code, (f) 6-bit m-sequence. Frequency spectra of (g) 11-bit Barker code, (h) 6-bit m-sequence. Frequency spectra of power vectors when watermark was on after the pipeline flush for (i) 11-bit Barker code, (j) 6-bit m-sequence.

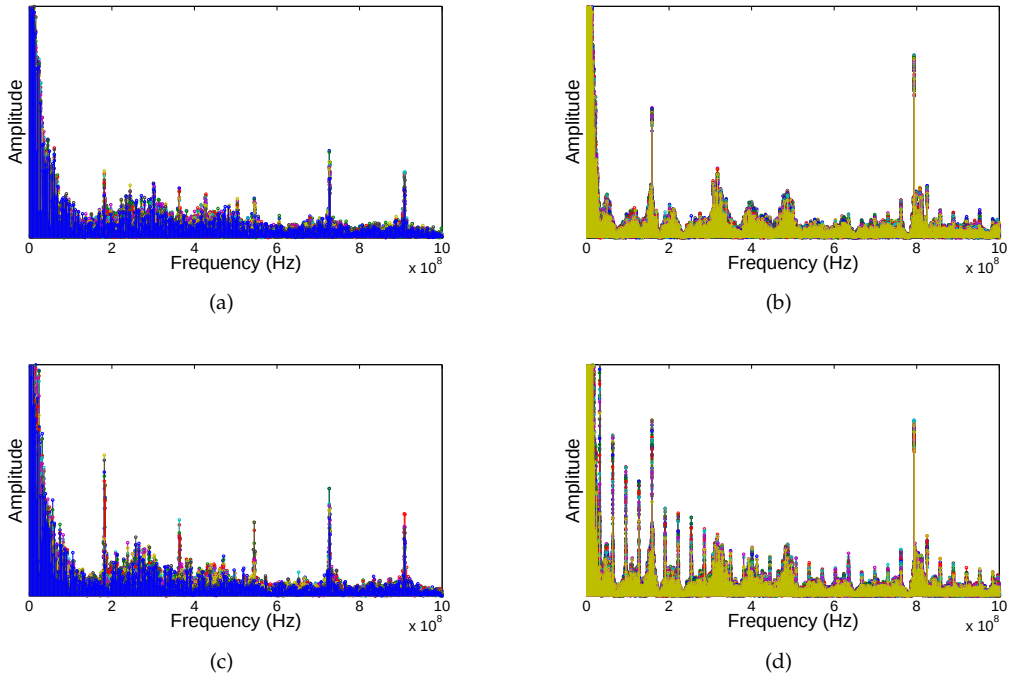


Figure 6.18: Frequency spectra of the convolved power signal with the watermark sequence for (a) 11-bit Barker code (watermark off), (b) 6-bit m-sequence (watermark off), (c) 11-bit Barker code (watermark on - after) and (d) 6-bit m-sequence (watermark on - after).

Correlation spectra for 6-bit m-sequence for mode I could not be performed, since the full period of a 6-bit m-sequence was not generated (Figure 6.14(a)). Furthermore, the watermark could not be found in any modes for 7-bit and 11-bit Barker codes. In Figure 6.16, the spread spectra for 11-bit Barker code are shown for mode II. Results for mode I are similar. In Figure 6.16(a), the watermark is off and no modulation of the top level clock gate is performed. In Figure 6.16(c), the watermark is active and the top level clock gate is modulated. As can be seen, the figures are similar and no significant correlation peaks can be distinguished. However, it is interesting to notice that the correlation values when watermark is active ($116 - 120$ on the x-axis in Figure 6.16(d)), are higher than correlation values when watermark is off ($116 - 120$ on the x-axis in Figure 6.16(b)). However, the correlation values do not cause any significant correlation peaks and the watermark cannot be detected. To understand why this occurs, consider the frequency spectra of power signals in Chapter 4 and Chapter 5. The watermark circuit is active throughout the experiment. Hence, the background noise varies and can be approximated as white noise (see Figure 4.4, Chapter 4, Section 4.3.1). The generation of a watermark sequence is continuous and the watermark power signal can be regarded as an additional offset added to the background noise. In this chapter, however, the watermark sequence is active during or immediately after the pipeline flush operation. Therefore, the frequency spectra must be reconsidered.

The detection algorithm (refer to Section 6.4.5) merges a part of each power signal to generate a longer power vector, Y. In Figure 6.17(a), the frequency spectrum of a single power vector is shown, when the watermark is off. As can be seen, the spectral power is not constant and few frequency components can be distinguished. In Figure 6.17(c), the spectra of all power vectors obtained during the experiment are shown. Similarly, none of the power vectors can be approximated as a white noise. The number of columns merged to form a power vector has a direct impact on the frequency spectrum of a vector. For example, in Figure 6.17(b) the frequency spectrum of a single power vector is shown and in Figure 6.17(d), the spectra of all power vectors are shown, when 6-bit m-sequence is considered. As can be seen, the number of frequency components is significantly higher, when compared with spectra for 11-bit Barker code. This is as expected if Figure 4.4 (Chapter 4, Section 4.3.1) is considered. As can be seen, the convolution of the background noise with 11-bit Barker code (Figure 6.17(e)) is lower than the convolution with 6-bit m-sequence (Figure 6.17(f)). However, when watermark is active the frequency components of watermarking sequences (Figure 6.17(g) and Figure 6.17(h)) are expected to be dominant. If the frequency spectra of power traces when watermark is active are considered (Figure 6.17(i) and Figure 6.17(j)), the frequency components of watermarking sequences cannot be clearly distinguished. Furthermore, from a simple visual inspection the spectra are similar to the spectra when watermark is off. However, if one compares the frequency spectra of power vectors when watermark is expected to be active, it can be seen that for 6-bit m-sequence the frequency spectra for the active watermark (Figure 6.18(d)) contain more frequency components corresponding to the sequence (Figure 6.17(h)), than the spectra for power vectors when watermark is off (Figure 6.18(b)). However, for 11-bit Barker code the frequency spectra of an active watermark (Figure 6.18(c)) contain only some of the watermark sequence frequency components, and in some cases increase the amplitude of the noise frequency components, when compared with Figure 6.18(a). Therefore, the correlation results do not produce significant correlation peaks, Figure 6.16. Addressing these issues form part of future research, discussed in Chapter 7.

6.5 Hardware Implementation Costs Reduction

The instruction based activation of the watermark circuit proposed in this chapter improves the hardware implementation costs of the technique introduced in Chapter 5. The clock modulation technique demonstrated the area overhead reduction close to 100% for bigger systems, such as application processors (i.e. ARM® Cortex®-A5), and close to 90% for smaller systems, such as microcontrollers (i.e. ARM® Cortex®-M0). The technique proposed in this chapter allows similar area overhead reduction. The watermark circuit implementation requires few minor modifications to the original processor architecture in the RTL description, to override the existing signals. The

watermark generation circuit requires only few registers to implement Barker codes or m-sequences. Therefore, the area overhead of the proposed technique is negligible and close to 0%. Moreover, the activation of the watermark circuit with dedicated instruction, such as WFI, allows further minimization of the dynamic power consumption. The dynamic power consumption depends on the current program and the frequency of the WFI instruction execution. Nevertheless, as demonstrated in Figure 6.14(a) and Figure 6.14(b), the watermark power pattern is generated for a short period. Therefore, it is expected that the dynamic power overhead for any program is negligible and close to 0%.

The clock modulation technique of Chapter 5 has shown a processor performance reduction of approximately 50% (refer to Chapter 5, Section 5.6). The technique proposed in this chapter builds upon the clock modulation technique, however, the performance overhead is negligible and synchronization issues do not occur. The performance overhead causes the increased latency of the pipeline flush operation (mode I) or adds the delay after the pipeline flush has finished and before the processor is power gated (mode II). It is expected, that the performance overhead is close to 0% and the synchronization issues do not occur in any mode. However, if longer pipeline flushes occur the watermark is generated multiple times and the performance overhead of mode I increases. Furthermore, mode I requires additional modifications to a processor architecture, to prevent vital parts of a processor being modulated during the pipeline flush operation. Mode II does not require such modifications since the processor has already finished execution of all tasks and it is ready to enter the sleep mode. Therefore, mode II requires less modifications of an original IP and it is a less invasive approach.

6.6 Improved Robustness

The robustness of the instruction based activation technique is inherited from the clock modulation technique of Chapter 5. The area overhead of the watermark circuit is negligible, which makes it harder to find in the RTL description when compared with technique in [8]. Additionally, the watermark circuit forms part of an existing system and its removal greatly impairs the system's functionality. Therefore, the robustness of the proposed technique against tampering attacks is significantly improved when compared with the current state-of-the-art in [8]. However, due to short sequences such as Barker codes, the number of possible combinations is reduced and brute force attacks become feasible. Hence, the robustness against finding ghosts and forging attacks is impaired (refer to Chapter 4, Section 4.7.1). To overcome this issue, m-sequences can be used in mode II implementation, as demonstrated in Section 6.4.6 (Figure 6.15). However, to achieve high robustness against finding ghosts and forging attacks longer m-sequences, such as 32-bit, must be used which is infeasible due to large performance overhead. Furthermore, such significant delay between the pipeline flush

operation and processor power down is easily noticeable by an attacker from a simple RTL simulation. To overcome the limitations of Barker codes, the private/public key encryption and cryptographic hash functions can be used as demonstrated in Chapter 4, Section 4.7.2. Determining the feasibility of such approach forms part of a future research, discussed in Chapter 7.

6.7 Concluding Remarks

The activation of a watermark based clock modulation with the WFI instruction technique has been introduced in this chapter. The technique was validated with two ASICs. A negligible area and power overheads was achieved, while synchronization issues do not occur. The experimental results demonstrated the influence of watermark sequences on processor dynamic power. A successful detection of a 6-bit m-sequence was achieved, for one of the watermark implementation approaches. Nevertheless, considering the robustness of the proposed technique, long period sequences such as 32-bit m-sequence are infeasible. Therefore, shorter period sequences must be used, to avoid robustness and performance overhead issues. However, the experimental results demonstrated that due to detection algorithm sequences such as 7-bit and 11-bit Barker codes could not be detected. To overcome this issue, the public/private key encryption and cryptographic hash functions can be used. This ensures lower amplitude of frequency components in the noise signal due to increased trace length and better approximation to the white noise and increased robustness against third party IP attacks. Determining the feasibility of such approach forms part of a future research.

Chapter 7

Conclusions and Future Work

The continuous technology scaling has increased the complexity of circuit design but it has lead to the design productivity gap and the need to accelerate the industry transformation. To shorten the design cycles and reduce design costs the design reuse approach has been introduced into the design flow, where new design blocks are integrated along the existing functional blocks sourced from external IP suppliers. To avoid the integration complications faced by SoC integrators, the sourced IP blocks are often supplied as unprotected design files but lack the appropriate protection mechanisms, which prevent an access to a design by a third party. As a result, the hardware piracy has become a major problem and IP protection has become a necessity. Auditing the presence of an IP in finished products through the application of reverse engineering techniques is an effective solution, but the process is slow and costly. Therefore, it is desirable to identify and prioritize IP candidates to be short-listed for more thorough investigation. This has lead to the development of many different IP protection techniques at various levels of the designed IP. This thesis focuses on protecting the IP in the soft level, due to its flexibility and high susceptibility to third party attacks. The contributions provide new techniques for minimization of hardware implementation costs of IP protection measures in embedded processors and increase robustness against hardware piracy. The contributions are summarized in the next section, followed the proposed areas for future work.

7.1 Thesis Contributions

Recent research has demonstrated an integration of a circuit known as digital watermark, as a source of IP supplier's information hidden in a protected IP. The information is extracted from a device and further used to determine if an IP infringement has occurred. The vast number of techniques for integration of digital watermark has been proposed. The most prominent techniques modify the Finite State Machines

(FSM) [35–40], or add a side-channel information through the implementation of a redundant circuit known as power watermark circuit, to generate the deterministic power pattern on a supply voltage rail [8,58–60]. The FSM techniques allow the hardware implementation costs close to 0%, but require an access to a watermarked device internals, such as IO ports or memory. The power watermarking techniques are less invasive and can be detected from a simple measurement of a device power consumption. Furthermore, such techniques can be applied on a black box device, where the architecture is not known, but the hardware implementation costs are much higher.

To ensure high robustness of a digital watermark against third party IP attacks, power watermarking techniques implement a 32-bit maximum length sequence (m-sequence), generated with the Linear Feedback Shift Registers (LFSR) [8,58–60]. Due to high number of possible combinations of an implemented power pattern, the detection of an embedded watermark is infeasible, but the hardware implementation costs are high. In Chapter 4, the first objective[1] of this thesis is addressed, by determining the most cost efficient sequences, in terms of area, power and detection, for the digital power watermark implementation. To find the intrinsic parameters of watermarking sequences and their impact on hardware implementation costs and detection performance, the Pearson's correlation coefficient was analyzed and two key parameters were extracted. Sequences not previously discussed in the area of IP power watermarking, such as Barker codes, were compared with the commonly used m-sequence. Based on the intrinsic sequence's parameters, MATLAB simulations have allowed to short-list the possible candidates. To validate the simulation results the watermark circuit was embedded in ARM® Cortex®-M0 implemented on FPGA. Additionally, the validation was performed on the first in the literature watermark circuit silicon implementation. Two ASICs have been fabricated (chip I and chip II), using the 65*nm* low leakage CMOS technology. In the first chip, the watermark circuit was integrated as a hard macro block on a separate power domain. The system consisted of ARM® Cortex®-M0 along with on-chip bus and numerous IP blocks. In the second chip, the watermark circuit was integrated from the RTL description which is closer to the intended scenario when implementing a watermarked IP. The system consisted of ARM® Cortex®-M0, on-chip bus and numerous IP blocks and ARM® Cortex®-A5 along with on-chip bus and caches. The FPGA and silicon results have confirmed the simulation results. Furthermore, the silicon results have demonstrated the effect of process variation on detection results. It was concluded that very short sequences are susceptible to process and run-to-run variations. The simulation and experimental results have allowed to determine the cost efficient sequences for power watermarking. The area overhead reduction of at least 72% and power overhead reduction of at least 73% was achieved with the proposed 7-bit and 11-bit Barker codes, whilst maintaining the same level of watermark detection as commonly used m-sequence. The robustness of the proposed sequences was analyzed, which formed part of the third objective, and it was shown that with the private/public

[1] All objectives are listed in Chapter 1, Section 1.4

key encryption and hash encoding high robustness against third party IP attacks can be achieved.

The watermarking sequences found in Chapter 4 have allowed an area and power overhead reduction, when compared with the m-sequence approach [8], but the architecture of the watermark circuit was the same as in the current state-of-the-art technique [8], and required a significant size Watermark Power Pattern Generator (WPPG) load circuit. The area overhead of the watermark circuit with 7-bit or 11-bit Barker codes was substantial. To address the second objective and reduce the hardware implementation costs, a new technique was proposed in Chapter 5. The WPPG circuit was removed. Instead, the watermark circuit was used to control the switching activity of the processor clock signal, to emulate a significant size WPPG circuit. When such technique is implemented in an embedded processor, the capacitive load of the clock tree buffers is utilized to achieve a high dynamic power consumption related to a watermark sequence. To validate the proposed technique, the watermark circuit implemented on chip I was used. To demonstrate the modulation of clock tree buffers with the watermark sequence, the WPPG circuit was active but no data switching occurred. Therefore, the only components contributing to a watermark power pattern were the clock tree buffers in WPPG registers. The experimental results have shown a clear correlation peak when watermark was active, which indicated a successful detection. The experimental results were repeated multiple times and 100% detection was achieved. The proposed technique was analyzed in terms of area and power overheads and it was shown that the area overhead is negligible, since only the WGC circuit must be implemented. Furthermore, the technique avoids the need of watermark circuit scaling, when the size of the system is increased. This is however not the case when the current state-of-the-art technique is considered [8], as the WPPG determines the SNR of the watermark power signal. To determine the power overhead of the proposed technique, the power consumption of the fully placed and routed watermark circuit implemented on chip I was estimated with Synopsys Primetime-Px. In smaller systems, such as microcontrollers (i.e. ARM® Cortex®-M0), the power overhead reduction is close to 90% and in bigger systems, such as application processors (i.e. ARM® Cortex®-A5), the power overhead reduction is close to 100%. The robustness of the proposed technique was analyzed as a part of the third objective of this thesis. It was demonstrated that the robustness is significantly improved, since the watermark circuit forms an integral part of an existing circuit and achieves a negligible area overhead.

The characterization of watermarking sequences in Chapter 4 and the proposed technique in Chapter 5 have allowed further hardware costs reduction. The clock modulation technique (Chapter 5) was combined with sequences found in Chapter 4, to activate the watermark circuit with a dedicated instruction, such as Wait for Interrupt (WFI), in Chapter 6. Two design approaches have been proposed and were validated on two test chips. In the first chip (chip II), the feasibility of the technique was confirmed. It was

shown that the watermark was detected when the clock signal of ARM® Cortex®-A5 was modulated with the program executed on ARM® Cortex®-M0. In the second chip (chip III), the proposed technique was implemented in ASIC, fabricated in 65*nm* low leakage CMOS technology. Both design approaches have been tested and experimental results for both m-sequence and Barker code demonstrated the effect of the proposed technique on dynamic power consumption. The watermark was successfully detected when m-sequence was used in mode II configuration. However, such sequences cannot be used due to their intrinsic significant length. Barker codes could not be detected in any of the proposed configurations. Therefore, experimental results were investigated and the improvement to the technique was demonstrated, which forms part of a future research. The proposed technique enables area and power overheads close to 0% and avoids the synchronization issues found in Chapter 5. The robustness is significantly improved when compared with the commonly used m-sequence technique [8], when private/public key encryption is used.

The contributions presented in this thesis provide novel techniques for reducing the hardware implementation costs of IP protection through digital power watermarking for embedded processors. The thesis demonstrates the first application of a watermark circuit in the ASIC implementation. The conclusions drawn in this thesis are supported through MATLAB simulations, FPGA and silicon validations and analysis of fully synthesized circuits. It is hoped that the proposed techniques will make useful contributions towards the protection of future embedded soft IP processors.

7.2 Future Work Directions

7.2.1 Improved Instruction Based Activation of Watermark Power Pattern

The watermark circuit activation with the WFI instruction (Chapter 6) have shown a successful detection of a 6-bit m-sequence, but could not detect any of the Barker codes. The analysis of frequency spectra of power signals measured with oscilloscope has clarified why such has occurred. Since m-sequences are infeasible (refer to Chapter 6, Section 6.6) and simple Barker codes are too short to be detected, the public/private key encryption and hash encoding of a side-channel message with Barker codes has been proposed. This ensures the longer period of the measured power signal, hence better approximation of the background noise signal to the white noise, and increased robustness against third party IP attacks. To achieve this, the watermark circuit must employ the encryption and hash encoding of a secret message on-chip and further modulate each bit with a watermark sequence. The technique must be analyzed in both modes (mode I and mode II, Chapter 6, Section 6.2) for 7-bit and 11-bit Barker codes, and hardware implementation costs such as area, power and performance overheads must be determined. It is expected that the power and performance overheads are negligible.

The area overhead in the ASIC implementation may be found to be significant, due to many additional circuits such as key encryptor and decryptor and hash encoder and decoder. To avoid such overhead in the final watermark implementation an already hash encoded message can be stored in a register. This ensures a negligible area overhead and improves the robustness of a watermark circuit.

7.2.2 Watermark Detection Through Power Classification

The detection technique used throughout this thesis is known as Correlation Power Analysis (CPA) and is considered as the current state-of-the-art technique for detection of an embedded power watermark, due to high sensitivity and ability to detect deeply embedded signals. However, as demonstrated through experimental results, the amplitude of a watermark power pattern is considerable and usually a significant size WPPG load circuit must be used. Techniques have been proposed to avoid the use of such circuit, with high robustness against third party attacks, but are limited to the modulation of the top level clock gate of a processor, due to significant dynamic power reduction which occurs after the pipeline has been flushed. This has been used in the experiments to trigger the measurement of the oscilloscope and capture the processor switch point from the active mode to the sleep mode. Despite high robustness against third party IP attacks and in case of key encryption high security of the encoded message, such obvious power trigger may be used by an attacker and traced back to the position of a watermark circuit. To overcome such limitations, it is necessary to generate a watermark power pattern during the active processor mode, when an additional watermark power consumption is obscured by a processor power consumption. However, the clock modulation technique of Chapter 5 cannot be used since this would lead to synchronization issues. Therefore, the WPPG load circuit must be considered, but the tradeoff between the area and power overheads and robustness against detection in the power consumption occurs, when technique of Chapter 6 is considered and must be investigated.

In [89], the regression analysis is used to detect an embedded trojan circuit. The technique plots two parameters, such as current consumption and frequency changes on certain input and output paths on a feature space and uses straight lines based on a 3σ rule as limits, to determine if a trojan has been detected. Another technique which allows separation of two classes in a hyperplane is known as the Support Vector Machine (SVM) [124]. The objective of the SVM is to find a separation function which maximizes the distance between both classes. The SVM technique allows various types of classifiers, including higher number of dimensions (hyperplane). To understand the ability of the SVM technique consider Figure 7.1, where a 2-D space is shown when RMS of the measured voltage and the variance is considered. Two classes can be determined. In the first (red), the watermark signal is not present. In the second (blue), the watermark

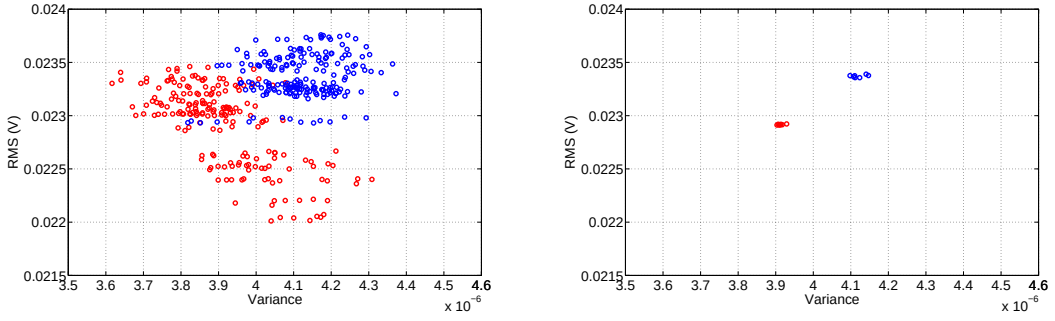


Figure 7.1: 2-D Feature space. Watermark active (blue) and off (red). (a) All repetitions. (b) Repetitions averaged.

circuit is active. As can be seen, after averaging the measured signals (Figure 7.1(b)) to reduce the noise level, both classes can be easily separated and a watermark can be detected. The use of higher dimensions and therefore multiple parameters is a viable solution to achieve a better watermark separation and classification. The area overhead is expected to be lower than demonstrated in Chapter 4, with the CPA technique due to higher sensitivity of SVM, when multiple parameters are used. Nevertheless, due to a constant generation of a watermark power pattern, the power overhead is considerable and robustness of this technique must be compared with the robustness of the technique proposed in Chapter 6.

7.2.3 Classification of Watermark Activation Instructions

In Chapter 6, the watermark circuit was activated with the WFI instruction. The technique demonstrates a negligible hardware implementation costs due to removal of the WPPG circuit, but it is limited to such instruction. The classification technique proposed in Section 7.2.2, enables the watermark circuit to be highly obscured by the system's power consumption but increases the area and power overheads when compared with the technique in Chapter 6. To overcome such limitations, it is necessary to generate a watermark power pattern non-deterministically and for a short period. To achieve this, the activation of a watermark circuit can be triggered by other instructions, such as multiplication. In Chapter 4, Section 4.5.2, the watermark circuit integrated in chip I allows the activation of a watermark circuit with the multiplication instruction executed on ARM® Cortex®-M0 (refer to Appendix B.1). As the multiplication is executed in a single clock cycle and consumes a significant power, the activation of a watermark circuit can be delayed to reduce such effect on detection results. The watermark generates a single period of a watermarking sequence and waits for another multiplication to occur. As in Section 7.2.2, this technique cannot use the clock modulation approach since the processor is active, which would lead to synchronization issues. However,

the power overhead is expected to be negligible, since the watermark circuit is active only for a short period. Nevertheless, due to WPPG circuit the tradeoff between the area overhead, hence the increased visibility in the RTL description, and the robustness against detection in the power consumption occurs, when technique of Chapter 6 is considered and must be investigated.

To detect the watermark power pattern the instruction which causes the activation of a watermark must be detected. The SVM technique can be used to detect the instruction executed by a processor, as demonstrated in Section 7.2.2, to find the starting point of watermark generation. Although, such a technique, if successful, allows the detection of intrinsic processor instructions and watermark implementation may be considered as redundant, the results may be misleading if other processor designs are considered. Therefore, the watermark circuit is necessary.

Appendix A

Pearson Correlation Coefficient Analysis

The Correlation Power Analysis (CPA) used throughout this thesis as the power watermark detection technique considers the Pearson correlation coefficient as in Equation (A.1).

$$\rho = \frac{N\sum_{i=1}^{N} X_i Y_i - \sum_{i=1}^{N} X_i \sum_{i=1}^{N} Y_i}{\sqrt{N\sum_{i=1}^{N} X_i^2 - (\sum_{i=1}^{N} X_i)^2}\sqrt{N\sum_{i=1}^{N} Y_i^2 - (\sum_{i=1}^{N} Y_i)^2}} \tag{A.1}$$

X is the watermark model vector, Y is the power vector and N is the length of both vectors. Vector Y can be considered as the sum of the watermark model, X, and the noise present in the system, β, such as global switching noise of digital IP blocks, environmental and measurement noise. Therefore, vector Y can be represented as

$$Y = X + \beta \tag{A.2}$$

However, since the phase difference between vectors X and Y is not known, vector X is rotated by a single value (clock cycle) and the correlation computation is repeated. Therefore, vector X can be substituted by X', which represents the rotated vector X.

$$X = X' \tag{A.3}$$

By substituting Equations (A.2) and (A.3) into Equation (A.1), the correlation coefficient is as follows

$$\rho = \frac{N\sum_{i=1}^{N} X_i'(X_i+\beta_i) - \sum_{i=1}^{N} X_i' \sum_{i=1}^{N}(X_i+\beta_i)}{\sqrt{N\sum_{i=1}^{N} X_i'^2 - (\sum_{i=1}^{N} X_i')^2}\sqrt{N\sum_{i=1}^{N}(X_i+\beta_i)^2 - (\sum_{i=1}^{N}(X_i+\beta_i))^2}} \tag{A.4}$$

$$\rho = \frac{N\sum_{i=1}^{N}(X_i'X_i + X_i'\beta_i) - \sum_{i=1}^{N} X_i' \sum_{i=1}^{N}(X_i+\beta_i)}{\sqrt{N\sum_{i=1}^{N} X_i'^2 - (\sum_{i=1}^{N} X_i')^2}\sqrt{N\sum_{i=1}^{N}(X_i^2 + 2X_i\beta_i + \beta_i^2) - (\sum_{i=1}^{N}(X_i+\beta_i))^2}} \tag{A.5}$$

Since vectors X and X_i represent the same binary sequence with different phase shifts, the sum of all terms (bits) in a vector equals the Hamming Weight, H, of a sequence. Hence,

$$\sum_{i=1}^{N} X_i = H \tag{A.6}$$

$$\sum_{i=1}^{N} X_i' = H \tag{A.7}$$

$$\sum_{i=1}^{N} X_i^2 = H \tag{A.8}$$

For the ease of reading and conciseness consider

$$\sum_{i=1}^{N} \beta_i = \beta \tag{A.9}$$

Therefore, Equation (A.5) can be represented as

$$\rho = \frac{N\sum_{i=1}^{N}(X_i'X_i) + N\sum_{i=1}^{N}(X_i'\beta_i) - H(H+\beta)}{\sqrt{NH-H^2}\sqrt{N(H + \sum_{i=1}^{N}(2X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - (H+\beta)^2}} \tag{A.10}$$

Solving Equation (A.10) further gives

$$\rho = \frac{N\sum_{i=1}^{N}(X_i'X_i) + N\sum_{i=1}^{N}(X_i'\beta_i) - H^2 - H\beta}{\sqrt{NH-H^2}\sqrt{NH - H^2 + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \beta(2H+\beta)}} \tag{A.11}$$

The dynamic power of a canonical static CMOS gate is linearly proportional to the switching activity, α [97]. However, in digital watermarking, the *activity factor* is intrinsic

to a watermark sequence, and the dynamic power is consumed in clock cycles when such sequence is '1'. Therefore, the Hamming Weight, H, can be substituted as a product of activity α and the length of vectors, N.

$$H = \alpha N \tag{A.12}$$

Substituting Equation (A.12) into Equation (A.11) gives

$$\rho = \frac{N\sum_{i=1}^{N}(X_i'X_i) + N\sum_{i=1}^{N}(X_i'\beta_i) - N^2\alpha^2 - N\alpha\beta}{\sqrt{N^2\alpha - N^2\alpha^2}\sqrt{N^2\alpha - N^2\alpha^2 + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \beta(2N\alpha + \beta)}} \tag{A.13}$$

Furthermore, if both vectors X and X' are in phase

$$\sum_{i=1}^{N} X_i'X_i = \sum_{i=1}^{N} X_i = H \tag{A.14}$$

However, since vectors X' and X can be out of phase, Equation (A.14) is modified by adding the *overlapping factor*, θ as follows

$$\sum_{i=1}^{N} X_i'X_i = \theta H = \theta\alpha N \tag{A.15}$$

Substituting Equation (A.15) into Equation (A.13) gives

$$\rho = \frac{N^2\alpha\theta + N\sum_{i=1}^{N}(X_i'\beta_i) - N^2\alpha^2 - N\alpha\beta}{\sqrt{N^2\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \beta(2N\alpha + \beta)}} \tag{A.16}$$

$$\rho = \frac{N^2\alpha(\theta-\alpha) + N(\sum_{i=1}^{N}(X_i'\beta_i) - \alpha\beta)}{\sqrt{N^2\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \beta(2N\alpha + \beta)}} \tag{A.17}$$

Reducing the common N from Equation (A.17) gives

$$\rho = \frac{N\alpha(\theta-\alpha) + \sum_{i=1}^{N}(X_i'\beta_i) - \alpha\beta}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \beta(2N\alpha + \beta)}} \tag{A.18}$$

Finally, substitute Equation (A.9) into Equation (A.18).

$$\rho = \frac{N\alpha(\theta-\alpha) + \sum_{i=1}^{N}(X_i'\beta_i) - \alpha\sum_{i=1}^{N}\beta_i}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N(2\sum_{i=1}^{N}(X_i\beta_i) + \sum_{i=1}^{N}\beta_i^2) - \sum_{i=1}^{N}\beta_i(2N\alpha + \sum_{i=1}^{N}\beta_i)}} \tag{A.19}$$

The terms $\sum_{i=1}^{N}(X_i'\beta_i)$ and $\sum_{i=1}^{N}(X_i\beta_i)$ in Equation (A.19) depend on the position of '1' in a watermark sequence. However, since $N >> 1$, such terms along with other terms in Equation (A.19) can be simplified and ρ becomes

$$\boldsymbol{\rho = \frac{N\alpha(\theta-\alpha)}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha) + N\sum_{i=1}^{N}\beta_i^2}}, \quad N >> 1} \tag{A.20}$$

In the noiseless environment Equation (A.20) is given by

$$\rho = \frac{N\alpha(\theta-\alpha)}{\sqrt{\alpha(1-\alpha)}\sqrt{N^2\alpha(1-\alpha)}}, \quad N >> 1 \tag{A.21}$$

$$\rho = \frac{N\alpha(\theta-\alpha)}{N\sqrt{\alpha(1-\alpha)}\sqrt{\alpha(1-\alpha)}}, \quad N >> 1 \tag{A.22}$$

$$\rho = \frac{\alpha(\theta-\alpha)}{\alpha(1-\alpha)}, \quad N >> 1 \tag{A.23}$$

Furthermore, if vectors X and X' are in phase, θ is 1 and

$$\rho = \frac{\alpha(1-\alpha)}{\alpha(1-\alpha)} = 1, \quad N >> 1 \tag{A.24}$$

Appendix B

Watermark Circuits Integrated on Test Chips

B.1 Test Chips I and II

The architecture of the watermark circuit embedded on test chips I and II, discussed in Chapter 4 and Chapter 5 is given in this appendix.

B.1.1 Brief Description

The watermark circuit contains: watermark generation circuit (WGC), watermark power pattern generation (WPPG) load circuit, interval counter and watermark controller. Watermark generation circuit generates a watermark sequence which further controls the switching activity of the WPPG circuit. The watermark circuit supports 3 modes of operation, such as *ALWAYS ON*, *INTERVAL* and *M0 TRIGGERED*. In an *ALWAYS ON* mode, the watermark sequence is generated in a continuous fashion. In an *INTERVAL* mode, the delay is introduced between two consecutive watermark sequences. In the *M0 TRIGGERED* mode, the watermark circuit can be triggered with the multiplication instruction executed on ARM® Cortex®-M0.

B.1.2 Interface

The watermark circuit interface is AHB bus compatible. Therefore, the circuit registers have been mapped to specific memory location and can be written to or read from using the AHB bus. The control signals are configured as inputs. However, after the watermark has been integrated as a part of a SoC, the control signals have been mapped

Table B.1: Watermark Circuit Working Registers (Chips I and II)

Address	Description
0xD8000000	WGC - Single Sequence Generator 0
0xD8000004	WGC - Single Sequence Generator 1
0xD8000008	Interval Counter - counter 0 initial register
0xD800000C	Interval Counter - counter 1 initial register
0xD8000010	WPPG - word 0
0xD8000014	WPPG - word 1
0xD8000018	WPPG - word 2
0xD800001C	WPPG - word 3
0xD8000020	WPPG - word 4
...	...
0xD800008C	WPPG - word 31

to the specific memory locations. Therefore, the Ascii Debug Protocol (ADP) interface implemented on a test chip can be used to control the watermark circuit.

The watermark circuit working registers, mapped to a SoC memory, are shown in Table B.1.

The watermark circuit control registers are shown in Table B.2.

B.1.3 Watermark Generation Circuit

Watermark Generation Circuit contains 2 Single Sequence Generators (SSG) to generate a watermark sequence. The SSG contains a 32-bit memory mapped register. Therefore, the initial value is configurable. Furthermore, the SSG can be configured as a circular shift register or Galois LFSR, to generate sequences such as Barker, m-sequence, Kasami and Gold codes. To generate Barker code or m-sequence a single SSG is used. To generate Kasami or Gold codes both SSG must be used and their outputs XORed.

B.1.4 Watermark Power Pattern Generator

The Watermark Power Pattern Generator (WPPG) is used to modulate the strength of the power dissipated due to watermark sequence. The WPPG circuit integrated on both test chips implements a 32 x 32-bit word block, where all registers are mapped to specific memory location. During the shift operation each 32-bit word is shifted to the next word. Since, the shift operation is circular the last word shifts into the 1st word.

Table B.2: Watermark Circuit Working Registers (Chips I and II)

Address	Description
0xD0000000	Watermark Circuit Control Register [1:0] operating mode 01 - always ON 10 - interval 11 - M0 multiplier triggered 4 counter clock enable 5 counter count enable 8 WPPG clock enable 9 WPPG shift enable 10 WPPG shift (1), write (0) 16 WGC clock enable 17 WGC shift (1), write(0) 18 WGC shift enable [23:19] WGC Single Sequence Generator 0 multiplexer control [28:24] WGC Single Sequence Generator 1 multiplexer control 29 WGC output multiplexer control 31 M0 multiplier trigger pin enable (output)
0xD0000004	WGC Taps Control Register 0 [30:0] 1 - indicates the LFSR operation (uses XOR gate) 0 - indicates the shift register operation
0xD0000008	WGC Taps Control Register 1 [30:0] 1 - indicates the LFSR operation (uses XOR gate) 0 - indicates the shift register operation
0xD000000C	Noise Generator Control Register [1:0] operating mode 01 - synchronous operation with WGC and WPPG start time 10 - asynchronous operation 4 Noise Generator clock enable 5 Noise Generator shift (1), write (0) 8 Noise Seed clock enable 9 Noise Seed shift (1), write (0) [14:10] Noise Seed multiplexer control
0xD0000010	Noise Generator Taps Control Register [30:0] 1 - indicates the LFSR operation (uses XOR gate) 0 - indicates the shift register operation

B.1.5 Interval Counter

The interval counter is used during the *Interval* mode. Therefore, the watermark circuit is activated for a single period of a watermark sequence and then it is switched off for a number of clock cycles determined by the interval counter. The interval counter contains two 32-bit count down counters. The first counter determines the length of the switch ON operation, while the second counter determines the number of clock cycles when the watermark circuit is inactive.

B.1.6 Watermark Controller

The watermark controller is a simple state machine and supports three operating modes, such as *ALWAYS ON*, *INTERVAL* and *M0 TRIGGERED*. In the *ALWAYS ON* mode, the state machine activates the WGC and WPPG circuits. Therefore, the watermark sequence is generated continuously, until a reset is asserted. In the *INTERVAL* mode, both counters are pre-loaded with the initial values. The WGC and WPPG are active during the count down of the first counter (Counter 0; the counter must be set to N-1, where N is the period of the watermark sequence). When Counter 0 reaches the value of 0, the WGC and WPPG circuits are disabled. Counter 1 is activated and counts the time to the next generation of the watermark sequence (the counter must be set to M-2, where M is the interval period). Due to the 32-bit delay counter the minimum separation between sequences is 2 clock cycles. The maximum separation can be up to $2^{32}+1$ clock cycles. In the *M0 TRIGGERED* mode, the enable signal from the multiplier circuit embedded in ARM® Cortex®-M0 microprocessor core is connected to the watermark circuit (chip I only). When the enable signal is '1', the multiplication instruction is executed causing a significant power consumption. Such instantaneous power consumption can be used as a trigger signal to start power measurements. The watermark power pattern generation can also be triggered by such signal. To prevent the interference between the multiplication operation and the generated watermark power pattern, the delay is introduced. The minimum delay is 6 clock cycles, while the maximum delay is 2^{32} + 5 clock cycles. Similarly to *INTERVAL* mode, Counter 1 can be used to delay the WGC and WPPG circuits activation after the multiplication has been executed by the core. Furthermore, Counter 0 is used to determine the active time of the WGC and WPPG circuits.

B.1.7 Noise Generator

The noise generator was implemented as a deterministic noise source. It consists of the Noise Seed Generator (NSG) and the Noise Power Pattern Generator (NPPG). The NSG can be regarded as the LFSR circuit, where all registers are additionally connected as

outputs. The architecture of the Noise Power Pattern Generator (NPPG) is similar to the architecture of the WPPG circuit. However, instead of the coarse grain control found in the WPPG circuit, where all words are shifted, the NPPG implements a fine grain control. If one considers the words to create a matrix of 32 rows and 32 columns, the NPPG allows the column-wise rotation of bits, controlled by the bits in the corresponding NSG circuit. For example, if bit 0 in the NSG is '1', all bits in column 0 are rotated.

B.1.8 RTL

B.1.8.1 Top Level Wrapper

```
1  module WATERMARK_TOKACHI_ECS_3 (
2
3  // AHB LITE INTERFACE----------------------------------------------
4  input   wire            HCLK,
5  input   wire            HRESETn,              //Reset (neg active)
6  input   wire    [31:0]  HADDR,                //Address
7  input   wire            HWRITE,               //Tranfer Direction
8  input   wire    [1:0]   HTRANS,               //Transaction Type
9  input   wire    [31:0]  HWDATA,               //Write Data Bus
10 input   wire            HREADY,               //Bus Ready
11 input   wire            HSEL,                 //Slave Select
12 output  wire            HREADYOUT,            //Slave Ready
13 output  wire            HRESP,                //Response
14 output  wire    [31:0]  HRDATA,               //Read Data Bus
15
16 // M0 TRIGGER INPUT-----------------------------------------------
17 input   wire            M0_MULTIPLIER_EN,     //M0 Multiplier Enable signal
18
19 // M0 TRIGGER OUTPUT PIN-----------------------------------------
20 output  wire            M0_TRIGGER_PIN        //M0 trigger signal connected to output (case) pin
21
22 );
23
24 //----------CONTROL BITS----------
25
26 //Watermark Controller
27 wire    [1:0]   WC_OP_MODE;     //Operation Mode
28
29 //Watermark Generation Circuit
30 wire            WGC_CLKEN;              //Clock Enable
31 wire            WGC_SHIFT_WRITE;        //Shift or Write
32 wire            WGC_SHIFT_EN;           //Shift Enable
33 wire    [4:0]   WGC_SSG0_MUX_CTRL;      //SSG0 Mux Control
34 wire    [30:0]  WGC_SSG0_SRL_LFSR;      //SSG0 Shift Register or LFSR control
35 wire    [4:0]   WGC_SSG1_MUX_CTRL;      //SSG1 Mux Control
36 wire    [30:0]  WGC_SSG1_SRL_LFSR;      //SSG1 Shift Register or LFSR control
37 wire            WGC_MUX_CTRL;           //Output MUX Control
38
39 //Leakage Circuit
40 wire            LC_CLKEN;               //Clock Enable
41 wire            LC_SHIFT_EN;            //Shift Enable
42 wire            LC_SHIFT_WRITE;         //Shift (1), Write (0)
43
44 //Interval Counter
45 wire            CNT_CLKEN;              //Clock Enable
46 wire            CNT_CNT_EN;             //Count Enable
47
48 //Noise Generator Circuit
49 wire            NOISE_GEN_CLKEN;                //Noise Generator Clock Enable
50 wire            NOISE_GEN_SHIFT_WRITE;          //Noise Generator Shift (1) or Write (0) mode
51 wire    [1:0]   NOISE_GEN_OP_MODE;              //Noise Generator Operation Mode - Synchronous or Asynchronous to
       watermark circuit
52
53 //Noise Generator Seed
54 wire            NOISE_SEED_CLKEN;               //Noise Shift Register/LFSR Clock Enable
55 wire            NOISE_SEED_SHIFT_WRITE;         //Noise Shift Register/LFSR Shift (1) or Write (0) mode
56 wire    [4:0]   NOISE_SEED_MUX_CTRL;            //Noise Shift Register/LFSR Mux Control - feedback loop
57 wire    [30:0]  NOISE_SEED_SRL_LFSR;            //Noise Shift Register/LFSR Operation Mode (shift register or LFSR
       control bits)
```

```
58
59  //Trigger pin control
60  wire            TRIGGER_PIN_CTRL;
61
62  //----------MEMORY MAP PARAMETERS----------
63  // Operational registers
64  localparam      WGC_SSG0_MAP_ADDR         = 32'hD800_0000;
65  localparam      WGC_SSG1_MAP_ADDR         = 32'hD800_0004;
66
67  localparam      CNT_CNT0_MAP_ADDR         = 32'hD800_0008;
68  localparam      CNT_CNT1_MAP_ADDR         = 32'hD800_000C;
69
70  localparam      LC_MAP_BASE_ADDR          = 32'hD800_0010;
71
72  localparam      NOISE_SEED_MAP_ADDR       = 32'hD800_0100;
73  localparam      NOISE_GEN_MAP_BASE_ADDR   = 32'hD800_0110;
74
75  // Control signals
76  localparam      WM_CTRL_MAP_ADDR          = 32'hD000_0000;
77  localparam      WGC_CTRL0_MAP_ADDR        = 32'hD000_0004;
78  localparam      WGC_CTRL1_MAP_ADDR        = 32'hD000_0008;
79  localparam      NOISE_CTRL_MAP_ADDR       = 32'hD000_000C;
80  localparam      NOISE_GEN_CTRL_MAP_ADDR   = 32'hD000_0010;
81
82
83  //----------AHB SIGNALS----------
84  reg     [31:0]  addr;
85  reg             write_en;
86  reg             read_en;
87  wire            access_en;
88
89  //AHB access transaction start
90  assign access_en = HREADY & HSEL & HTRANS[1];
91
92  //----------AHB DE-PIPELINE----------
93
94          always@(posedge HCLK or negedge HRESETn)
95              if(~HRESETn)
96                  begin
97                    addr[31:0] <= {32{1'b0}};
98                  end
99              else
100                 begin
101                   addr[31:0] <= HADDR[31:0];
102                 end
103
104         always@(posedge HCLK or negedge HRESETn)
105                 if(~HRESETn) begin
106                         write_en <= 1'b0;
107                         read_en <= 1'b0;
108                 end else if(HREADY) begin                       //when bus is ready
109                         write_en <= access_en & HWRITE;         //HWRITE == 1 for writing
110                         read_en <= access_en & ~HWRITE;         //HWRITE == 0 for reading
111                 end
112
113 //----------REGISTER M0 TRIGGER SIGNAL----------
114 reg     m0_trigger_q;
115
116         always@(posedge HCLK or negedge HRESETn)
117             if(~HRESETn)
118                 m0_trigger_q <= 1'b0;
119             else
120                 m0_trigger_q <= M0_MULTIPLIER_EN;
121
122 //----------SELECTION DECODE----------
123
124 wire            wm_ctrl_sel;
125 wire            wgc_ctrl0_sel;
126 wire            wgc_ctrl1_sel;
127 wire            noise_ctrl_sel;
128 wire            noise_gen_ctrl_sel;
129
130 wire            wgc_ssg0_sel;
131 wire            wgc_ssg1_sel;
132
133 wire            lc_sel;
134 reg     [31:0]  lc_sel_dec;
135
136 wire            cnt_cnt0_sel;
137 wire            cnt_cnt1_sel;
```

```
138
139 wire            noise_seed_sel;
140
141 wire            noise_gen_sel;
142 reg     [31:0]  noise_gen_sel_dec;
143
144 assign  wm_ctrl_sel            =       (addr[31:0] == WM_CTRL_MAP_ADDR);
145 assign  wgc_ctrl0_sel          =       (addr[31:0] == WGC_CTRL0_MAP_ADDR);
146 assign  wgc_ctrl1_sel          =       (addr[31:0] == WGC_CTRL1_MAP_ADDR);
147 assign  noise_ctrl_sel         =       (addr[31:0] == NOISE_CTRL_MAP_ADDR);
148 assign  noise_gen_ctrl_sel     =       (addr[31:0] == NOISE_GEN_CTRL_MAP_ADDR);
149
150 assign  wgc_ssg0_sel           =       (addr[31:0] == WGC_SSG0_MAP_ADDR);
151 assign  wgc_ssg1_sel           =       (addr[31:0] == WGC_SSG1_MAP_ADDR);
152
153 assign  cnt_cnt0_sel           =       (addr[31:0] == CNT_CNT0_MAP_ADDR);
154 assign  cnt_cnt1_sel           =       (addr[31:0] == CNT_CNT1_MAP_ADDR);
155
156 assign  lc_sel                 =       (addr[31:8] == LC_MAP_BASE_ADDR[31:8]);
157
158         always@(*)
159                 case({lc_sel,addr[7:0]})
160                         9'h1_10 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_0001;
161                         9'h1_14 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_0010;
162                         9'h1_18 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_0100;
163                         9'h1_1C :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_1000;
164                         9'h1_20 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0001_0000;
165                         9'h1_24 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0010_0000;
166                         9'h1_28 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0100_0000;
167                         9'h1_2C :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_1000_0000;
168                         9'h1_30 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0001_0000_0000;
169                         9'h1_34 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0010_0000_0000;
170                         9'h1_38 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0100_0000_0000;
171                         9'h1_3C :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_1000_0000_0000;
172                         9'h1_40 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0001_0000_0000_0000;
173                         9'h1_44 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0010_0000_0000_0000;
174                         9'h1_48 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0100_0000_0000_0000;
175                         9'h1_4C :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0000_1000_0000_0000_0000;
176                         9'h1_50 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0001_0000_0000_0000_0000;
177                         9'h1_54 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0010_0000_0000_0000_0000;
178                         9'h1_58 :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_0100_0000_0000_0000_0000;
179                         9'h1_5C :       lc_sel_dec[31:0] <= 32'b0000_0000_0000_1000_0000_0000_0000_0000;
180                         9'h1_60 :       lc_sel_dec[31:0] <= 32'b0000_0000_0001_0000_0000_0000_0000_0000;
181                         9'h1_64 :       lc_sel_dec[31:0] <= 32'b0000_0000_0010_0000_0000_0000_0000_0000;
182                         9'h1_68 :       lc_sel_dec[31:0] <= 32'b0000_0000_0100_0000_0000_0000_0000_0000;
183                         9'h1_6C :       lc_sel_dec[31:0] <= 32'b0000_0000_1000_0000_0000_0000_0000_0000;
184                         9'h1_70 :       lc_sel_dec[31:0] <= 32'b0000_0001_0000_0000_0000_0000_0000_0000;
185                         9'h1_74 :       lc_sel_dec[31:0] <= 32'b0000_0010_0000_0000_0000_0000_0000_0000;
186                         9'h1_78 :       lc_sel_dec[31:0] <= 32'b0000_0100_0000_0000_0000_0000_0000_0000;
187                         9'h1_7C :       lc_sel_dec[31:0] <= 32'b0000_1000_0000_0000_0000_0000_0000_0000;
188                         9'h1_80 :       lc_sel_dec[31:0] <= 32'b0001_0000_0000_0000_0000_0000_0000_0000;
189                         9'h1_84 :       lc_sel_dec[31:0] <= 32'b0010_0000_0000_0000_0000_0000_0000_0000;
190                         9'h1_88 :       lc_sel_dec[31:0] <= 32'b0100_0000_0000_0000_0000_0000_0000_0000;
191                         9'h1_8C :       lc_sel_dec[31:0] <= 32'b1000_0000_0000_0000_0000_0000_0000_0000;
192                         default :       lc_sel_dec[31:0] <= 32'h0000_0000;
193                 endcase
194
195 assign  noise_seed_sel = (addr[31:0] == NOISE_SEED_MAP_ADDR);
196
197 assign  noise_gen_sel = (addr[31:8] == NOISE_GEN_MAP_BASE_ADDR[31:8]);
198
199         always@(*)
200                 case({noise_gen_sel,addr[7:0]})
201                         9'h1_10 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_0001;
202                         9'h1_14 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_0010;
203                         9'h1_18 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_0100;
204                         9'h1_1C :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0000_1000;
205                         9'h1_20 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0001_0000;
206                         9'h1_24 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0010_0000;
207                         9'h1_28 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_0100_0000;
208                         9'h1_2C :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0000_1000_0000;
209                         9'h1_30 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0001_0000_0000;
210                         9'h1_34 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0010_0000_0000;
211                         9'h1_38 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_0100_0000_0000;
212                         9'h1_3C :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0000_1000_0000_0000;
213                         9'h1_40 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0001_0000_0000_0000;
214                         9'h1_44 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0010_0000_0000_0000;
215                         9'h1_48 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_0100_0000_0000_0000;
216                         9'h1_4C :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0000_1000_0000_0000_0000;
217                         9'h1_50 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0001_0000_0000_0000_0000;
```

```
218                         9'h1_54 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0010_0000_0000_0000_0000;
219                         9'h1_58 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_0100_0000_0000_0000_0000;
220                         9'h1_5C :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0000_1000_0000_0000_0000_0000;
221                         9'h1_60 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0001_0000_0000_0000_0000_0000;
222                         9'h1_64 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0010_0000_0000_0000_0000_0000;
223                         9'h1_68 :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_0100_0000_0000_0000_0000_0000;
224                         9'h1_6C :       noise_gen_sel_dec[31:0] <= 32'b0000_0000_1000_0000_0000_0000_0000_0000;
225                         9'h1_70 :       noise_gen_sel_dec[31:0] <= 32'b0000_0001_0000_0000_0000_0000_0000_0000;
226                         9'h1_74 :       noise_gen_sel_dec[31:0] <= 32'b0000_0010_0000_0000_0000_0000_0000_0000;
227                         9'h1_78 :       noise_gen_sel_dec[31:0] <= 32'b0000_0100_0000_0000_0000_0000_0000_0000;
228                         9'h1_7C :       noise_gen_sel_dec[31:0] <= 32'b0000_1000_0000_0000_0000_0000_0000_0000;
229                         9'h1_80 :       noise_gen_sel_dec[31:0] <= 32'b0001_0000_0000_0000_0000_0000_0000_0000;
230                         9'h1_84 :       noise_gen_sel_dec[31:0] <= 32'b0010_0000_0000_0000_0000_0000_0000_0000;
231                         9'h1_88 :       noise_gen_sel_dec[31:0] <= 32'b0100_0000_0000_0000_0000_0000_0000_0000;
232                         9'h1_8C :       noise_gen_sel_dec[31:0] <= 32'b1000_0000_0000_0000_0000_0000_0000_0000;
233                         default :       noise_gen_sel_dec[31:0] <= 32'h0000_0000;
234                 endcase
235
236 // CONTROL register programming------------------------------------------------------
237
238 reg     [31:0]  WM_CTRL;
239 reg     [31:0]  WGC0_CTRL;
240 reg     [31:0]  WGC1_CTRL;
241 reg     [31:0]  NOISE_CTRL;
242 reg     [31:0]  NOISE_GEN_CTRL;
243
244 // Control bit allocations ----------
245
246 // Watermark Generation Circuit
247 // byte 0 - mode and counter control
248 assign WC_OP_MODE                       = WM_CTRL[1:0];         //mode
249 assign CNT_CLKEN                        = WM_CTRL[4];           //Clock Enable
250 assign CNT_CNT_EN                       = WM_CTRL[5];           //Count Enable
251
252 // byte 1 - leakage circuit control
253 assign LC_CLKEN                         = WM_CTRL[8];           //Clock Enable
254 assign LC_SHIFT_EN                      = WM_CTRL[9];           //Shift Enable
255 assign LC_SHIFT_WRITE                   = WM_CTRL[10];          //Shift (1), Write (0)
256
257 // byte 2/3 - watermark generator control and trigger pin output control
258 assign WGC_CLKEN                        = WM_CTRL[16];          //Clock Enable
259 assign WGC_SHIFT_WRITE                  = WM_CTRL[17];          //Shift (1), Write (0)
260 assign WGC_SHIFT_EN                     = WM_CTRL[18];          //Shift Enable
261 assign WGC_SSG0_MUX_CTRL[4:0]           = WM_CTRL[23:19];       //SSG0 Mux Control
262 assign WGC_SSG1_MUX_CTRL[4:0]           = WM_CTRL[28:24];       //SSG1 Mux Control
263 assign WGC_MUX_CTRL                     = WM_CTRL[29];          //Output Mux Control
264
265 assign TRIGGER_PIN_CTRL                 = WM_CTRL[31];          //Trigger pin (output control)
266
267 assign WGC_SSG0_SRL_LFSR[30:0]          = WGC0_CTRL[30:0];      //SSG0 Shift Register or LFSR
268 assign WGC_SSG1_SRL_LFSR[30:0]          = WGC1_CTRL[30:0];      //SSG1 Shift Register or LFSR
269
270 // Noise Generation Circuit
271 // byte 0 - noise generator
272 assign NOISE_GEN_OP_MODE                = NOISE_CTRL[1:0];      //mode
273 assign NOISE_GEN_CLKEN                  = NOISE_CTRL[4];        //Clock Enable
274 assign NOISE_GEN_SHIFT_WRITE            = NOISE_CTRL[5];        //Shift (1), Write (0)
275
276 // byte 1 - noise seed
277 assign NOISE_SEED_CLKEN                 = NOISE_CTRL[8];        //Clock Enable
278 assign NOISE_SEED_SHIFT_WRITE           = NOISE_CTRL[9];        //Shift (1), Write (0)
279 assign NOISE_SEED_MUX_CTRL[4:0]         = NOISE_CTRL[14:10];    //Noise random number Mux control
280
281 assign NOISE_SEED_SRL_LFSR[30:0]        = NOISE_GEN_CTRL[30:0]; //Noise random number Shift Register or LFSR
282
283         always@(posedge HCLK or negedge HRESETn)
284             if(~HRESETn)
285                 begin
286                   WM_CTRL           <= {32{1'b0}};
287                   WGC0_CTRL         <= {32{1'b0}};
288                   WGC1_CTRL         <= {32{1'b0}};
289                   NOISE_CTRL        <= {32{1'b0}};
290                   NOISE_GEN_CTRL    <= {32{1'b0}};
291                 end
292             else
293                 begin
294                   WM_CTRL           <= (wm_ctrl_sel & write_en)             ? HWDATA : WM_CTRL;
295                   WGC0_CTRL         <= (wgc_ctrl0_sel & write_en)           ? HWDATA : WGC0_CTRL;
296                   WGC1_CTRL         <= (wgc_ctrl1_sel & write_en)           ? HWDATA : WGC1_CTRL;
297                   NOISE_CTRL        <= (noise_ctrl_sel & write_en)          ? HWDATA : NOISE_CTRL;
```

```
298                 NOISE_GEN_CTRL  <= (noise_gen_ctrl_sel & write_en)      ? HWDATA : NOISE_GEN_CTRL;
299               end
300
301 //----------MODULES INTERCONNECTIONS----------
302 //Watermark controller
303 wire    wc_wgc_shift_en;
304 wire    wc_lc_shift_en;
305 wire    wc_ic_cnt_ld_ini;
306 wire    wc_ic_cnt0_en;
307 wire    wc_ic_cnt1_en;
308
309 //Watermark Generation Circuit
310 wire            wgc_lc_enable;          //Output signal from WGC connected to LC for LC modulation
311 wire    [31:0]  wgc_q_word;                     //WGC Muxed Registers Output, 32-bit Word
312
313 //Leakage Circuit
314 wire    [31:0]  lc_q_word;                      //Leakage Circuit Muxed Registers Output, 32-bit Word
315
316 //Interval Counter
317 wire            ic_cnt0_zero_flag;
318 wire            ic_cnt1_zero_flag;
319 wire    [31:0]  ic_qini_word;                   //Interval Counter Muxed Initial Register Output, 32-bit Word
320
321 //Noise Generator
322 wire    [31:0]  noise_gen_q;
323
324 //Noise Seed (random number generator or shift register)
325 wire    [31:0]  noise_seed_q;
326
327 //Noise Controller
328 wire            noise_gen_shift_en_wire;
329 wire            noise_seed_shift_en_wire;
330
331 //----------MODULES INSTANTATIONS----------
332
333 WATERMARK_CONTROLLER    wat_ctrl        (.CLK(HCLK),
334                                          .RESETn(HRESETn),
335                                          .OP_MODE(WC_OP_MODE[1:0]),
336                                          .WGC_SHIFT_EN(WGC_SHIFT_EN),
337                                          .LC_SHIFT_EN(LC_SHIFT_EN),
338                                          .IC_CNT_EN(CNT_CNT_EN),
339                                          .M0_TRIG_EN(m0_trigger_q),
340                                          .IC_CNT0_ZERO_FLAG(ic_cnt0_zero_flag),
341                                          .IC_CNT1_ZERO_FLAG(ic_cnt1_zero_flag),
342                                          .WGC_SHIFT_ENO(wc_wgc_shift_en),
343                                          .LC_SHIFT_ENO(wc_lc_shift_en),
344                                          .IC_CNT_LD_INI(wc_ic_cnt_ld_ini),
345                                          .IC_CNT0_ENO(wc_ic_cnt0_en),
346                                          .IC_CNT1_ENO(wc_ic_cnt1_en));
347
348 WGC     wat_gen_circ    (
349                                  .CLK(HCLK),
350                                  .CLKEN(WGC_CLKEN),
351                                  .SHIFT_WRITE(WGC_SHIFT_WRITE),
352                                  .SHIFT_EN(wc_wgc_shift_en),
353                                  .SSG0_SEL(wgc_ssg0_sel),
354                                  .SSG1_SEL(wgc_ssg1_sel),
355                                  .WRITE_EN(write_en),
356                                  .SSG0_MUX_CTRL(WGC_SSG0_MUX_CTRL[4:0]),
357                                  .SSG0_SRL_LFSR(WGC_SSG0_SRL_LFSR[30:0]),
358                                  .SSG1_MUX_CTRL(WGC_SSG1_MUX_CTRL[4:0]),
359                                  .SSG1_SRL_LFSR(WGC_SSG1_SRL_LFSR[30:0]),
360                                  .D(HWDATA[31:0]),
361                                  .MUX_CTRL(WGC_MUX_CTRL),
362                                  .WGCO(wgc_lc_enable),
363                                  .Q(wgc_q_word[31:0])
364                          );
365
366 LEAKAGE_CIRCUIT lc      (
367                                  .CLK(HCLK),
368                                  .CLKEN(LC_CLKEN),
369                                  .SHIFT_EN(wc_lc_shift_en),
370                                  .SHIFT_CTRL(wgc_lc_enable),
371                                  .SHIFT_WRITE(LC_SHIFT_WRITE),
372                                  .WRITE_EN(write_en),
373                                  .SEL_DEC(lc_sel_dec[31:0]),
374                                  .D(HWDATA[31:0]),
375                                  .Q(lc_q_word[31:0])
376                          );
377
```

```
378 INTERVAL_COUNTER        int_counter     (
379                                             .CLK(HCLK),
380                                             .CLKEN(CNT_CLKEN),
381                                             .RESETn(HRESETn),
382                                             .CNT_LD_INI(wc_ic_cnt_ld_ini),
383                                             .CNT0_CNT_EN(wc_ic_cnt0_en),
384                                             .CNT1_CNT_EN(wc_ic_cnt1_en),
385                                             .CNT0_SEL(cnt_cnt0_sel),
386                                             .CNT1_SEL(cnt_cnt1_sel),
387                                             .WRITE_EN(write_en),
388                                             .D(HWDATA[31:0]),
389                                             .CNT0_ZERO_FLAG(ic_cnt0_zero_flag),
390                                             .CNT1_ZERO_FLAG(ic_cnt1_zero_flag),
391                                             .Q_INI(ic_qini_word[31:0])
392                                         );
393
394 //Noise Generator
395 NOISE_GEN       noise_generator         (
396                                             .CLK(HCLK),
397                                             .CLKEN(NOISE_GEN_CLKEN),
398                                             .SHIFT_EN(noise_gen_shift_en_wire),
399                                             .SHIFT_CTRL(noise_seed_q),
400                                             .SHIFT_WRITE(NOISE_GEN_SHIFT_WRITE),
401                                             .WRITE_EN(write_en),
402                                             .SEL_DEC(noise_gen_sel_dec),
403                                             .D(HWDATA),
404                                             .Q(noise_gen_q)
405                                         );
406
407 //Noise Seed
408 SINGLE_SEQ_GEN  noise_seed              (
409                                             .CLK(HCLK),
410                                             .CLKEN(NOISE_SEED_CLKEN),
411                                             .SHIFT_WRITE(NOISE_SEED_SHIFT_WRITE),
412                                             .SHIFT_EN(noise_seed_shift_en_wire),
413                                             .SEL(noise_seed_sel),
414                                             .WRITE_EN(write_en),
415                                             .MUX_CTRL(NOISE_SEED_MUX_CTRL),
416                                             .D(HWDATA),
417                                             .SRL_LFSR(NOISE_SEED_SRL_LFSR),
418                                             .SSGO(),                                    //left unconnected
      intentionally
419                                             .Q(noise_seed_q)
420                                         );
421
422 //Noise Generator Controller
423 NOISE_GEN_CONTROLLER    noise_control   (
424                                             .CLK(HCLK),
425                                             .RESETn(HRESETn),
426                                             .NOISE_GEN_OP_MODE(NOISE_GEN_OP_MODE),
427                                             .WC_WGC_SHIFT_EN(wc_wgc_shift_en),
428                                             .NOISE_GEN_SHIFT_ENO(noise_gen_shift_en_wire),
429                                             .NOISE_SEED_SHIFT_ENO(noise_seed_shift_en_wire)
430                                         );
431
432 //----------AHB BUS READ MUX----------
433 wire    [31:0]  bus_read_mux;
434
435 assign bus_read_mux[31:0] = ({32{wgc_ssg0_sel | wgc_ssg1_sel}} & wgc_q_word[31:0]) |
436                             ({32{cnt_cnt0_sel | cnt_cnt1_sel}} & ic_qini_word[31:0]) |
437                             ({32{lc_sel}} & lc_q_word[31:0]) |
438                             ({32{noise_seed_sel}} & noise_seed_q[31:0]) |
439                             ({32{noise_gen_sel}} & noise_gen_q[31:0]);
440
441 // Output data
442
443 assign HRDATA[31:0] = (read_en) ? bus_read_mux[31:0] : 32'h0000_0000;
444 assign HREADYOUT = 1'b1;                                        //zero wait-state
445 assign HRESP = 1'b0;                                            //no errors
446
447 assign M0_TRIGGER_PIN = TRIGGER_PIN_CTRL & m0_trigger_q;        //Trigger pin assignment, routed when control is 1
448
449 endmodule
```

B.1.8.2 Watermark Generation Circuit

```
1  //*****************************************************************************
2  //WATERMARK GENERATION CIRCUIT MODULE
3  //*****************************************************************************
4  //Instantiates 2 Single Sequence Generator (SSG) Modules and the 2-to-1 MUX
5  //In case of Barker code the SSG0 is implemented as as shift register and the output MUX
6  //routes the output from the last used (set) register
7  //In case of M-sequence the SSG0 is implemented as a LFSR (Galois) and the output MUX
8  //routes the output from the last used (set) register
9  //In case of the Gold and Kasami codes for SSG0 and SSG1 are implemented as LFSRs
10 //with variable length (for Gold both LFSRs have the same length, for Kasami both
11 //have different lengths); the output MUX routes the output from the XOR gate
12 //which XORs the output of both SSGs.
13 //
14 //Most of the control bits are like in SSG module
15 //The additional control bit is WGC_MUX_CTRL which controls if SSG0 or SSG0 ^ SSG1 is
16 //routed to the output of the WGC (this will serve as the control line (enable) for the
17 //leakage circuit
18 //*****************************************************************************
19 module WGC (
20
21 input   wire            CLK,
22 input   wire            CLKEN,          //Clock Enable
23 input   wire            SHIFT_WRITE,    //Watermark Generation Circuit Shift or Write
24 input   wire            SHIFT_EN,       //WGC Shift Enable
25 input   wire            SSG0_SEL,       //SSG0 Select
26 input   wire            SSG1_SEL,       //SSG1 Select
27 input   wire            WRITE_EN,       //SSG Write Enable
28 input   wire    [4:0]   SSG0_MUX_CTRL,  //SSG0 Mux Control
29 input   wire    [30:0]  SSG0_SRL_LFSR,  //SSG0 Shift Register or LFSR control
30 input   wire    [4:0]   SSG1_MUX_CTRL,  //SSG1 Mux Control
31 input   wire    [30:0]  SSG1_SRL_LFSR,  //SSG1 Shift Register or LFSR control
32 input   wire    [31:0]  D,              //Input Data
33 input   wire            MUX_CTRL,       //Output MUX Control
34
35 output  wire            WGCO,           //WGC Output
36 output  wire    [31:0]  Q               //Output Data
37 );
38
39 //INSTANTIATE SINGLE SEQUENCE GENERATORS
40 wire            ssg0_out;       //single bit data from feedback loop
41 wire            ssg1_out;
42 wire    [31:0]  ssg0_q;         //32-bit data from register
43 wire    [31:0]  ssg1_q;
44
45 SINGLE_SEQ_GEN  ssg0    (
46                                 .CLK(CLK),
47                                 .CLKEN(CLKEN),
48                                 .SHIFT_WRITE(SHIFT_WRITE),
49                                 .SHIFT_EN(SHIFT_EN),
50                                 .SEL(SSG0_SEL),
51                                 .WRITE_EN(WRITE_EN),
52                                 .MUX_CTRL(SSG0_MUX_CTRL),
53                                 .D(D),.SRL_LFSR(SSG0_SRL_LFSR),
54                                 .SSGO(ssg0_out),
55                                 .Q(ssg0_q[31:0])
56                         );
57
58 SINGLE_SEQ_GEN  ssg1    (
59                                 .CLK(CLK),
60                                 .CLKEN(CLKEN),
61                                 .SHIFT_WRITE(SHIFT_WRITE),
62                                 .SHIFT_EN(SHIFT_EN),
63                                 .SEL(SSG1_SEL),
64                                 .WRITE_EN(WRITE_EN),
65                                 .MUX_CTRL(SSG1_MUX_CTRL),
66                                 .D(D),
67                                 .SRL_LFSR(SSG1_SRL_LFSR),
68                                 .SSGO(ssg1_out),
69                                 .Q(ssg1_q[31:0])
70                         );
71
72 //---
73
74 //WGC WATERMARK SEQUENCE OUTPUT MUX
75 //For Barker and M-sequence only 1 SSG is required
76 //For Gold and Kasami 2 SSGs must be XORed
77
```

```
78  assign WCCO = (MUX_CTRL) ? (ssg0_out ^ ssg1_out) : ssg0_out;
79
80  //OUTPUT DATA MUX
81  wire     [31:0]  q_mux;
82
83  assign q_mux[31:0] = ({32{SSG0_SEL}} & ssg0_q[31:0]) |
84                       ({32{SSG1_SEL}} & ssg1_q[31:0]);
85
86  assign Q[31:0] = q_mux[31:0];
87
88  endmodule
89
90
91  //*****************************************************************************
92  //SINGLE SEQUENCE GENERATOR MODULE
93  //*****************************************************************************
94  //Can be set to implement LFSR, up to 32-bit (Galois LFSR) or up to 32-bit Shift Register
95  //SHIFT_WRITE control bit determines if SSG is operating as SRL/LFSR or reading in DATA from
96  //the BUS (the initial DATA)
97  //Since all active registers will read in the DATA from the BUS all have the same WRITE_EN
98  //In order to choose the feedback loop for the LFSR (and the output) and the output for
99  //the SRL, the 32-to-1 MUX is used with MUX_CTRL control bits
100 //In order to implement the LFSR, the feedback is XORed with registers depending on the
101 //LFSR length to be implemented, therefore 31 SRL_LFSR control bits are required
102 //*****************************************************************************
103
104 `timescale 1ns / 1ps
105
106 module SINGLE_SEQ_GEN (
107
108 input   wire            CLK,
109 input   wire            CLKEN,          //Clock enable
110 input   wire            SHIFT_WRITE,    //Shift (1) or Write (0)
111 input   wire            SHIFT_EN,       //Shift Enable
112 input   wire            SEL,            //Select
113 input   wire            WRITE_EN,       //Write Enable
114 input   wire    [4:0]   MUX_CTRL,       //Mux 32-to-1 Control bits
115 input   wire    [31:0]  D,              //Input Data
116 input   wire    [31:1]  SRL_LFSR,       //SRL_LFSR Control bits
117
118 output  wire            SSGO,           //Single Sequence Generator Output
119 output  wire    [31:0]  Q               //Output Data
120 );
121
122 //Single Sequence Generator Output (Q) Bits
123 wire    ssgq0;
124 wire    ssgq1;
125 wire    ssgq2;
126 wire    ssgq3;
127 wire    ssgq4;
128 wire    ssgq5;
129 wire    ssgq6;
130 wire    ssgq7;
131 wire    ssgq8;
132 wire    ssgq9;
133 wire    ssgq10;
134 wire    ssgq11;
135 wire    ssgq12;
136 wire    ssgq13;
137 wire    ssgq14;
138 wire    ssgq15;
139 wire    ssgq16;
140 wire    ssgq17;
141 wire    ssgq18;
142 wire    ssgq19;
143 wire    ssgq20;
144 wire    ssgq21;
145 wire    ssgq22;
146 wire    ssgq23;
147 wire    ssgq24;
148 wire    ssgq25;
149 wire    ssgq26;
150 wire    ssgq27;
151 wire    ssgq28;
152 wire    ssgq29;
153 wire    ssgq30;
154 wire    ssgq31;
155
156 //Output from 32-to-1 MUX, fed back to the 1st register and all feedback wires to XNOR gates
157 wire    feedback_loop;
```

```
158
159 //REGISTER BANK INSTANTIATION
160 SEQ_GEN_BIT_WITHOUT_XOR reg0 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(SEL),.
        WRITE_EN(WRITE_EN),.D(D[0]),.D_Q(feedback_loop),.Q(ssgq0));
161
162 SEQ_GEN_BIT_WITH_XOR reg1 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[1]),.D_Q(ssgq0),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[1]),.Q(ssgq1));
163
164 SEQ_GEN_BIT_WITH_XOR reg2 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[2]),.D_Q(ssgq1),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[2]),.Q(ssgq2));
165
166 SEQ_GEN_BIT_WITH_XOR reg3 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[3]),.D_Q(ssgq2),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[3]),.Q(ssgq3));
167
168 SEQ_GEN_BIT_WITH_XOR reg4 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[4]),.D_Q(ssgq3),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[4]),.Q(ssgq4));
169
170 SEQ_GEN_BIT_WITH_XOR reg5 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[5]),.D_Q(ssgq4),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[5]),.Q(ssgq5));
171
172 SEQ_GEN_BIT_WITH_XOR reg6 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[6]),.D_Q(ssgq5),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[6]),.Q(ssgq6));
173
174 SEQ_GEN_BIT_WITH_XOR reg7 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[7]),.D_Q(ssgq6),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[7]),.Q(ssgq7));
175
176 SEQ_GEN_BIT_WITH_XOR reg8 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[8]),.D_Q(ssgq7),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[8]),.Q(ssgq8));
177
178 SEQ_GEN_BIT_WITH_XOR reg9 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[9]),.D_Q(ssgq8),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[9]),.Q(ssgq9));
179
180 SEQ_GEN_BIT_WITH_XOR reg10 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[10]),.D_Q(ssgq9),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[10]),.Q(ssgq10));
181
182 SEQ_GEN_BIT_WITH_XOR reg11 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[11]),.D_Q(ssgq10),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[11]),.Q(ssgq11));
183
184 SEQ_GEN_BIT_WITH_XOR reg12 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[12]),.D_Q(ssgq11),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[12]),.Q(ssgq12));
185
186 SEQ_GEN_BIT_WITH_XOR reg13 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[13]),.D_Q(ssgq12),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[13]),.Q(ssgq13));
187
188 SEQ_GEN_BIT_WITH_XOR reg14 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[14]),.D_Q(ssgq13),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[14]),.Q(ssgq14));
189
190 SEQ_GEN_BIT_WITH_XOR reg15 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[15]),.D_Q(ssgq14),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[15]),.Q(ssgq15));
191
192 SEQ_GEN_BIT_WITH_XOR reg16 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[16]),.D_Q(ssgq15),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[16]),.Q(ssgq16));
193
194 SEQ_GEN_BIT_WITH_XOR reg17 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[17]),.D_Q(ssgq16),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[17]),.Q(ssgq17));
195
196 SEQ_GEN_BIT_WITH_XOR reg18 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[18]),.D_Q(ssgq17),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[18]),.Q(ssgq18));
197
198 SEQ_GEN_BIT_WITH_XOR reg19 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[19]),.D_Q(ssgq18),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[19]),.Q(ssgq19));
199
200 SEQ_GEN_BIT_WITH_XOR reg20 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[20]),.D_Q(ssgq19),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[20]),.Q(ssgq20));
201
202 SEQ_GEN_BIT_WITH_XOR reg21 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[21]),.D_Q(ssgq20),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[21]),.Q(ssgq21));
203
204 SEQ_GEN_BIT_WITH_XOR reg22 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[22]),.D_Q(ssgq21),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[22]),.Q(ssgq22));
205
206 SEQ_GEN_BIT_WITH_XOR reg23 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[23]),.D_Q(ssgq22),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[23]),.Q(ssgq23));
207
208 SEQ_GEN_BIT_WITH_XOR reg24 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[24]),.D_Q(ssgq23),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[24]),.Q(ssgq24));
209
210 SEQ_GEN_BIT_WITH_XOR reg25 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[25]),.D_Q(ssgq24),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[25]),.Q(ssgq25));
211
```

```
212 SEQ_GEN_BIT_WITH_XOR                reg26 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[26]),.D_Q(ssgq25),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[26]),.Q(ssgq26));
213
214 SEQ_GEN_BIT_WITH_XOR                reg27 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[27]),.D_Q(ssgq26),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[27]),.Q(ssgq27));
215
216 SEQ_GEN_BIT_WITH_XOR                reg28 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[28]),.D_Q(ssgq27),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[28]),.Q(ssgq28));
217
218 SEQ_GEN_BIT_WITH_XOR                reg29 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[29]),.D_Q(ssgq28),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[29]),.Q(ssgq29));
219
220 SEQ_GEN_BIT_WITH_XOR                reg30 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[30]),.D_Q(ssgq29),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[30]),.Q(ssgq30));
221
222 SEQ_GEN_BIT_WITH_XOR                reg31 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_WRITE(SHIFT_WRITE),.SHIFT_EN(SHIFT_EN),.SEL(
        SEL),.WRITE_EN(WRITE_EN),.D(D[31]),.D_Q(ssgq30),.FEEDBACK(feedback_loop),.SRL_LFSR(SRL_LFSR[31]),.Q(ssgq31));
223 //---
224
225
226 //FEEDBACK MUX INSTANTATION
227 wire [31:0] reg_bank_q;
228
229 assign reg_bank_q[31:0] = {ssgq31,
230                            ssgq30,
231                            ssgq29,
232                            ssgq28,
233                            ssgq27,
234                            ssgq26,
235                            ssgq25,
236                            ssgq24,
237                            ssgq23,
238                            ssgq22,
239                            ssgq21,
240                            ssgq20,
241                            ssgq19,
242                            ssgq18,
243                            ssgq17,
244                            ssgq16,
245                            ssgq15,
246                            ssgq14,
247                            ssgq13,
248                            ssgq12,
249                            ssgq11,
250                            ssgq10,
251                            ssgq9,
252                            ssgq8,
253                            ssgq7,
254                            ssgq6,
255                            ssgq5,
256                            ssgq4,
257                            ssgq3,
258                            ssgq2,
259                            ssgq1,
260                            ssgq0
261                          };
262
263 MUX32   ssg_mux (.D(reg_bank_q[31:0]),.CTRL(MUX_CTRL),.Q(feedback_loop));
264
265 //OUTPUT DATA
266 assign Q[31:0] = reg_bank_q[31:0];
267 assign SSGO = feedback_loop;
268
269 endmodule
270
271
272 //*********************************************************************************
273 //1-bit REG with XOR gate
274 //*********************************************************************************
275 //Used to implement SRL/LFSR function (XOR gates used for flip-flops from 1st to 31st for LFSR)
276 //Also additional MUX is used to choose the data either from the BUS (in case of memory mapping)
277 //or from the previous REG
278 //The clock can be disabled by setting CLK_EN to 0
279 //
280 //The input DATA can be routed either from the BUS or from the previous REG
281 //When initial DATA comes from the BUS the D input is used and SHIFT_WRITE is set to 0,
282 //also WRITE_EN is set to 1 and SEL must be 1
283 //If DATA comes from the previous REG the SHIFT_WRITE is set to 1
284 //WRITE_EN and SEL are not important
285 //
```

```
286 //Since the whole 32-bit REG can be implemented as either SRL (shift register) or LFSR the
287 //additional control bit SRL_LFSR specifies which mode the WORD is currently in (1 for SRL,
288 //0 for LFSR). In case of SRL mode the data is shifted from the previous REG, while in case
289 //of the LFSR the data from the previous REG is XORed with the bit from the last usable REG
290 //(feedback loop)
291 //*********************************************************************************
292
293 `timescale 1ns / 1ps
294
295 module SEQ_GEN_BIT_WITH_XOR (
296
297 input   wire    CLK,
298 input   wire    CLKEN,          //Clock enable
299 input   wire    SHIFT_WRITE,    //Shift (1) or Write (0)
300 input   wire    SHIFT_EN,       //Shift Enable
301 input   wire    SEL,            //Select
302 input   wire    WRITE_EN,       //Write Enable
303 input   wire    D,              //Input Data
304 input   wire    D_Q,            //Data from previous REG
305 input   wire    FEEDBACK,       //Data from last REG
306 input   wire    SRL_LFSR,       //Connected as shift register (SRL - 0) or LFSR (1);
307 output  wire    Q               //Data Out
308
309 );
310
311
312 wire    d_srl_lfsr; //data connected directly from previous Q (SRL) or through XOR (LFSR)
313
314 assign d_srl_lfsr = (SRL_LFSR) ? (D_Q ^ FEEDBACK) : D_Q;
315
316 wire    d_sel; //data connected from the bus or D_SRL_LFSR
317
318 assign d_sel = (SHIFT_WRITE) ? d_srl_lfsr : D;
319
320 reg     q_reg;
321
322         always@(posedge CLK)
323           if(CLKEN)
324             if((SHIFT_WRITE & SHIFT_EN) | (SEL & WRITE_EN & ~SHIFT_WRITE)) q_reg <= d_sel;
325
326 assign Q = q_reg;
327
328 endmodule
329
330
331 //*********************************************************************************
332 //1-bit REG without XOR gate
333 //*********************************************************************************
334 //Used to implement SRL/LFSR function (XOR gate not needed as this would be 1st REG for LFSR)
335 //Also additional MUX is used to choose the data either from the BUS (in case of memory mapping)
336 //or from the previous REG
337 //The clock can be disabled by setting CLK_EN to 0
338 //
339 //The input DATA can be routed either from the BUS or from the previous REG
340 //When initial DATA comes from the BUS the D input is used and SHIFT_WRITE is set to 0,
341 //also WRITE_EN is set to 1 and SEL must be 1
342 //If DATA comes from the previous REG the SHIFT_WRITE is set to 1
343 //WRITE_EN and SEL are not important
344 //*********************************************************************************
345
346 `timescale 1ns / 1ps
347
348 module SEQ_GEN_BIT_WITHOUT_XOR (
349
350 input   wire    CLK,
351 input   wire    CLKEN,          //Clock enable
352 input   wire    SHIFT_WRITE,    //Shift (1) or Write (0)
353 input   wire    SHIFT_EN,       //Shift Enable
354 input   wire    SEL,            //Select
355 input   wire    WRITE_EN,       //Write Enable
356 input   wire    D,              //Input Data
357 input   wire    D_Q,            //Data from previous REG
358 output  wire    Q               //Data Out
359
360 );
361
362
363 wire    d_sel; //data connected from the bus or data input
364
365 assign d_sel = (SHIFT_WRITE) ? D_Q : D;
```

```
366
367 reg       q_reg;
368
369         always@(posedge CLK)
370           if(CLKEN)
371             if((SHIFT_WRITE & SHIFT_EN) | (SEL & WRITE_EN & ~SHIFT_WRITE)) q_reg <= d_sel;
372
373 assign Q = q_reg;
374
375 endmodule
376
377
378 //*****************************************************************************************
379 //32-to-1 MUX
380 //*****************************************************************************************
381
382 `timescale 1ns / 1ps
383
384 module MUX32 (
385
386 input   wire    [31:0]  D,      //Input
387 input   wire    [4:0]   CTRL,   //Control
388 output  wire            Q       //Output
389 );
390
391 assign Q = ( (CTRL == 5'b0_0000) & D[0]) |
392                 ( (CTRL == 5'b0_0001) & D[1]) |
393                 ( (CTRL == 5'b0_0010) & D[2]) |
394                 ( (CTRL == 5'b0_0011) & D[3]) |
395                 ( (CTRL == 5'b0_0100) & D[4]) |
396                 ( (CTRL == 5'b0_0101) & D[5]) |
397                 ( (CTRL == 5'b0_0110) & D[6]) |
398                 ( (CTRL == 5'b0_0111) & D[7]) |
399                 ( (CTRL == 5'b0_1000) & D[8]) |
400                 ( (CTRL == 5'b0_1001) & D[9]) |
401                 ( (CTRL == 5'b0_1010) & D[10]) |
402                 ( (CTRL == 5'b0_1011) & D[11]) |
403                 ( (CTRL == 5'b0_1100) & D[12]) |
404                 ( (CTRL == 5'b0_1101) & D[13]) |
405                 ( (CTRL == 5'b0_1110) & D[14]) |
406                 ( (CTRL == 5'b0_1111) & D[15]) |
407                 ( (CTRL == 5'b1_0000) & D[16]) |
408                 ( (CTRL == 5'b1_0001) & D[17]) |
409                 ( (CTRL == 5'b1_0010) & D[18]) |
410                 ( (CTRL == 5'b1_0011) & D[19]) |
411                 ( (CTRL == 5'b1_0100) & D[20]) |
412                 ( (CTRL == 5'b1_0101) & D[21]) |
413                 ( (CTRL == 5'b1_0110) & D[22]) |
414                 ( (CTRL == 5'b1_0111) & D[23]) |
415                 ( (CTRL == 5'b1_1000) & D[24]) |
416                 ( (CTRL == 5'b1_1001) & D[25]) |
417                 ( (CTRL == 5'b1_1010) & D[26]) |
418                 ( (CTRL == 5'b1_1011) & D[27]) |
419                 ( (CTRL == 5'b1_1100) & D[28]) |
420                 ( (CTRL == 5'b1_1101) & D[29]) |
421                 ( (CTRL == 5'b1_1110) & D[30]) |
422                 ( (CTRL == 5'b1_1111) & D[31]);
423
424 endmodule
```

B.1.8.3 Watermark Power Pattern Generator

```
1  //*****************************************************************************************
2  //LEAKAGE CIRCUIT MODULE
3  //*****************************************************************************************
4  //Implements a big dummy shift register (32 x 32-bit WORD), shifted in a WORD fashion
5  //
6  //In case of reading from the BUS, the appropriate WORD is selected by decoding the SEL_ADDR
7  //input. WRITE_EN is 1 and SHIFT_WRITE is 0.
8  //
9  //In case of operating as a shift register SHIFT_WRITE is 1 and the shifting is controlled by
10 //SHIFT_EN which in turn will be connected to the output of the Watermark Generation Circuit,
11 //SEL_ADDR and WRITE_EN are not important
12 //*****************************************************************************************
13
14 `timescale 1ns / 1ps
15
16 module LEAKAGE_CIRCUIT (
17
18 input   wire            CLK,
19 input   wire            CLKEN,          //Clock Enable
20 input   wire            SHIFT_EN,       //Shift Enable
21 input   wire            SHIFT_CTRL,     //Shift Control
22 input   wire            SHIFT_WRITE,    //Shift (1), Write (0)
23 input   wire            WRITE_EN,       //Write Enable
24 input   wire    [31:0]  SEL_DEC,        //Selection Decode
25 input   wire    [31:0]  D,              //Input Data
26
27 output  wire    [31:0]  Q               //Output Data
28 );
29
30 //lc_word Outputs
31 wire    [31:0]  lc_word [0:31];
32
33 //LEAKAGE CIRCUIT INSTANTATION - 32 X 32-bit Leakage Circuit
34
35 LEAKAGE_WORD lc_word0 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(SHIFT_WRITE
       ),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[0]),.D(D),.D_Q(lc_word[31]),.Q(lc_word[0][31:0]));
36
37 LEAKAGE_WORD lc_word1 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(SHIFT_WRITE
       ),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[1]),.D(D),.D_Q(lc_word[0]),.Q(lc_word[1][31:0]));
38
39 LEAKAGE_WORD lc_word2 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(SHIFT_WRITE
       ),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[2]),.D(D),.D_Q(lc_word[1]),.Q(lc_word[2][31:0]));
40
41 LEAKAGE_WORD lc_word3   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[3]),.D(D),.D_Q(lc_word[2]),.Q(lc_word[3][31:0]));
42
43 LEAKAGE_WORD lc_word4   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[4]),.D(D),.D_Q(lc_word[3]),.Q(lc_word[4][31:0]));
44
45 LEAKAGE_WORD lc_word5   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[5]),.D(D),.D_Q(lc_word[4]),.Q(lc_word[5][31:0]));
46
47 LEAKAGE_WORD lc_word6   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[6]),.D(D),.D_Q(lc_word[5]),.Q(lc_word[6][31:0]));
48
49 LEAKAGE_WORD lc_word7   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[7]),.D(D),.D_Q(lc_word[6]),.Q(lc_word[7][31:0]));
50
51 LEAKAGE_WORD lc_word8   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[8]),.D(D),.D_Q(lc_word[7]),.Q(lc_word[8][31:0]));
52
53 LEAKAGE_WORD lc_word9   (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[9]),.D(D),.D_Q(lc_word[8]),.Q(lc_word[9][31:0]));
54
55 LEAKAGE_WORD lc_word10  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[10]),.D(D),.D_Q(lc_word[9]),.Q(lc_word[10][31:0]));
56
57 LEAKAGE_WORD lc_word11  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[11]),.D(D),.D_Q(lc_word[10]),.Q(lc_word[11][31:0]));
58
59 LEAKAGE_WORD lc_word12  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[12]),.D(D),.D_Q(lc_word[11]),.Q(lc_word[12][31:0]));
60
61 LEAKAGE_WORD lc_word13  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[13]),.D(D),.D_Q(lc_word[12]),.Q(lc_word[13][31:0]));
62
```

```
63  LEAKAGE_WORD lc_word14  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[14]),.D(D),.D_Q(lc_word[13]),.Q(lc_word[14][31:0]));
64
65  LEAKAGE_WORD lc_word15  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[15]),.D(D),.D_Q(lc_word[14]),.Q(lc_word[15][31:0]));
66
67  LEAKAGE_WORD lc_word16  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[16]),.D(D),.D_Q(lc_word[15]),.Q(lc_word[16][31:0]));
68
69  LEAKAGE_WORD lc_word17  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[17]),.D(D),.D_Q(lc_word[16]),.Q(lc_word[17][31:0]));
70
71  LEAKAGE_WORD lc_word18  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[18]),.D(D),.D_Q(lc_word[17]),.Q(lc_word[18][31:0]));
72
73  LEAKAGE_WORD lc_word19  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[19]),.D(D),.D_Q(lc_word[18]),.Q(lc_word[19][31:0]));
74
75  LEAKAGE_WORD lc_word20  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[20]),.D(D),.D_Q(lc_word[19]),.Q(lc_word[20][31:0]));
76
77  LEAKAGE_WORD lc_word21  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[21]),.D(D),.D_Q(lc_word[20]),.Q(lc_word[21][31:0]));
78
79  LEAKAGE_WORD lc_word22  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[22]),.D(D),.D_Q(lc_word[21]),.Q(lc_word[22][31:0]));
80
81  LEAKAGE_WORD lc_word23  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[23]),.D(D),.D_Q(lc_word[22]),.Q(lc_word[23][31:0]));
82
83  LEAKAGE_WORD lc_word24  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[24]),.D(D),.D_Q(lc_word[23]),.Q(lc_word[24][31:0]));
84
85  LEAKAGE_WORD lc_word25  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[25]),.D(D),.D_Q(lc_word[24]),.Q(lc_word[25][31:0]));
86
87  LEAKAGE_WORD lc_word26  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[26]),.D(D),.D_Q(lc_word[25]),.Q(lc_word[26][31:0]));
88
89  LEAKAGE_WORD lc_word27  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[27]),.D(D),.D_Q(lc_word[26]),.Q(lc_word[27][31:0]));
90
91  LEAKAGE_WORD lc_word28  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[28]),.D(D),.D_Q(lc_word[27]),.Q(lc_word[28][31:0]));
92
93  LEAKAGE_WORD lc_word29  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[29]),.D(D),.D_Q(lc_word[28]),.Q(lc_word[29][31:0]));
94
95  LEAKAGE_WORD lc_word30  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[30]),.D(D),.D_Q(lc_word[29]),.Q(lc_word[30][31:0]));
96
97  LEAKAGE_WORD lc_word31  (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
        SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[31]),.D(D),.D_Q(lc_word[30]),.Q(lc_word[31][31:0]));
98
99  //DATA OUTPUT MUX
100 wire    [31:0]  q_mux;
101
102 assign  Q[31:0] = q_mux[31:0];
103
104 assign q_mux[31:0] = ({32{SEL_DEC[0]}} & lc_word[0][31:0]) |
105                      ({32{SEL_DEC[1]}} & lc_word[1][31:0]) |
106                      ({32{SEL_DEC[2]}} & lc_word[2][31:0]) |
107                      ({32{SEL_DEC[3]}} & lc_word[3][31:0]) |
108                      ({32{SEL_DEC[4]}} & lc_word[4][31:0]) |
109                      ({32{SEL_DEC[5]}} & lc_word[5][31:0]) |
110                      ({32{SEL_DEC[6]}} & lc_word[6][31:0]) |
111                      ({32{SEL_DEC[7]}} & lc_word[7][31:0]) |
112                      ({32{SEL_DEC[8]}} & lc_word[8][31:0]) |
113                      ({32{SEL_DEC[9]}} & lc_word[9][31:0]) |
114                      ({32{SEL_DEC[10]}} & lc_word[10][31:0]) |
115                      ({32{SEL_DEC[11]}} & lc_word[11][31:0]) |
116                      ({32{SEL_DEC[12]}} & lc_word[12][31:0]) |
117                      ({32{SEL_DEC[13]}} & lc_word[13][31:0]) |
118                      ({32{SEL_DEC[14]}} & lc_word[14][31:0]) |
119                      ({32{SEL_DEC[15]}} & lc_word[15][31:0]) |
120                      ({32{SEL_DEC[16]}} & lc_word[16][31:0]) |
121                      ({32{SEL_DEC[17]}} & lc_word[17][31:0]) |
122                      ({32{SEL_DEC[18]}} & lc_word[18][31:0]) |
123                      ({32{SEL_DEC[19]}} & lc_word[19][31:0]) |
124                      ({32{SEL_DEC[20]}} & lc_word[20][31:0]) |
```

```
125                         ({32{SEL_DEC[21]}} & lc_word[21][31:0]) |
126                         ({32{SEL_DEC[22]}} & lc_word[22][31:0]) |
127                         ({32{SEL_DEC[23]}} & lc_word[23][31:0]) |
128                         ({32{SEL_DEC[24]}} & lc_word[24][31:0]) |
129                         ({32{SEL_DEC[25]}} & lc_word[25][31:0]) |
130                         ({32{SEL_DEC[26]}} & lc_word[26][31:0]) |
131                         ({32{SEL_DEC[27]}} & lc_word[27][31:0]) |
132                         ({32{SEL_DEC[28]}} & lc_word[28][31:0]) |
133                         ({32{SEL_DEC[29]}} & lc_word[29][31:0]) |
134                         ({32{SEL_DEC[30]}} & lc_word[30][31:0]) |
135                         ({32{SEL_DEC[31]}} & lc_word[31][31:0]) ;
136
137 endmodule
138
139
140 //*****************************************************************************
141 //LEAKAGE CIRCUIT 32-BIT WORD
142 //*****************************************************************************
143 //A single 32-bit WORD implements a shift register (SRL)
144 //The Data can be read from the BUS or shifted in from the previous REG
145 //
146 //CLKEN (clock enable) is used to switch the clock on/off to the whole WORD
147 //SHIFT_EN (shift enable) is used to control shifting operation, hence when SHIFT_EN is 0, the
148 //module must be still operational in case of reading Data from the BUS
149 //
150 //In case of reading from the BUS SHIFT_WRITE is set to 0 to MUX the Data from the BUS,
151 //WRITE_EN must be 1 and SEL must be 1 as well to choose the WORD
152 //
153 //In case of shifting from the previous REG, SHIFT_WRITE is 1 chosing the Data from input,
154 //also EN (enable) must be 1 (controlled by watermark) to shift,
155 //SEL and WRITE_EN are not important
156 //*****************************************************************************
157
158 'timescale 1ns / 1ps
159
160 module LEAKAGE_WORD (
161
162 input   wire            CLK,
163 input   wire            CLKEN,          //Clock Enable
164 input   wire            SHIFT_EN,       //Shift Enable
165 input   wire            SHIFT_CTRL,     //Shift Control (should be connected to watermark to determine switching
        modulation)
166 input   wire            SHIFT_WRITE,    //Shift (1), Write (0)
167 input   wire            WRITE_EN,       //Write Enable
168 input   wire            SEL,            //Select
169 input   wire    [31:0]  D,              //Data from BUS
170 input   wire    [31:0]  D_Q,            //Data from prev REG
171 output  wire    [31:0]  Q               //Data out
172
173 );
174
175 wire    [31:0]  d_sel;  //Data from BUS or previous REG
176 reg     [31:0]  q_reg;
177
178 assign d_sel[31:0] = (SHIFT_WRITE) ? D_Q[31:0] : D[31:0];
179
180         always@(posedge CLK)
181           if(CLKEN)
182                   //In case of shifting - the mode must be set to shift (SHIFT_WRITE==1), shifting must be enabled (
        SHIFT_EN==1) and
183                   //the shift control must be 1 (SHIFT_CTRL==1)
184                   //In case of writing - the mode must be set to write (SHIFT_WRITE==0), writing must be enabled (
        WRITE_EN==1) and
185                   //the word must be selected (SEL==1)
186             if((SHIFT_WRITE & SHIFT_EN & SHIFT_CTRL) | (SEL & WRITE_EN & ~SHIFT_WRITE)) q_reg[31:0] <= d_sel[31:0];
187
188 assign Q[31:0] = q_reg[31:0];
189
190 endmodule
```

B.1.8.4 Interval Counter

```
1  //*****************************************************************************
2  //INTERVAL COUNTER MODULE
3  //*****************************************************************************
4  //Contains 2 32-bit count down counters with initial registers
5  //One of the counters will be used to count the period of the watermark sequence single cycle
6  //The other counter will be used to count the interval between two consecutive watermark
7  //cycles
8  //*****************************************************************************
9
10 `timescale 1ns / 1ps
11
12 module INTERVAL_COUNTER (
13
14 input   wire            CLK,
15 input   wire            CLKEN,          //Clock Enable
16 input   wire            RESETn,         //Reset negative edge
17 input   wire            CNT_LD_INI,     //Interval Counter Load Initial
18 input   wire            CNT0_CNT_EN,    //Counter 0 Count Enable
19 input   wire            CNT1_CNT_EN,    //Counter 1 Count Enable
20 input   wire            CNT0_SEL,       //Counter 0 Select
21 input   wire            CNT1_SEL,       //Counter 1 Select
22 input   wire            WRITE_EN,       //Write Enable
23 input   wire    [31:0]  D,              //Input Data
24
25 output  wire            CNT0_ZERO_FLAG, //Counter 0 Zero Flag
26 output  wire            CNT1_ZERO_FLAG, //Counter 1 Zero Flag
27 output  wire    [31:0]  Q_INI           //Output Initial Data
28 );
29
30 //Instantiate 2 Counters (1st - Watermark Sequence Counter,
31 //2nd - Clock Interval Counter)
32 wire    [31:0]  cnt0_q_ini;
33 wire    [31:0]  cnt1_q_ini;
34
35         CNT_DOWN_32_REG_INI wat_seq_count       (
36                                                         .CLK(CLK),
37                                                         .CLKEN(CLKEN),
38                                                         .RESETn(RESETn),
39                                                         .LD_INI(CNT_LD_INI),
40                                                         .CNT_EN(CNT0_CNT_EN),
41                                                         .SEL(CNT0_SEL),
42                                                         .WRITE_EN(WRITE_EN),
43                                                         .D(D[31:0]),
44                                                         .ZERO_FLAG(CNT0_ZERO_FLAG),
45                                                         .Q_INI(cnt0_q_ini[31:0])
46                                                 );
47         CNT_DOWN_32_REG_INI interval_count      (
48                                                         .CLK(CLK),
49                                                         .CLKEN(CLKEN),
50                                                         .RESETn(RESETn),
51                                                         .LD_INI(CNT_LD_INI),
52                                                         .CNT_EN(CNT1_CNT_EN),
53                                                         .SEL(CNT1_SEL),
54                                                         .WRITE_EN(WRITE_EN),
55                                                         .D(D[31:0]),
56                                                         .ZERO_FLAG(CNT1_ZERO_FLAG),
57                                                         .Q_INI(cnt1_q_ini[31:0])
58                                                 );
59
60 //OUTPUT DATA MUX
61 wire    [31:0]  q_mux;
62
63 assign q_mux[31:0] = ({32{CNT0_SEL}} & cnt0_q_ini[31:0]) |
64                      ({32{CNT1_SEL}} & cnt1_q_ini[31:0]);
65
66 assign Q_INI[31:0] = q_mux[31:0];
67
68 endmodule
69
70
71 //*****************************************************************************
72 //32-bit COUNT DOWN COUNTER WITH 32-BIT REGISTER TO HOLD INITIAL VALUE
73 //*****************************************************************************
74 //The Data from the BUS can be loaded into the Initial Register reg_ini (memory mapped)
75 //In order to do that SEL and WRITE_EN must be 1
76 //
77 //The counter loads the initial value from reg_ini to reg_count when LD_INI (load initial)
```

```
78  //is 1. When CNT_EN (count enable) is 1 the counter counts down and when it reaches 0, the
79  //Zero Flag is set (1)
80  //*********************************************************************************
81
82  `timescale 1ns / 1ps
83
84  module CNT_DOWN_32_REG_INI (
85
86  input   wire            CLK,
87  input   wire            CLKEN,          //Clock Enable
88  input   wire            RESETn,         //Reset negative edge
89  input   wire            LD_INI,         //Load Initial Value from INI REG to COUNT REG
90  input   wire            CNT_EN,         //Count Enable
91  input   wire            SEL,            //Select
92  input   wire            WRITE_EN,       //Write Enable
93  input   wire    [31:0]  D,              //Input Data
94
95  output  wire            ZERO_FLAG,      //Count == 0x0000_0000
96  output  wire    [31:0]  Q_INI           //Output Data
97
98  );
99
100 //Registers
101
102         reg     [31:0]  reg_ini;        //Initial Register
103         reg     [31:0]  reg_count;      //Count Register
104
105 //Store Initial Data (if block is selected and write is enabled)
106
107         always@(posedge CLK)
108             if(CLKEN)
109                 if(SEL & WRITE_EN) reg_ini[31:0] <= D[31:0];
110
111
112 //Load Initial Data into Count Register if LD_INI is 1
113 //Count down if CNT_EN is 1
114 //Both CNT_EN and LD_INI cannot be 1 at the same instance
115
116         always@(posedge CLK or negedge RESETn)
117             if(~RESETn)
118                 reg_count <= {32{1'b1}};
119             else begin
120                 if(CLKEN)
121                     begin
122                         if(LD_INI)
123                             reg_count[31:0] <= reg_ini[31:0];
124                         else if(CNT_EN)
125                             reg_count[31:0] <= reg_count[31:0] - 32'b1;
126                     end
127             end
128
129 //Set ZERO_FLAG to 1 when Counter Register reaches 0
130
131 assign ZERO_FLAG = ~(|reg_count[31:0]);
132
133 //OUTPUT DATA
134 assign Q_INI[31:0] = reg_ini[31:0];
135
136 endmodule
```

B.1.8.5 Watermark Controller

```
1  module WATERMARK_CONTROLLER (
2
3  input   wire            CLK,
4  input   wire            RESETn,         //Reset (neg active)
5
6  //----------INPUT CONTROL----------
7  //Watermark Controller
8  input   wire    [1:0]   OP_MODE,        //Operation Mode
9
10 //Watermark Generation Circuit
11 input   wire            WGC_SHIFT_EN,   //Watermark Generation Circuit Shift Enable
12
13 //Leakage Circuit
14 input   wire            LC_SHIFT_EN,    //Leakage Circuit Shift Enable
15
16 //Interval Counter
17 input   wire            IC_CNT_EN,      //Interval Counter Count Enable
18
19 //M0 Trigger Enable
20 input   wire            M0_TRIG_EN,     //Cortex M0 Trigger Enable - signal for the multiplier enable
21
22 //----------INTERVAL COUNTER INTERFACE----------
23 input   wire            IC_CNT0_ZERO_FLAG,      //Interval Counter 0 Zero Flag
24 input   wire            IC_CNT1_ZERO_FLAG,      //Interval Counter 1 Zero Flag
25
26 //----------OUTPUT CONTROL----------
27 //WGC and LC
28 output  wire            WGC_SHIFT_ENO,          //Watermark Gen. Circuit Shift Enable Out
29 output  wire            LC_SHIFT_ENO,           //Leakage Circuit Shift Enable Out
30
31 //IC
32 output  wire            IC_CNT_LD_INI,          //Interval Counter Load Initial
33 output  wire            IC_CNT0_ENO,            //Interval Counter 0 Count Enable Out
34 output  wire            IC_CNT1_ENO             //Interval Counter 1 Count Enable Out
35
36 );
37
38
39 //OPERATION MODE PARAMETERS
40 localparam      ON              = 2'b01;        //Normal always ON (WGC and LC) operation
41 localparam      INTERVAL        = 2'b10;        //Interval using Interval Counter
42 localparam      M0_TRIG         = 2'b11;        //M0 Trigger - trigger selected by preprogrammed operation
43
44 //----------STATE MACHINE
45
46 //Parameters (4 bits for ECS Tokachi as will have more states for triggering, for A5 4th bit will be optimised)
47 localparam      S_MODECHOICE                            = 4'b0000;
48 localparam      S_ON_START                              = 4'b0001;
49 localparam      S_ON_ENABLE                             = 4'b0010;
50 localparam      S_INT_START                             = 4'b0011;
51 localparam      S_INT_LOAD_CNT_INI                      = 4'b0100;
52 localparam      S_INT_SHIFT_EN_CNT0_EN                  = 4'b0101;
53 localparam      S_INT_SHIFT_DIS_CNT0_DIS_CNT1_EN        = 4'b0110;
54 localparam      S_M0_TRIG_START                         = 4'b0111;
55 localparam      S_M0_TRIG_WAIT_TRIG                     = 4'b1000;
56 localparam      S_M0_TRIG_LOAD_CNT_INI                  = 4'b1001;
57 localparam      S_M0_TRIG_CNT1_EN                       = 4'b1010;
58 localparam      S_M0_TRIG_SHIFT_EN_CNT0_EN              = 4'b1011;
59
60 //Sequential
61 reg     [3:0]   state;
62
63 //Combinational
64 reg     [3:0]   next_state;
65
66 //Sequential logic
67         always@(posedge CLK or negedge RESETn)
68             if (~RESETn)
69                 state <= S_MODECHOICE;
70             else
71                 state <= next_state;
72
73 //Combinational logic
74         always@(*)
75             begin
76                 next_state = state;
77                 case(state)
```

```
78                              S_MODECHOICE                    :       begin
79                                                                              if(OP_MODE == ON)
80                                                                              next_state = S_ON_START;
81                                                                              else if(OP_MODE == INTERVAL)
82                                                                              next_state = S_INT_START;
83                                                                              else if(OP_MODE == M0_TRIG)
84                                                                              next_state = S_M0_TRIG_START;
85                                                                      end
86                      //----------OPERATION MODE ON----------
87
88                              //Wait until WGC and LC input shift enables are on
89                              S_ON_START                      :       if(WGC_SHIFT_EN & LC_SHIFT_EN)
90                                                                              next_state = S_ON_ENABLE;
91
92                              //When both shift enables are on, stay in this state until reset
93                              S_ON_ENABLE                     :       ;
94
95                      //----------OPERATION MODE INTERVAL----------
96
97                              //Wait until WGC, LC and IC input shift enables are on
98                              S_INT_START                     :       if(WGC_SHIFT_EN & LC_SHIFT_EN & IC_CNT_EN)
99                                                                              next_state = S_INT_LOAD_CNT_INI;
100
101                             //Enable WGC and LC, load INI to IC (first has to load initial, then can start counting).
        Disable CNT1 (for loop operation)
102                             S_INT_LOAD_CNT_INI              :                       next_state = S_INT_SHIFT_EN_CNT0_EN
        ;
103
104                             //Enable Interval Counter 0 and wait for the Zero Flag
105                             S_INT_SHIFT_EN_CNT0_EN          :       if(IC_CNT0_ZERO_FLAG)
106                                                                             next_state =
        S_INT_SHIFT_DIS_CNT0_DIS_CNT1_EN;
107
108                             //Disable WGC, LC and CNT0. Enable CNT1 and wait for Zero Flag. On completion go back to
        S_INT_ENABLE_WGC_LC
109                             S_INT_SHIFT_DIS_CNT0_DIS_CNT1_EN:       if(IC_CNT1_ZERO_FLAG)
110                                                                             next_state = S_INT_SHIFT_EN_CNT0_EN
        ;
111
112                     //----------OPERATION MODE M0 TRIGGER----------
113                             //Not supported (for A5)
114                             //Wait until WGC, LC and IC are enabled
115                             S_M0_TRIG_START                 :       if(WGC_SHIFT_EN & LC_SHIFT_EN & IC_CNT_EN)
116                                                                             next_state = S_M0_TRIG_WAIT_TRIG;
117                             //Wait for the trigger signal
118                             S_M0_TRIG_WAIT_TRIG             :       if(M0_TRIG_EN)
119                                                                             next_state = S_M0_TRIG_LOAD_CNT_INI
        ;
120                             //Load counters
121                             S_M0_TRIG_LOAD_CNT_INI          :                       next_state = S_M0_TRIG_CNT1_EN;
122
123                             //Enable counter 1 (counts the interval from the trigger signal to the beginning of the
        watermark sequence)
124                             S_M0_TRIG_CNT1_EN               :       if(IC_CNT1_ZERO_FLAG)
125                                                                             next_state =
        S_M0_TRIG_SHIFT_EN_CNT0_EN;
126
127                             //Disable counter 1, enable counter 0 (counter 0 counts the watermark sequence period),
128                             S_M0_TRIG_SHIFT_EN_CNT0_EN      :       if(IC_CNT0_ZERO_FLAG)
129                                                                             next_state = S_M0_TRIG_WAIT_TRIG;
130
131                     //Default
132                             default                         :                       next_state = S_MODECHOICE;
133                     endcase
134             end
135
136     //OUTPUTS
137
138     //Sequential
139     reg     wgc_shift_en_out_q;
140     reg     lc_shift_en_out_q;
141     reg     ic_cnt0_en_out_q;
142     reg     ic_cnt1_en_out_q;
143     reg     ic_cnt_ld_ini_out_q;
144
145     //Combinational
146     reg     wgc_shift_en_out;
147     reg     lc_shift_en_out;
148     reg     ic_cnt0_en_out;
149     reg     ic_cnt1_en_out;
```

```
150 reg     ic_cnt_ld_ini_out;
151
152 //Sequential logic
153         always@(posedge CLK or negedge RESETn)
154             if(~RESETn)
155                 begin
156                   wgc_shift_en_out_q      <= 1'b0;
157                   lc_shift_en_out_q       <= 1'b0;
158                   ic_cnt0_en_out_q        <= 1'b0;
159                   ic_cnt1_en_out_q        <= 1'b0;
160                   ic_cnt_ld_ini_out_q     <= 1'b0;
161                 end
162             else
163                 begin
164                   wgc_shift_en_out_q      <= wgc_shift_en_out;
165                   lc_shift_en_out_q       <= lc_shift_en_out;
166                   ic_cnt0_en_out_q        <= ic_cnt0_en_out;
167                   ic_cnt1_en_out_q        <= ic_cnt1_en_out;
168                   ic_cnt_ld_ini_out_q     <= ic_cnt_ld_ini_out;
169                 end
170
171 //Combinational logic
172         always@(*)
173             begin
174                   wgc_shift_en_out        = wgc_shift_en_out_q;
175                   lc_shift_en_out         = lc_shift_en_out_q;
176                   ic_cnt0_en_out          = ic_cnt0_en_out_q;
177                   ic_cnt1_en_out          = ic_cnt1_en_out_q;
178                   ic_cnt_ld_ini_out       = ic_cnt_ld_ini_out_q;
179
180                   case(state)
181                   //----------OPERATION MODE ON----------
182                           //Switch WGC and LC shift enables high, interval counter inactive
183                           S_ON_ENABLE                     :       begin
184                                                                           wgc_shift_en_out        = 1'b1;
185                                                                           lc_shift_en_out         = 1'b1;
186                                                                   end
187
188                   //----------OPERATION MODE INTERVAL----------
189                           //Load initial data into interval counter
190                           S_INT_LOAD_CNT_INI              :               ic_cnt_ld_ini_out       = 1'b1;
191
192                           //Enable shifting and counter CNT0 as long as the ZERO FLAG from CNT0 is low, otherwise
        disable shifting and switch CNT1 ON
193                           S_INT_SHIFT_EN_CNT0_EN          :       begin
194                                                                           if(IC_CNT0_ZERO_FLAG) begin
195                                                                                   wgc_shift_en_out        = 1'b0;
196                                                                                   lc_shift_en_out         = 1'b0;
197                                                                                   ic_cnt0_en_out          = 1'b0;
198                                                                                   ic_cnt1_en_out          = 1'b1;
199                                                                           end else begin
200                                                                                   wgc_shift_en_out        = 1'b1;
201                                                                                   lc_shift_en_out         = 1'b1;
202                                                                                   ic_cnt0_en_out          = 1'b1;
203                                                                                   ic_cnt1_en_out          = 1'b0;
204                                                                           end
205
206                                                                           ic_cnt_ld_ini_out       = 1'b0;
207                                                                   end
208
209                   //Keep CNT1 Enabled as long as the ZERO FLAG is low, as soon as the ZERO FLAG is high, disable CNT1
        (on the next clock) and load initial data
210                           S_INT_SHIFT_DIS_CNT0_DIS_CNT1_EN:       begin
211                                                                           if(IC_CNT1_ZERO_FLAG) begin
212                                                                                   ic_cnt_ld_ini_out       = 1'b1;
213                                                                                   ic_cnt1_en_out          = 1'b0;
214                                                                           end else
215                                                                                   ic_cnt1_en_out          = 1'b1;
216                                                                   end
217
218                   //----------OPERATION MODE TRIGGER----------
219                           //Load counters
220                           S_M0_TRIG_LOAD_CNT_INI          :               ic_cnt_ld_ini_out       = 1'b1;
221
222                           //Enable counter 1 (counts the interval from the trigger signal to the beginning of the
        watermark sequence)
223                           S_M0_TRIG_CNT1_EN               :       begin
224                                                                           if(IC_CNT1_ZERO_FLAG) begin
225                                                                                   ic_cnt1_en_out          = 1'b0;
226                                                                                   ic_cnt0_en_out          = 1'b1;
```

```
227                                                                     wgc_shift_en_out        = 1'b1;
228                                                                     lc_shift_en_out         = 1'b1;
229                                                             end else
230                                                                     ic_cnt1_en_out          = 1'b1;
231
232                                                             ic_cnt_ld_ini_out       = 1'b0;
233                                                     end
234
235                     //Disable counter 1, enable counter 0 (counter 0 counts the watermark sequence period),
236                     S_M0_TRIG_SHIFT_EN_CNT0_EN      :       begin
237                                                             //If counter 1 finished (count == 0 hence
        WGC did full sequence cycle)
238                                                             //switch the WGC, LC and counter 1 off
239                                                             if(IC_CNT0_ZERO_FLAG) begin
240                                                                     ic_cnt0_en_out          = 1'b0;
241                                                                     wgc_shift_en_out        = 1'b0;
242                                                                     lc_shift_en_out         = 1'b0;
243                                                             end
244                                                     end
245
246             //Default
247             default :       begin
248                                     wgc_shift_en_out        = 1'b0;
249                                     lc_shift_en_out         = 1'b0;
250                                     ic_cnt0_en_out          = 1'b0;
251                                     ic_cnt1_en_out          = 1'b0;
252                                     ic_cnt_ld_ini_out       = 1'b0;
253                             end
254             endcase
255         end
256
257 //----------
258 assign WGC_SHIFT_ENO = wgc_shift_en_out_q;
259 assign LC_SHIFT_ENO = lc_shift_en_out_q;
260 assign IC_CNT_LD_INI = ic_cnt_ld_ini_out_q;
261 assign IC_CNT0_ENO = ic_cnt0_en_out_q;
262 assign IC_CNT1_ENO = ic_cnt1_en_out_q;
263
264 endmodule
```

B.1.8.6 Noise Generator

```
1  //*****************************************************************************
2  //NOISE GENERATOR
3  //*****************************************************************************
4  //Implements a big shift register (32 x 32-bit WORD)
5  //
6  //In order to make shifting manner more random a 32-bit LFSR, which sits above the
7  //register block, is used to control switching of every single column. Hence, if
8  //bit 0 of LFSR is 1 all register bits 0 will shift on the next clock cycle
9  //*****************************************************************************
10
11 `timescale 1ns / 1ps
12
13 module NOISE_GEN (
14
15 input  wire          CLK,
16 input  wire          CLKEN,          //Clock Enable
17 input  wire          SHIFT_EN,       //Shift Enable
18 input  wire  [31:0]  SHIFT_CTRL,     //Shift Control bit-wise
19 input  wire          SHIFT_WRITE,    //Shift (1), Write (0)
20 input  wire          WRITE_EN,       //Write Enable
21 input  wire  [31:0]  SEL_DEC,        //Selection Decode
22 input  wire  [31:0]  D,              //Input Data
23
24 output wire  [31:0]  Q               //Output Data
25 );
26
27 //Noise Generator Word Outputs
28 wire  [31:0]  noise_gen_word [0:31];
29
30 //NOISE GENERATOR WORDS INSTANTATION - 32 X 32-bit NOISE GENERATOR
31
32 NOISE_GEN_WORD lc_word0        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[0]),.D(D),.D_Q(noise_gen_word[31][31:0]),.Q(noise_gen_word
       [0][31:0]));
33
34 NOISE_GEN_WORD lc_word1        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[1]),.D(D),.D_Q(noise_gen_word[0][31:0]),.Q(noise_gen_word
       [1][31:0]));
35
36 NOISE_GEN_WORD lc_word2        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[2]),.D(D),.D_Q(noise_gen_word[1][31:0]),.Q(noise_gen_word
       [2][31:0]));
37
38 NOISE_GEN_WORD lc_word3        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[3]),.D(D),.D_Q(noise_gen_word[2][31:0]),.Q(noise_gen_word
       [3][31:0]));
39
40 NOISE_GEN_WORD lc_word4        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[4]),.D(D),.D_Q(noise_gen_word[3][31:0]),.Q(noise_gen_word
       [4][31:0]));
41
42 NOISE_GEN_WORD lc_word5        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[5]),.D(D),.D_Q(noise_gen_word[4][31:0]),.Q(noise_gen_word
       [5][31:0]));
43
44 NOISE_GEN_WORD lc_word6        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[6]),.D(D),.D_Q(noise_gen_word[5][31:0]),.Q(noise_gen_word
       [6][31:0]));
45
46 NOISE_GEN_WORD lc_word7        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[7]),.D(D),.D_Q(noise_gen_word[6][31:0]),.Q(noise_gen_word
       [7][31:0]));
47
48 NOISE_GEN_WORD lc_word8        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[8]),.D(D),.D_Q(noise_gen_word[7][31:0]),.Q(noise_gen_word
       [8][31:0]));
49
50 NOISE_GEN_WORD lc_word9        (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[9]),.D(D),.D_Q(noise_gen_word[8][31:0]),.Q(noise_gen_word
       [9][31:0]));
51
52 NOISE_GEN_WORD lc_word10       (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[10]),.D(D),.D_Q(noise_gen_word[9][31:0]),.Q(noise_gen_word
       [10][31:0]));
53
```

```
54 NOISE_GEN_WORD lc_word11 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[11]),.D(D),.D_Q(noise_gen_word[10][31:0]),.Q(noise_gen_word
       [11][31:0]));
55
56 NOISE_GEN_WORD lc_word12 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[12]),.D(D),.D_Q(noise_gen_word[11][31:0]),.Q(noise_gen_word
       [12][31:0]));
57
58 NOISE_GEN_WORD lc_word13 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[13]),.D(D),.D_Q(noise_gen_word[12][31:0]),.Q(noise_gen_word
       [13][31:0]));
59
60 NOISE_GEN_WORD lc_word14 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[14]),.D(D),.D_Q(noise_gen_word[13][31:0]),.Q(noise_gen_word
       [14][31:0]));
61
62 NOISE_GEN_WORD lc_word15 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[15]),.D(D),.D_Q(noise_gen_word[14][31:0]),.Q(noise_gen_word
       [15][31:0]));
63
64 NOISE_GEN_WORD lc_word16 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[16]),.D(D),.D_Q(noise_gen_word[15][31:0]),.Q(noise_gen_word
       [16][31:0]));
65
66 NOISE_GEN_WORD lc_word17 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[17]),.D(D),.D_Q(noise_gen_word[16][31:0]),.Q(noise_gen_word
       [17][31:0]));
67
68 NOISE_GEN_WORD lc_word18 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[18]),.D(D),.D_Q(noise_gen_word[17][31:0]),.Q(noise_gen_word
       [18][31:0]));
69
70 NOISE_GEN_WORD lc_word19 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[19]),.D(D),.D_Q(noise_gen_word[18][31:0]),.Q(noise_gen_word
       [19][31:0]));
71
72 NOISE_GEN_WORD lc_word20 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[20]),.D(D),.D_Q(noise_gen_word[19][31:0]),.Q(noise_gen_word
       [20][31:0]));
73
74 NOISE_GEN_WORD lc_word21 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[21]),.D(D),.D_Q(noise_gen_word[20][31:0]),.Q(noise_gen_word
       [21][31:0]));
75
76 NOISE_GEN_WORD lc_word22 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[22]),.D(D),.D_Q(noise_gen_word[21][31:0]),.Q(noise_gen_word
       [22][31:0]));
77
78 NOISE_GEN_WORD lc_word23 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[23]),.D(D),.D_Q(noise_gen_word[22][31:0]),.Q(noise_gen_word
       [23][31:0]));
79
80 NOISE_GEN_WORD lc_word24 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[24]),.D(D),.D_Q(noise_gen_word[23][31:0]),.Q(noise_gen_word
       [24][31:0]));
81
82 NOISE_GEN_WORD lc_word25 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[25]),.D(D),.D_Q(noise_gen_word[24][31:0]),.Q(noise_gen_word
       [25][31:0]));
83
84 NOISE_GEN_WORD lc_word26 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[26]),.D(D),.D_Q(noise_gen_word[25][31:0]),.Q(noise_gen_word
       [26][31:0]));
85
86 NOISE_GEN_WORD lc_word27 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[27]),.D(D),.D_Q(noise_gen_word[26][31:0]),.Q(noise_gen_word
       [27][31:0]));
87
88 NOISE_GEN_WORD lc_word28 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[28]),.D(D),.D_Q(noise_gen_word[27][31:0]),.Q(noise_gen_word
       [28][31:0]));
89
90 NOISE_GEN_WORD lc_word29 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[29]),.D(D),.D_Q(noise_gen_word[28][31:0]),.Q(noise_gen_word
       [29][31:0]));
91
92 NOISE_GEN_WORD lc_word30 (.CLK(CLK),.CLKEN(CLKEN),.SHIFT_EN(SHIFT_EN),.SHIFT_CTRL(SHIFT_CTRL),.SHIFT_WRITE(
       SHIFT_WRITE),.WRITE_EN(WRITE_EN),.SEL(SEL_DEC[30]),.D(D),.D_Q(noise_gen_word[29][31:0]),.Q(noise_gen_word
       [30][31:0]));
93
```

```
94  NOISE_GEN_WORD lc_word31          (.CLK(CLK) ,.CLKEN(CLKEN) ,.SHIFT_EN(SHIFT_EN) ,.SHIFT_CTRL(SHIFT_CTRL) ,.SHIFT_WRITE(
        SHIFT_WRITE) ,.WRITE_EN(WRITE_EN) ,.SEL(SEL_DEC[31]) ,.D(D) ,.D_Q(noise_gen_word[30][31:0]) ,.Q(noise_gen_word
        [31][31:0]));
95
96  //DATA OUTPUT MUX
97  wire     [31:0]  q_mux;
98
99  assign  Q[31:0] = q_mux[31:0];
100
101 assign q_mux[31:0] = ({32{SEL_DEC[0]}} & noise_gen_word[0][31:0]) |
102                       ({32{SEL_DEC[1]}} & noise_gen_word[1][31:0]) |
103                       ({32{SEL_DEC[2]}} & noise_gen_word[2][31:0]) |
104                       ({32{SEL_DEC[3]}} & noise_gen_word[3][31:0]) |
105                       ({32{SEL_DEC[4]}} & noise_gen_word[4][31:0]) |
106                       ({32{SEL_DEC[5]}} & noise_gen_word[5][31:0]) |
107                       ({32{SEL_DEC[6]}} & noise_gen_word[6][31:0]) |
108                       ({32{SEL_DEC[7]}} & noise_gen_word[7][31:0]) |
109                       ({32{SEL_DEC[8]}} & noise_gen_word[8][31:0]) |
110                       ({32{SEL_DEC[9]}} & noise_gen_word[9][31:0]) |
111                       ({32{SEL_DEC[10]}} & noise_gen_word[10][31:0]) |
112                       ({32{SEL_DEC[11]}} & noise_gen_word[11][31:0]) |
113                       ({32{SEL_DEC[12]}} & noise_gen_word[12][31:0]) |
114                       ({32{SEL_DEC[13]}} & noise_gen_word[13][31:0]) |
115                       ({32{SEL_DEC[14]}} & noise_gen_word[14][31:0]) |
116                       ({32{SEL_DEC[15]}} & noise_gen_word[15][31:0]) |
117                       ({32{SEL_DEC[16]}} & noise_gen_word[16][31:0]) |
118                       ({32{SEL_DEC[17]}} & noise_gen_word[17][31:0]) |
119                       ({32{SEL_DEC[18]}} & noise_gen_word[18][31:0]) |
120                       ({32{SEL_DEC[19]}} & noise_gen_word[19][31:0]) |
121                       ({32{SEL_DEC[20]}} & noise_gen_word[20][31:0]) |
122                       ({32{SEL_DEC[21]}} & noise_gen_word[21][31:0]) |
123                       ({32{SEL_DEC[22]}} & noise_gen_word[22][31:0]) |
124                       ({32{SEL_DEC[23]}} & noise_gen_word[23][31:0]) |
125                       ({32{SEL_DEC[24]}} & noise_gen_word[24][31:0]) |
126                       ({32{SEL_DEC[25]}} & noise_gen_word[25][31:0]) |
127                       ({32{SEL_DEC[26]}} & noise_gen_word[26][31:0]) |
128                       ({32{SEL_DEC[27]}} & noise_gen_word[27][31:0]) |
129                       ({32{SEL_DEC[28]}} & noise_gen_word[28][31:0]) |
130                       ({32{SEL_DEC[29]}} & noise_gen_word[29][31:0]) |
131                       ({32{SEL_DEC[30]}} & noise_gen_word[30][31:0]) |
132                       ({32{SEL_DEC[31]}} & noise_gen_word[31][31:0]) ;
133
134 endmodule
135
136
137 //*****************************************************************************************
138 //NOISE GENERATOR 32-bit WORD
139 //*****************************************************************************************
140 //A single 32-bit noise generator word.
141 //
142 //32-bit data can be written to the register
143 //
144 //Shifting operation is a bit-wise operation. The controller (SHIFT_CTRL) controls how
145 //many bits shift in any clock cycles. Hence, shift can be word-wise, all 32-bits can
146 //be shifted out, or only a single bit can be shifted
147 //*****************************************************************************************
148
149 `timescale 1ns / 1ps
150
151 module NOISE_GEN_WORD (
152
153 input   wire            CLK,
154 input   wire            CLKEN,              //Clock Enable
155 input   wire            SHIFT_EN,           //Shift Enable
156 input   wire    [31:0]  SHIFT_CTRL,         //Shift Control bit-wise
157 input   wire            SHIFT_WRITE,        //Shift (1), Write (0)
158 input   wire            WRITE_EN,           //Write Enable
159 input   wire            SEL,                //Select
160 input   wire    [31:0]  D,                  //Data from BUS
161 input   wire    [31:0]  D_Q,                //Data from prev REG
162 output  wire    [31:0]  Q                   //Data out
163
164 );
165
166 reg     [31:0]  q_reg;          //Word Register
167
168 wire    [31:0]  bw_word_shift;  //Bit-wise word shift
169
170 assign bw_word_shift[0] = (SHIFT_CTRL[0]) ? D_Q[0] : q_reg[0];
171 assign bw_word_shift[1] = (SHIFT_CTRL[1]) ? D_Q[1] : q_reg[1];
```

```
172  assign bw_word_shift[2] = (SHIFT_CTRL[2]) ? D_Q[2] : q_reg[2];
173  assign bw_word_shift[3] = (SHIFT_CTRL[3]) ? D_Q[3] : q_reg[3];
174  assign bw_word_shift[4] = (SHIFT_CTRL[4]) ? D_Q[4] : q_reg[4];
175  assign bw_word_shift[5] = (SHIFT_CTRL[5]) ? D_Q[5] : q_reg[5];
176  assign bw_word_shift[6] = (SHIFT_CTRL[6]) ? D_Q[6] : q_reg[6];
177  assign bw_word_shift[7] = (SHIFT_CTRL[7]) ? D_Q[7] : q_reg[7];
178  assign bw_word_shift[8] = (SHIFT_CTRL[8]) ? D_Q[8] : q_reg[8];
179  assign bw_word_shift[9] = (SHIFT_CTRL[9]) ? D_Q[9] : q_reg[9];
180  assign bw_word_shift[10] = (SHIFT_CTRL[10]) ? D_Q[10] : q_reg[10];
181  assign bw_word_shift[11] = (SHIFT_CTRL[11]) ? D_Q[11] : q_reg[11];
182  assign bw_word_shift[12] = (SHIFT_CTRL[12]) ? D_Q[12] : q_reg[12];
183  assign bw_word_shift[13] = (SHIFT_CTRL[13]) ? D_Q[13] : q_reg[13];
184  assign bw_word_shift[14] = (SHIFT_CTRL[14]) ? D_Q[14] : q_reg[14];
185  assign bw_word_shift[15] = (SHIFT_CTRL[15]) ? D_Q[15] : q_reg[15];
186  assign bw_word_shift[16] = (SHIFT_CTRL[16]) ? D_Q[16] : q_reg[16];
187  assign bw_word_shift[17] = (SHIFT_CTRL[17]) ? D_Q[17] : q_reg[17];
188  assign bw_word_shift[18] = (SHIFT_CTRL[18]) ? D_Q[18] : q_reg[18];
189  assign bw_word_shift[19] = (SHIFT_CTRL[19]) ? D_Q[19] : q_reg[19];
190  assign bw_word_shift[20] = (SHIFT_CTRL[20]) ? D_Q[20] : q_reg[20];
191  assign bw_word_shift[21] = (SHIFT_CTRL[21]) ? D_Q[21] : q_reg[21];
192  assign bw_word_shift[22] = (SHIFT_CTRL[22]) ? D_Q[22] : q_reg[22];
193  assign bw_word_shift[23] = (SHIFT_CTRL[23]) ? D_Q[23] : q_reg[23];
194  assign bw_word_shift[24] = (SHIFT_CTRL[24]) ? D_Q[24] : q_reg[24];
195  assign bw_word_shift[25] = (SHIFT_CTRL[25]) ? D_Q[25] : q_reg[25];
196  assign bw_word_shift[26] = (SHIFT_CTRL[26]) ? D_Q[26] : q_reg[26];
197  assign bw_word_shift[27] = (SHIFT_CTRL[27]) ? D_Q[27] : q_reg[27];
198  assign bw_word_shift[28] = (SHIFT_CTRL[28]) ? D_Q[28] : q_reg[28];
199  assign bw_word_shift[29] = (SHIFT_CTRL[29]) ? D_Q[29] : q_reg[29];
200  assign bw_word_shift[30] = (SHIFT_CTRL[30]) ? D_Q[30] : q_reg[30];
201  assign bw_word_shift[31] = (SHIFT_CTRL[31]) ? D_Q[31] : q_reg[31];
202
203  wire    [31:0]  d_sel;  //Data from BUS or previous REG
204
205  assign d_sel[31:0] = (SHIFT_WRITE) ? bw_word_shift[31:0] : D[31:0];
206
207          always@(posedge CLK)
208            if(CLKEN)
209               if((SEL & WRITE_EN & ~SHIFT_WRITE) | (SHIFT_WRITE & SHIFT_EN))
210                  q_reg <= d_sel[31:0];
211
212
213  assign Q[31:0] = q_reg[31:0];
214
215  endmodule
216
217
218  //
       ***********************************************************************************************************************

219  //NOISE GENERATOR CONTROLLER
220  //
       ***********************************************************************************************************************

221  //Noise Generator can work synchronously or asynchronously to Watermark Generator Circuit and Leakage Circuit.
222  //
223  //In synchronous mode Noise Generator waits until WGC and LC are operational, this means that it waits until
224  //watermark controller provides enable signals to WGC and LC. Since the input is the enable signal from the
       watermark controller
225  //the noise generator will start 1 clock cycle after WGC and LC. Using this mode it will be possible to predict the
       number of
226  //currently switching registers, in case of analysis of the strength of noise on the watermark performance.
227  //
228  //In asynchronous mode Noise Generator does not wait for WGC or LC, it starts as soon as possible after reset.
       Therefore,
229  //it will not be possible to predict how many registers are switching in any particular moment.
230  //
       ***********************************************************************************************************************

231
232  `timescale 1ns / 1ps
233
234  module NOISE_GEN_CONTROLLER (
235
236  input   wire            CLK,
237  input   wire            RESETn,
238
239  input   wire    [1:0]   NOISE_GEN_OP_MODE,          //Noise Generator Operation Mode (Synchronous or
       Asynchronous)
240
```

```
241 input  wire          WC_WGC_SHIFT_EN,          //Watermark Controller - Watermark Generation Circuit Shift
        Enable Out
242                                                //used for synchronizing the Noise Generator to WGC and LC
243 output wire          NOISE_GEN_SHIFT_ENO,      //Noise Generator Shift Enable Out
244 output wire          NOISE_SEED_SHIFT_ENO      //Noise Seed Shift Enable Out
245
246 );
247
248 //----------PARAMETERS----------
249 localparam      NOISE_GEN_MODE_SYNCH    = 2'b01;
250 localparam      NOISE_GEN_MODE_ASYNCH   = 2'b10;
251
252 //----------STATE MACHINE----------
253 reg     [1:0]   state;
254 reg     [1:0]   next_state;
255
256 //State parameters
257 localparam      S_MODECHOICE            = 2'b00;
258 localparam      S_MODE_SYNCH_WAIT       = 2'b01;
259 localparam      S_MODE_START            = 2'b10;
260
261
262 //Sequential Logic
263       always@(posedge CLK or negedge RESETn)
264          if(~RESETn)
265               state <= S_MODECHOICE;
266          else
267               state <= next_state;
268
269 //Combinational Logic
270       always@(*)
271          begin
272             next_state = state;
273             case(state)
274               S_MODECHOICE          :     if(NOISE_GEN_OP_MODE == NOISE_GEN_MODE_SYNCH)          next_state
      = S_MODE_SYNCH_WAIT;
275                                           else if(NOISE_GEN_OP_MODE == NOISE_GEN_MODE_ASYNCH)    next_state
      = S_MODE_START;
276                //SYNCHRONOUS MODE
277                //Wait for the WGC Shift Enable
278                S_MODE_SYNCH_WAIT     :     if(WC_WGC_SHIFT_EN)     next_state = S_MODE_START;
279
280                //START NOISE GENERATOR (JUMP HERE IN CASE OF ASYNCHRONOUS MODE)
281                S_MODE_START          :     ;
282
283                //DEFAULT
284                default               :     next_state = S_MODECHOICE;
285
286             endcase
287          end
288
289 reg            noise_gen_shift_en_out;
290 reg            noise_seed_shift_en_out;
291
292 reg            noise_gen_shift_en_out_q;
293 reg            noise_seed_shift_en_out_q;
294
295       always@(posedge CLK or negedge RESETn)
296          if(~RESETn)
297             begin
298               noise_gen_shift_en_out_q <= 1'b0;
299               noise_seed_shift_en_out_q <= 1'b0;
300             end
301          else
302             begin
303               noise_gen_shift_en_out_q <= noise_gen_shift_en_out;
304               noise_seed_shift_en_out_q <= noise_seed_shift_en_out;
305             end
306
307       always@(*)
308          begin
309             noise_gen_shift_en_out = noise_gen_shift_en_out_q;
310             noise_seed_shift_en_out = noise_seed_shift_en_out_q;
311             case(state)
312               S_MODE_START    :     begin
313                                           noise_gen_shift_en_out = 1'b1;
314                                           noise_seed_shift_en_out = 1'b1;
315                                     end
316             endcase
317          end
```

```
318
319 assign NOISE_GEN_SHIFT_ENO = noise_gen_shift_en_out_q;
320 assign NOISE_SEED_SHIFT_ENO = noise_seed_shift_en_out_q;
321
322 endmodule
```

B.2 Test Chip III

B.2.1 Brief Description

The watermark circuit modulates the architectural clock gate (except Bus Interface Unit clock) during (case I) or immediately after (case II) the pipeline flush, caused by Wait-for-Interrupt (WFI) request in ARM® Cortex®-A5 microprocessor. In case I, the activation of the watermark circuit increases the time required to flush the pipeline and depends on the bit pattern of a watermark sequence. In case II, the addition delay occurs between clock gating and power gating, due to an extra period required to generate the watermark power pattern.

B.2.2 Watermark Generation Circuit

The architecture of a WGC is the same as in test chips I and II. However, to reduce the area overhead the WGC has been reduced to 16 bits. Therefore, the configuration allows up to a 16-bit LFSR or 16-bit shift register.

B.2.3 Watermark Controller

The watermark circuit can be operated in two different modes, such as during WFI pipeline flush (case I) or after WFI pipeline flush (case II). In case I, the controller waits until Data Processing Unit (DPU) requests WFx pipeline flush, which is caused by the WFI instruction. If such occurs, the controller enables WGC and the inverted watermark sequence is generated. Once the pipeline has been flushed and DPU requests the architectural clock to be gated, the controller preloads the WGC registers with the data stored in $layer0_misc[31:16]$ register. This is to make sure watermark sequence starts from the same position next time WFI is issued. In case II, the controller waits until DPU requests the architectural clock to be gated. Upon receiving the signal, the WGC is activated and executes a single period of a sequence, i.e. if 6-bit LFSR is used 63 clock cycles will be required to perform clock modulation. Once the full watermark sequence rotation is complete, the WGC is stopped and controller waits for another clock gate request from the DPU. In this case the output from the watermark circuit is the original non-inverted watermark sequence.

The control registers are shown in Table B.3.

Table B.3: Watermark Circuit Working Registers (Chips I and II)

Address	Description	
0xF0D00000	layer0_misc	
	[15:1]	1 - indicates the LFSR operation (uses XOR gate)
		0 - indicates the shift register operation
	[31:16]	value stored initially and pre-loaded to the WGC register
0xF0D00004	layer1_misc	
	0	reset (active low)
	1	clock enable
	[3:2]	operating mode
		01 - clock modulation during WFI pipeline flush
		10 - clock modulation after WFI pipeline flush
	[9:8]	write/shift control
		01 - WGC shift operation (watermark active)
		10 - WGC write (initial data read)
	[19:16]	WGC multiplexer control

B.2.4 RTL

```
1  //*******************************************************************************************
2  // WATERMARK_CA5_CRICKET
3  //*******************************************************************************************
4  // Watermark block for Cortex-A5 CRICKETi chip spin. Operates in 2 active modes
5  //    1. active during the WFI pipeline flush
6  //    2. active after the pipeline has been flushed, before entering the STANDBY mode
7  //
8  //*******************************************************************************************
9
10 module WATERMARK_CA5_CRICKET (
11
12 // Inputs
13 input  wire          CLKIN,
14 input  wire  [31:0]  ahb_layer0_misc,      // AHB Layer0 misc register (0xF0C0 0008)
15 input  wire  [31:0]  ahb_layer1_misc,      // AHB Layer1 misc register (0xF0C0 000C)
16 input  wire          nxt_standby_is_wfi,   // Next STANDBY is due to WFI (used for both wmark modes)
17 input  wire          dpu_gate_clk_req,     // DPU request signal to architectural clock gate (after pipeline
       flush)
18 input  wire          dpu_wfx_int_req,      // DPU WFX request to flush the pipeline due to WFx (during
       pipeline flush)
19
20 // Outputs
21 output wire          wmark_gate_clk_req    // Watermark clock gate request
22 );
23
24 // --- Watermark Control ---
25 wire                  nreset;               // active low reset
26 wire                  clken;                // global clock enable
27 wire          [1:0]   opmode;               // watermark operation mode
28 wire                  wgc_shift_en;         // WGC shift enable
29 wire                  wgc_write_en;         // WGC write enable
30 wire                  wgc_preload;          // WGC preload flag for 'during the pipeline flush' mode
31 wire          [3:0]   wgc_mux_ctrl;         // WGC mux control
32
33 // --- Watermark Data ---
34 wire          [15:0]  wgc_d;                // WGC data
```

```
35  wire            [15:1]  wgc_srl_lfsr;           // WGC srl_lfsr bits
36
37  // --- Watermark Control allocation ---
38  // byte 0 - general / operation mode
39  assign nreset                   = ahb_layer1_misc[0];
40  assign clken                    = ahb_layer1_misc[1];
41  assign opmode           [1:0]   = ahb_layer1_misc[3:2];
42
43  // byte 1/2 - WGC control
44  assign wgc_shift_en             = ahb_layer1_misc[8];
45  assign wgc_write_en             = ahb_layer1_misc[9];
46
47  assign wgc_mux_ctrl     [3:0]   = ahb_layer1_misc[19:16];
48
49  // --- Watermark Data allocation ---
50  assign wgc_srl_lfsr     [15:1]  = ahb_layer0_misc[15:1];
51  assign wgc_d            [15:0]  = ahb_layer0_misc[31:16];
52
53  // --- Wires and Regs ---
54  wire                    wmark;                  // watermark sequence
55  wire                    ctrl_wgc_shift_en;      // watermark controller WGC shift en
56  wire                    ctrl_wgc_preload;       // watermark controller WGC preload
57  wire            [15:0]  wgc;                    // Watermark Generation Circuit output
58
59  reg                     wmarkrot;               // watermark sequence full rotation flag
60  wire            [15:0]  wmarkrot_nxor;
61  wire                    wmarkrot_and_1;
62  wire                    wmarkrot_and_2;
63  wire                    wmarkrot_and_3;
64  wire                    wmarkrot_and_4;
65  wire                    wmarkrot_and_5;
66  wire                    wmarkrot_and_6;
67  wire                    wmarkrot_and_7;
68  wire                    wmarkrot_and_8;
69  wire                    wmarkrot_and_9;
70  wire                    wmarkrot_and_10;
71  wire                    wmarkrot_and_11;
72  wire                    wmarkrot_and_12;
73  wire                    wmarkrot_and_13;
74  wire                    wmarkrot_and_14;
75  wire                    wmarkrot_and_15;
76
77  // watermark sequence full rotation flag
78  assign wmarkrot_nxor    [15:0]  = ~(wgc[15:0] ^ wgc_d[15:0]);
79  assign wmarkrot_and_1           = wmarkrot_nxor[0] & wmarkrot_nxor[1];
80  assign wmarkrot_and_2           = wmarkrot_and_1   & wmarkrot_nxor[2];
81  assign wmarkrot_and_3           = wmarkrot_and_2   & wmarkrot_nxor[3];
82  assign wmarkrot_and_4           = wmarkrot_and_3   & wmarkrot_nxor[4];
83  assign wmarkrot_and_5           = wmarkrot_and_4   & wmarkrot_nxor[5];
84  assign wmarkrot_and_6           = wmarkrot_and_5   & wmarkrot_nxor[6];
85  assign wmarkrot_and_7           = wmarkrot_and_6   & wmarkrot_nxor[7];
86  assign wmarkrot_and_8           = wmarkrot_and_7   & wmarkrot_nxor[8];
87  assign wmarkrot_and_9           = wmarkrot_and_8   & wmarkrot_nxor[9];
88  assign wmarkrot_and_10          = wmarkrot_and_9   & wmarkrot_nxor[10];
89  assign wmarkrot_and_11          = wmarkrot_and_10  & wmarkrot_nxor[11];
90  assign wmarkrot_and_12          = wmarkrot_and_11  & wmarkrot_nxor[12];
91  assign wmarkrot_and_13          = wmarkrot_and_12  & wmarkrot_nxor[13];
92  assign wmarkrot_and_14          = wmarkrot_and_13  & wmarkrot_nxor[14];
93  assign wmarkrot_and_15          = wmarkrot_and_14  & wmarkrot_nxor[15];
94
95      always@(*)
96          case(wgc_mux_ctrl)
97              4'b0001             : wmarkrot = wmarkrot_and_1;
98              4'b0010             : wmarkrot = wmarkrot_and_2;
99              4'b0011             : wmarkrot = wmarkrot_and_3;
100             4'b0100             : wmarkrot = wmarkrot_and_4;
101             4'b0101             : wmarkrot = wmarkrot_and_5;
102             4'b0110             : wmarkrot = wmarkrot_and_6;
103             4'b0111             : wmarkrot = wmarkrot_and_7;
104             4'b1000             : wmarkrot = wmarkrot_and_8;
105             4'b1001             : wmarkrot = wmarkrot_and_9;
106             4'b1010             : wmarkrot = wmarkrot_and_10;
107             4'b1011             : wmarkrot = wmarkrot_and_11;
108             4'b1100             : wmarkrot = wmarkrot_and_12;
109             4'b1101             : wmarkrot = wmarkrot_and_13;
110             4'b1110             : wmarkrot = wmarkrot_and_14;
111             4'b1111             : wmarkrot = wmarkrot_and_15;
112             default             : wmarkrot = 1'bx;
113         endcase
114
```

```
115 // --- Register Inputs
116 reg                     nxt_standby_is_wfi_q;
117 reg                     dpu_gate_clk_req_q;
118 reg                     dpu_wfx_int_req_q;
119
120   always@(posedge CLKIN or negedge nreset)
121     if (~nreset)
122       begin
123         nxt_standby_is_wfi_q <= 1'b0;
124         dpu_gate_clk_req_q   <= 1'b0;
125         dpu_wfx_int_req_q    <= 1'b0;
126       end
127     else
128       begin
129         nxt_standby_is_wfi_q <= nxt_standby_is_wfi;
130         dpu_gate_clk_req_q   <= dpu_gate_clk_req;
131         dpu_wfx_int_req_q    <= dpu_wfx_int_req;
132       end
133
134 WMARK_CTRL
135         watermark_controller
136         (
137                 .clk(CLKIN),
138                 .clken(clken),
139                 .nreset(nreset),
140                 .opmode(opmode),
141                 .wgc_shift_en_i(wgc_shift_en),
142                 .wmark(wmark),
143                 .wmarkrot(wmarkrot),
144                 .wgc_shift_en_o(ctrl_wgc_shift_en),
145                 .wgc_preload_o(ctrl_wgc_preload),
146                 .nxt_standby_is_wfi(nxt_standby_is_wfi_q),
147                 .dpu_gate_clk_req_i(dpu_gate_clk_req_q),
148                 .dpu_wfx_int_req(dpu_wfx_int_req_q),
149                 .wmark_gate_clk_req(wmark_gate_clk_req)
150         );
151
152 WGC
153         wat_gen_circ
154         (
155                 .clk(CLKIN),
156                 .clken(clken),
157                 .shift_en(ctrl_wgc_shift_en),
158                 .write_en(wgc_write_en | ctrl_wgc_preload),
159                 .mux_ctrl(wgc_mux_ctrl),
160                 .srl_lfsr(wgc_srl_lfsr),
161                 .d_i(wgc_d),
162                 .wmark(wmark),
163                 .wgc_o(wgc)
164         );
165
166 endmodule
167
168
169 //*****************************************************************************
170 // WATERMARK CIRCUIT CONTROLLER
171 //*****************************************************************************
172 //
173 //*****************************************************************************
174
175 `timescale 1ns / 1ps
176
177 module WMARK_CTRL (
178
179 // --- Watermark control ---
180 // Inputs
181 input   wire            clk,
182 input   wire            clken,                  // Clock enable
183 input   wire            nreset,                 // Active low reset
184 input   wire    [1:0]   opmode,                 // Watermark operation mode
185 input   wire            wgc_shift_en_i,         // WGC shift enable input control
186 input   wire            wmark,                  // watermark sequence
187 input   wire            wmarkrot,               // watermark full rotation flag
188
189 // Outputs
190 output  reg             wgc_shift_en_o,         // WGC shift enable output control
191 output  reg             wgc_preload_o,          // WGC preload
192
193 // --- CA5 control ---
194 // Inputs
```

```
195  input   wire            nxt_standby_is_wfi,     // Next STANDBY is due to WFI (used for both wmark modes)
196  input   wire            dpu_gate_clk_req_i,     // DPU request signal to architectural clock gate (after pipeline
         flush)
197  input   wire            dpu_wfx_int_req,        // DPU WFX request to flush the pipeline due to WFx (during
         pipeline flush)
198
199  // Outputs
200  output  reg             wmark_gate_clk_req      // Watermark clock gate request
201  );
202
203  // --- Operation Modes ---
204  localparam      WFI_WITH_FLUSH  = 2'b01;        // Watermark active during WFI pipeline flush
205  localparam      WFI_AFTER_FLUSH = 2'b10;        // Watermark active after WFI pipeline flush
206
207  // --- State Machine ---
208  localparam      ST_WAIT         = 3'b000;
209  localparam      ST_WITH_PRELOAD = 3'b001;
210  localparam      ST_WITH_IDLE    = 3'b010;
211  localparam      ST_WITH_ACTIVE  = 3'b011;
212  localparam      ST_AFTER_IDLE   = 3'b100;
213  localparam      ST_AFTER_ACTIVE = 3'b101;
214
215  reg     [2:0]   wmark_state;
216  reg     [2:0]   nxt_wmark_state;
217
218     always@(posedge clk or negedge nreset)
219        if(~nreset)
220           wmark_state <= ST_WAIT;
221        else
222           wmark_state <= nxt_wmark_state;
223
224     always@(*)
225        begin
226           // Defaults
227           wgc_shift_en_o          = 1'b0;
228           wgc_preload_o           = 1'b0;
229           wmark_gate_clk_req      = 1'b0;
230           nxt_wmark_state         = wmark_state;
231
232           case(wmark_state)
233              // --- WATERMARK ACTIVE DURING PIPELINE FLUSH ---
234              ST_WAIT: begin
235                 // WAIT - waits until WGC shift is enabled and one of the operation modes has been selected
236                 if(wgc_shift_en_i)
237                   begin
238                     if (opmode == WFI_WITH_FLUSH)                nxt_wmark_state = ST_WITH_IDLE;
239                     else if (opmode == WFI_AFTER_FLUSH)          nxt_wmark_state = ST_AFTER_IDLE;
240                     else                                         nxt_wmark_state = wmark_state;
241                   end
242              end
243              ST_WITH_PRELOAD: begin
244                 // PRELOAD WATERMARK (with) - after the pipeline has been flushed and the watermark
245                 // is expected to stop, the sequence may not be in the correct starting position
246                 // hence it may be difficult to merge multiple short power vectors together to
247                 // obtain a longer power vector for correlation. To make sure watermark starts
248                 // from the same sequence every time WFI is issued, the watermark is preloaded
249                 // with the data from misc register after the pipeline has been flushed and the
250                 // request to gate the architectural clock has been issued
251
252                 wgc_preload_o = 1'b1;
253                 nxt_wmark_state = ST_WITH_IDLE;
254              end
255              ST_WITH_IDLE: begin
256                 // IDLE (with) - waits until DPU requests a pipeline flush
257                 if (nxt_standby_is_wfi & dpu_wfx_int_req) nxt_wmark_state = ST_WITH_ACTIVE;
258              end
259              ST_WITH_ACTIVE: begin
260                 // ACTIVE (with) - activates watermark for the duration of the pipeline flush
261                 // hence modulates the entire CA5 clock (except reset synchronizers, interrupt etc.)
262                 // Waits until clock gate request is issued to the architectural clock gate
263                 if (dpu_gate_clk_req_i) nxt_wmark_state = ST_WITH_PRELOAD;
264                 else
265                 begin
266                    wgc_shift_en_o = 1'b1;
267                    wmark_gate_clk_req = ~wmark;
268                 end
269              end
270              // --- WATERMARK ACTIVE AFTER PIPELINE FLUSH ---
271              ST_AFTER_IDLE: begin
272                 // IDLE (after) - waits until DPU request clock to be gated and activates the watermark
```

```
273                 if (dpu_gate_clk_req_i) begin
274                     wgc_shift_en_o = 1'b1;
275                     wmark_gate_clk_req = wmark;
276                     nxt_wmark_state = ST_AFTER_ACTIVE;
277                 end
278             end
279             ST_AFTER_ACTIVE: begin
280                 // ACTIVE (after) - waits until watermark has done a single full rotation and disables it
281                 // also waits
282                 if (wmarkrot) begin
283                     if(~dpu_gate_clk_req_i) nxt_wmark_state = ST_AFTER_IDLE;
284                 end
285                 else begin
286                     wgc_shift_en_o = 1'b1;
287                     wmark_gate_clk_req = wmark;
288                 end
289             end
290             default: begin
291                 wgc_shift_en_o          = 1'bx;
292                 wgc_preload_o           = 1'bx;
293                 wmark_gate_clk_req      = 1'bx;
294                 nxt_wmark_state         = 3'bxxx;
295             end
296         endcase
297     end
298
299 endmodule
300
301
302 //*****************************************************************************
303 // WATERMARK GENERATION CIRCUIT MODULE
304 //*****************************************************************************
305 // Instantiates a Single Sequence Generator (SSG) module
306 //*****************************************************************************
307
308 `timescale 1ns / 1ps
309
310 module WGC (
311
312 input   wire            clk,
313 input   wire            clken,          // Clock Enable
314 input   wire            shift_en,       // WGC Shift Enable
315 input   wire            write_en,       // SSG Write Enable
316 input   wire    [3:0]   mux_ctrl,       // SSG Mux Control
317 input   wire    [15:1]  srl_lfsr,       // SSG Shift Register or LFSR control
318 input   wire    [15:0]  d_i,            // Input Data
319
320 output  wire            wmark,          // Watermark sequence
321 output  wire    [15:0]  wgc_o           // Watermark Generation Circuit output
322 );
323
324 SINGLE_SEQ_GEN ssg (
325                     .clk(clk),
326                     .clken(clken),
327                     .shift_en(shift_en),
328                     .write_en(write_en),
329                     .mux_ctrl(mux_ctrl),
330                     .ssg_i(d_i),
331                     .srl_lfsr(srl_lfsr),
332                     .ssgs_o(wmark),
333                     .ssg_o(wgc_o)
334                    );
335 //---
336
337 endmodule
338
339
340 //*****************************************************************************
341 // SINGLE SEQUENCE GENERATOR MODULE
342 //*****************************************************************************
343 // Can be set to implement LFSR, up to 16-bit (Galois LFSR) or up to 16-bit Shift Register
344 //
345 // In order to choose the feedback loop for the LFSR (and the output) and the output for
346 // the SRL, the 16-to-1 MUX is used with mux_ctrl control bits
347 // In order to implement the LFSR, the feedback is XORed with registers depending on the
348 // LFSR length to be implemented, therefore 15 srl_lfsr control bits are required
349 //*****************************************************************************
350
351 `timescale 1ns / 1ps
352
```

```
353 module SINGLE_SEQ_GEN (
354
355 input  wire          clk,
356 input  wire          clken,          // Clock enable
357 input  wire          shift_en,       // Shift Enable
358 input  wire          write_en,       // Write Enable
359 input  wire  [3:0]   mux_ctrl,       // Mux 16-to-1 Control bits
360 input  wire  [15:0]  ssg_i,          // Input Data
361 input  wire  [15:1]  srl_lfsr,       // SRL_LFSR Control bits
362
363 output wire          ssgs_o,         // Single bit output
364 output wire  [15:0]  ssg_o           // Single Sequence Generator output
365 );
366
367 // Bit Generators Outputs
368 wire   bitgen0;
369 wire   bitgen1;
370 wire   bitgen2;
371 wire   bitgen3;
372 wire   bitgen4;
373 wire   bitgen5;
374 wire   bitgen6;
375 wire   bitgen7;
376 wire   bitgen8;
377 wire   bitgen9;
378 wire   bitgen10;
379 wire   bitgen11;
380 wire   bitgen12;
381 wire   bitgen13;
382 wire   bitgen14;
383 wire   bitgen15;
384
385 // Output from 16-to-1 MUX, fed back to the 1st register and all feedback wires to XNOR gates
386 wire   feedback;
387
388 // REGISTER BANK INSTANTIATION
389 SEQ_GEN_BIT_WITHOUT_XOR reg0 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i[0]),.
        dprev_i(feedback),.d_o(bitgen0));
390
391 SEQ_GEN_BIT_WITH_XOR    reg1 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [1]),.dprev_i(bitgen0),.dlast_i(feedback),.srl_lfsr(srl_lfsr[1]),.d_o(bitgen1));
392
393 SEQ_GEN_BIT_WITH_XOR    reg2 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [2]),.dprev_i(bitgen1),.dlast_i(feedback),.srl_lfsr(srl_lfsr[2]),.d_o(bitgen2));
394
395 SEQ_GEN_BIT_WITH_XOR    reg3 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [3]),.dprev_i(bitgen2),.dlast_i(feedback),.srl_lfsr(srl_lfsr[3]),.d_o(bitgen3));
396
397 SEQ_GEN_BIT_WITH_XOR    reg4 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [4]),.dprev_i(bitgen3),.dlast_i(feedback),.srl_lfsr(srl_lfsr[4]),.d_o(bitgen4));
398
399 SEQ_GEN_BIT_WITH_XOR    reg5 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [5]),.dprev_i(bitgen4),.dlast_i(feedback),.srl_lfsr(srl_lfsr[5]),.d_o(bitgen5));
400
401 SEQ_GEN_BIT_WITH_XOR    reg6 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [6]),.dprev_i(bitgen5),.dlast_i(feedback),.srl_lfsr(srl_lfsr[6]),.d_o(bitgen6));
402
403 SEQ_GEN_BIT_WITH_XOR    reg7 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [7]),.dprev_i(bitgen6),.dlast_i(feedback),.srl_lfsr(srl_lfsr[7]),.d_o(bitgen7));
404
405 SEQ_GEN_BIT_WITH_XOR    reg8 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [8]),.dprev_i(bitgen7),.dlast_i(feedback),.srl_lfsr(srl_lfsr[8]),.d_o(bitgen8));
406
407 SEQ_GEN_BIT_WITH_XOR    reg9 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [9]),.dprev_i(bitgen8),.dlast_i(feedback),.srl_lfsr(srl_lfsr[9]),.d_o(bitgen9));
408
409 SEQ_GEN_BIT_WITH_XOR    reg10 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [10]),.dprev_i(bitgen9),.dlast_i(feedback),.srl_lfsr(srl_lfsr[10]),.d_o(bitgen10));
410
411 SEQ_GEN_BIT_WITH_XOR    reg11 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [11]),.dprev_i(bitgen10),.dlast_i(feedback),.srl_lfsr(srl_lfsr[11]),.d_o(bitgen11));
412
413 SEQ_GEN_BIT_WITH_XOR    reg12 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [12]),.dprev_i(bitgen11),.dlast_i(feedback),.srl_lfsr(srl_lfsr[12]),.d_o(bitgen12));
414
415 SEQ_GEN_BIT_WITH_XOR    reg13 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [13]),.dprev_i(bitgen12),.dlast_i(feedback),.srl_lfsr(srl_lfsr[13]),.d_o(bitgen13));
416
417 SEQ_GEN_BIT_WITH_XOR    reg14 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
        [14]),.dprev_i(bitgen13),.dlast_i(feedback),.srl_lfsr(srl_lfsr[14]),.d_o(bitgen14));
```

```
418
419 SEQ_GEN_BIT_WITH_XOR reg15 (.clk(clk),.clken(clken),.shift_en(shift_en),.write_en(write_en),.d_i(ssg_i
      [15]),.dprev_i(bitgen14),.dlast_i(feedback),.srl_lfsr(srl_lfsr[15]),.d_o(bitgen15));
420
421 //---
422
423
424 // FEEDBACK MUX INSTANTATION
425 wire [15:0] bitgen;
426
427 assign bitgen[15:0] = {bitgen15,
428                        bitgen14,
429                        bitgen13,
430                        bitgen12,
431                        bitgen11,
432                        bitgen10,
433                        bitgen9,
434                        bitgen8,
435                        bitgen7,
436                        bitgen6,
437                        bitgen5,
438                        bitgen4,
439                        bitgen3,
440                        bitgen2,
441                        bitgen1,
442                        bitgen0
443                       };
444
445 MUX16 ssg_mux (.d(bitgen[15:0]),.ctrl(mux_ctrl),.q(feedback));
446
447 // OUTPUT DATA
448 assign ssgs_o      = feedback;
449 assign ssg_o[15:0] = bitgen[15:0];
450
451 endmodule
452
453 //*****************************************************************************
454 // 1-bit REG with XOR gate
455 //*****************************************************************************
456 // Used to implement SRL/LFSR function (XOR gates used for flip-flops from 1st to 15th for LFSR)
457 // Also additional MUX is used to choose the data either from the misc registers
458 // or from the previous register
459 //
460 // The input data can be routed either from the misc registers or from the previous register
461 // When initial data comes from the misc registers shift_en is cleared and write_en is set
462 // If data comes from the previous register shift_en is set and write_en is cleared
463 //
464 // Since the whole 16-bit REG can be implemented as either SRL (shift register) or LFSR the
465 // additional control bit srl_lfsr specifies which mode the bit is currently in (1 for SRL,
466 // 0 for LFSR). In case of SRL mode the data is shifted from the previous register, while in case
467 // of the LFSR the data from the previous register is XORed with the bit from the last usable register
468 // (feedback loop)
469 //*****************************************************************************
470
471 `timescale 1ns / 1ps
472
473 module SEQ_GEN_BIT_WITH_XOR (
474
475 input  wire  clk,
476 input  wire  clken,          //Clock enable
477 input  wire  shift_en,       //Shift Enable
478 input  wire  write_en,       //Write Enable
479 input  wire  d_i,            //Input Data
480 input  wire  dprev_i,        //Data from previous REG
481 input  wire  dlast_i,        //Data from last REG
482 input  wire  srl_lfsr,       //Connected as shift register (SRL - 0) or LFSR (1);
483 output wire  d_o             //Data Out
484 );
485
486 wire d_srl_lfsr; //data connected directly from previous Q (SRL) or through XOR (LFSR)
487
488 assign d_srl_lfsr = (srl_lfsr) ? (dprev_i ^ dlast_i) : dprev_i;
489
490 wire d_sel; //data connected from the misc registers or d_srl_lfsr
491
492 assign d_sel = (shift_en) ? d_srl_lfsr : d_i;
493
494 reg seq_bit;
495
496         always@(posedge clk)
```

```
497             if(clken)
498               if(shift_en | write_en) seq_bit <= d_sel;
499
500   assign d_o = seq_bit;
501
502   endmodule
503
504
505   //*****************************************************************************
506   // 1-bit REG without XOR gate
507   //*****************************************************************************
508   // Used to implement SRL/LFSR function (XOR gate not needed as this would be 1st REG for LFSR)
509   // Also additional MUX is used to choose the data either from the misc registers
510   // or from the previous register
511   //
512   // The input data can be routed either from the misc registers or from the previous register
513   // When initial data comes from the misc registers shift_en is cleared and write_en is set
514   // If data comes from the previous register shift_en is set and write_en is cleared
515   //*****************************************************************************
516
517   `timescale 1ns / 1ps
518
519   module SEQ_GEN_BIT_WITHOUT_XOR (
520
521   input   wire    clk,
522   input   wire    clken,          //Clock enable
523   input   wire    shift_en,       //Shift Enable
524   input   wire    write_en,       //Write Enable
525   input   wire    d_i,            //Input Data
526   input   wire    dprev_i,        //Data from previous REG
527   output  wire    d_o             //Data Out
528   );
529
530   wire d_sel; // data from the misc registers or data input
531
532   assign d_sel = (shift_en) ? dprev_i : d_i;
533
534   reg seq_bit;
535
536           always@(posedge clk)
537             if(clken)
538               if(shift_en | write_en) seq_bit <= d_sel;
539
540   assign d_o = seq_bit;
541
542   endmodule
543
544
545   //*****************************************************************************
546   //16-to-1 MUX
547   //*****************************************************************************
548
549   `timescale 1ns / 1ps
550
551   module MUX16 (
552
553   input   wire    [15:0]  d,      //Input
554   input   wire    [3:0]   ctrl,   //Control
555   output  wire            q       //Output
556   );
557
558   assign q = ( (ctrl == 4'b0000) & d[0]) |
559                  ( (ctrl == 4'b0001) & d[1]) |
560                  ( (ctrl == 4'b0010) & d[2]) |
561                  ( (ctrl == 4'b0011) & d[3]) |
562                  ( (ctrl == 4'b0100) & d[4]) |
563                  ( (ctrl == 4'b0101) & d[5]) |
564                  ( (ctrl == 4'b0110) & d[6]) |
565                  ( (ctrl == 4'b0111) & d[7]) |
566                  ( (ctrl == 4'b1000) & d[8]) |
567                  ( (ctrl == 4'b1001) & d[9]) |
568                  ( (ctrl == 4'b1010) & d[10]) |
569                  ( (ctrl == 4'b1011) & d[11]) |
570                  ( (ctrl == 4'b1100) & d[12]) |
571                  ( (ctrl == 4'b1101) & d[13]) |
572                  ( (ctrl == 4'b1110) & d[14]) |
573                  ( (ctrl == 4'b1111) & d[15]);
574
575   endmodule
```

References

[1] VSI Alliance. VSI Alliance Architecture Document: Version 1.0. VSI Alliance, 1997.

[2] G.E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, 1965.

[3] Semiconductor Industry Association. The National Technology Roadmap For Semiconductors: Technology Needs. Semiconductor Industry Association, 1997.

[4] G. Qu and M. Potkonjak. *Intellectual Property Protection in VLSI Design: Theory and Practice*. Springer, 1st edition, 2003.

[5] Intellectual Property Protection Development Working Group. Intellectual Property Protection: Schemes, Alternatives and Discussion. VSI Alliance, White Paper, August 2001.

[6] R. Torrance and D. James. The State-of-the-Art in IC Reverse Engineering. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 5747 of *Lecture Notes in Computer Science*, pages 363–381. Springer Berlin / Heidelberg, 2009.

[7] Semiconductor Industry Association. International Technology Roadmap For Semiconductors: Design. Semiconductor Industry Association, 2003.

[8] G. Becker, M. Kasper, A. Moradi, and C. Paar. Side-Channel Based Watermarks for Integrated Circuits. In *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 30–35, June 2010.

[9] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis With a Leakage Model. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 3156 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg, 2004.

[10] R.S. Chakraborty and S. Bhunia. HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, October 2009.

[11] K. Lofstrom, W.R. Daasch, and D. Taylor. IC Identification Circuit Using Device Mismatch. In *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*, pages 372–373, 2000.

[12] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas. Identification and Authentication of Integrated Circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, 2004.

[13] F. Koushanfar and M. Potkonjak. CAD-based Security, Cryptography, and Digital Rights Management. In *Proceedings of Design Automation Conference (DAC)*, pages 268–269, June 2007.

[14] J.W. Lee, L. Daihyun, B. Gassend, G.E. Suh, M. van Dijk, and S. Devadas. A Technique to Build a Secret Key in Integrated Circuits For Identification and Authentication Applications. In *Proceedings of Symposium on VLSI Circuits, Digest of Technical Papers*, pages 176–179, June 2004.

[15] Y. Alkabani, F. Koushanfar, and M. Potkonjak. Remote Activation of ICs for Piracy Prevention and Digital Right Management. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 674–677, November 2007.

[16] F. Koushanfar. Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management. *IEEE Transactions on Information Forensics and Security*, 7(1):51–63, February 2012.

[17] Y. Alkabani, F. Koushanfar, N. Kiyavash, and M. Potkonjak. Trusted Integrated Circuits: A Nondestructive Hidden Characteristics Extraction Approach, 2008.

[18] G. Sun, Z. Gao, and Y. Xu. A Watermarking System for IP Protection by Buffer Insertion Technique. In *Proceedings of International Symposium on Quality Electronic Design (ISQED)*, pages 675–680, March 2006.

[19] A.B. Kahng, J. Lach, W.H. Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Constraint-Based Watermarking Techniques for Design IP Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(10):1236–1252, October 2001.

[20] M. Ni and Z. Gao. Constraint-Based Watermarking Technique for Hard IP Core Protection in Physical Layout Design Level. In *Proceedings of International Conference on Solid-State and Integrated Circuits Technology*, pages 1360–1363, 2004.

[21] X. Cai, Z. Gao, F. Bai, and Y. Xu. A Watermarking Technique for Hard IP Protection in Post-Layout Design Level. In *Proceedings of International Conference on ASIC (ASICON)*, pages 1317–1320, October 2007.

[22] N. Narayan, R.D. Newbould, J.D. Carothens, J.J. Rodriguez, and W.T. Holman. IP Protection for VLSI Designs via Watermarking of Routes. In *Proceedings of International ASIC/SOC Conference*, pages 406–410, September 2001.

[23] T. Nie, T. Kisaka, and M. Toyonaga. A Post Layout Watermarking Method for IP Protection. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 6206–6209, 2005.

[24] Y. Du, Y. Ding, Y. Chen, and Z. Gao. IP protection platform based on watermarking technique. In *Proceedings of International Symposium on Quality of Electronic Design (ISQED)*, pages 287–290, March 2009.

[25] G. Qu. Publicly Detectable Techniques for the Protection of Virtual Components. In *Proceedings of Design Automation Conference (DAC)*, pages 474–479, 2001.

[26] G. Qu. Publicly Detectable Watermarking for Intellectual Property Authentication in VLSI Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1363–1368, November 2002.

[27] S. Chen and C. Cheng. Tutorial on VLSI Partitioning. *VLSI Design*, 00(00):1–43, 2000.

[28] A.E. Caldwell, H. Choi, A.B. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J.L. Wong. Effective Iterative Techniques for Fingerprinting Design IP. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(2):208–215, February 2004.

[29] D. Kirovski and M. Potkonjak. Local Watermarks: Methodology and Application to Behavioral Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1277–1283, November 2003.

[30] M.M. Khan and S. Tragoudas. Rewiring for Watermarking Digital Circuit Netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1132–1137, July 2005.

[31] Aijiao Cui, Wei Liang, and Gang Qu. A low-overhead dynamic watermarking scheme on scan design for easy authentication. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 778–781, June 2014.

[32] A. Cui and C. Chang. Stego-Signature at Logic Synthesis Level for Digital Design IP Protection. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2006.

[33] A. Cui and C. Chang. Kernel Extraction for Watermarking Combinational Logic Networks. In *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pages 1023–1026, 2006.

[34] A.L. Oliveira. Robust Techniques For Watermarking Sequential Circuit Designs. In *Proceedings of Design Automation Conference (DAC)*, pages 837–842, June 1999.

[35] I. Torunoglu and E. Charbon. Watermarking-Based Copyright Protection of Sequential Functions. *IEEE Journal of Solid-State Circuits*, 35(3):434–440, March 2000.

[36] A. Abdel-Hamid, S. Tahar, and E.M. Aboulhamid. A Public-Key Watermarking Technique for IP Designs. In *Proceedings of Design, Automation and Test in Europe (DATE) Conference*, volume 1, pages 330–335, March 2005.

[37] Abdel-Hamid, A. and Tahar, S. and Aboulhamidm E.M. Finite State Machine IP Watermarking: A Tutorial. In *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 457–464, June 2006.

[38] A. Abdel-Hamid and S. Tahar. Fragile IP Watermarking Techniques. In *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 513–519, June 2008.

[39] A. Cui, C. Chang, S. Tahar, and A.T. Abdel-Hamid. A Robust FSM Watermarking Scheme for IP Protection of Sequential Circuit Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(5):678–690, May 2011.

[40] A. Cui, C. Chang, and L. Zhang. A Hybrid Watermarking Scheme for Sequential Functions. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2333–2336, May 2011.

[41] A. Cui and C. Chang. Intellectual Property Authentication by Watermarking Scan Chain in Design-for-Testability Flow. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2645–2648, May 2008.

[42] A. Cui, C. Chang, and S. Tahar. IP Watermarking Using Incremental Technology Mapping at Logic Synthesis Level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(9):1565–1570, September 2008.

[43] A. Cui and C. Chang. An Improved Publicly Detectable Watermarking Scheme Based on Scan Chain Ordering. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 29–32, May 2009.

[44] C. Chang and A. Cui. Synthesis-for-Testability Watermarking for Field Authentication of VLSI Intellectual Property. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(7):1618–1630, July 2010.

[45] J. Echavarria, A. Morales-Reyes, R. Cumplido, and M.A. Salido. Fsm merging and reduction for ip cores watermarking using genetic algorithms. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–7, Dec 2014.

[46] Y. Fan. Testing-Based Watermarking Techniques for Intellectual-Property Identification in SOC Design. *IEEE Transactions on Instrumentation and Measurement*, 57(3):467–479, March 2008.

[47] Y. Fan and H. Tsao. Watermarking Based IP Core Protection. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 5, pages 181–184, May 2003.

[48] Y. Fan and J. Shen. DFT-Based SoC/VLSI IP Protection and Digital Rights Management Platform. *IEEE Transactions on Instrumentation and Measurement*, 58(6):2026–2033, June 2009.

[49] E. Castillo, U. Meyer-Baese, A. Garcia, L. Parrilla, and A. Lloris. IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(5):578–591, May 2007.

[50] E. Castillo, L. Parrilla, A. Garcia, A. Lloris, and U. Meyer-Baese. Watermarking Strategies for RNS-Based System Intellectual Property Protection. In *IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 160–165, November 2005.

[51] E. Castillo. *RNS-Based Watermarking for IP Cores*. PhD thesis, University of Granada, Spain, Department of Electronics and Computer Technology, 2006.

[52] E. Castillo, L. Parrilla, A. Garcia, A. Loris, and U. Meyer-Baese. IPP Watermarking Technique for IP Core Protection on FPL Devices. In *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, August 2006.

[53] E. Castillo, A. Garcia, L. Parrilla, D.P. Morales, A. Lloris, and U. Meyer-Baese. Digital Signature Embedding Technique for IP Core Protection. In *Proceedings of 3rd Southern Conference on Programmable Logic (SPL)*, pages 143–148, February 2007.

[54] L. Parrilla, E. Castillo, A. Garcia, D. Gonzalez, A. Lloris, E. Todorovich, and E. Boemo. Protection of microprocessor-based cores for FPL devices. In *Proceedings of VI Southern Programmable Logic Conference (SPL)*, pages 15–20, March 2010.

[55] A. Garca L. Parrilla E. Todorovich E. Castillo, D. P. Morales and U. Meyer-Baese. Design Time Optimization for Hardware Watermarking Protection of HDL Designs. *The Scientific World Journal*, 2015, 2015.

[56] J.H. Daniel, D.F. Moore, and J.F. Walker. Focused Ion Beams For Microfabrication. In *Journal of Engineering Science & Education*, volume 7, pages 53–56, April 1998.

[57] I.W. Rangelow, T. Gotszalk, P. Grabiec, K. Edinger, and N. Abedinov. Thermal Nano-Probe. *Microelectronic Engineering*, 5758:737–748, 2000.

[58] D. Ziener and J. Teich. FPGA Core Watermarking Based on Power Signature Analysis. In *Proceedings of IEEE International Conference on Field Programmable Technology (FPT)*, pages 205–212, December 2006.

[59] D. Ziener and J. Teich. Power Signature Watermarking of IP Cores for FPGAs. *Journal of Signal Processing Systems*, 51:123–136, April 2008.

[60] D. Ziener, F. Baueregger, and J. Teich. Using the Power Side Channel of FPGAs for Communication. In *Proceedings of IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 237–244, May 2010.

[61] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Proceedings of International Cryptology Conference on Advances in Cryptology (CRYPTO)*, volume 1666 of *Lecture Notes in Computer Science*, pages 789–789. Springer Berlin / Heidelberg, 1999.

[62] Data Encryption Standard (DES). Technical report, U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, October 1999.

[63] P. Rakers, L. Connell, T. Collins, and D. Russell. Secure Contactless Smartcard ASIC with DPA Protection. *IEEE Journal of Solid-State Circuits*, 36(3):559–565, March 2001.

[64] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *Proceedings of USENIX Workshop on Smartcard Technology*, pages 151–162, May 1999.

[65] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Examining Smart-Card Security Under The Threat of Power Analysis Attacks. *IEEE Transactions on Computers*, 51(5):541–552, May 2002.

[66] A. Rastogi, K. Ganeshpure, and S. Kundu. A Study on Impact of Leakage Current on Dynamic Power. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1069–1072, May 2007.

[67] L. Lin and W. Burleson. Leakage-Based Differential Power Analysis (LDPA) on Sub-90nm CMOS Cryptosystems. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 252–255, May 2008.

[68] Moradi A. Side-channel leakage through static power should we care about in practice? Cryptology ePrint Archive, Report 2014/025, 2014. http://eprint.iacr.org/.

[69] T. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer Berlin Heidelberg, 2000.

[70] J.S. Coron, P. Kocher, and D. Naccache. Statistics and Secret Leakage. In *Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 157–173. Springer Berlin Heidelberg, 2001.

[71] C. Clavier, J.S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer Berlin Heidelberg, 2000.

[72] S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer Berlin Heidelberg, 2005.

[73] John Goodwin. *Novel Countermeasures and Techniques for Differential Power Analysis*. PhD thesis, University of Southampton, UK, School of Electronics and Computer Science, 2009.

[74] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson. Trojan Side-Channels: Lightweight Hardware Trojans Through Side-Channel Engineering. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 382–395, Berlin, Heidelberg, 2009. Springer-Verlag.

[75] J. Goodwin and P. Wilson. Power analysis detectable watermarks for protecting intellectual property. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2342–2345, March 2010.

[76] S. Armstrong. Significance Tests Harm Progress in Forecasting. *International Journal of Forecasting*, 23(2):321–327, 2007.

[77] J. Hunter and F. Schmidt. *Methods of Meta-Analysis: Correcting Error and Bias in Research Findings*. Sage Publications, 2nd edition, 2004.

[78] F. Schmidt. Statistical Significance Testing and Cumulative Knowledge in Psychology: Implications for Training of Researchers. *Psychological Methods*, 1(2):115–129, 1996.

[79] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. Trustworthy Hardware: Identifying and Classifying Hardware Trojans. *Computer*, 43(10):39–46, October 2010.

[80] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan Detection using IC Fingerprinting. In *Proceedings of IEEE Symposium on Security and Privacy (SP)*, pages 296–310, May 2007.

[81] X. Zhang and M. Tehranipoor. RON: An On-Chip Ring Oscillator Network For Hardware Trojan Detection. In *Proceedings of Design, Automation Test in Europe (DATE) Conference*, pages 1–6, March 2011.

[82] L. Bohy, M. Neve, D. Samyde, and J.J. Quisquater. Principal and Independent Component Analysis for Crypto-Systems with Hardware Unmasked Units. In *Proceedings of International Conference on Research in Smart Cards, Smart Card Programming and Security (E-smart)*, 2003.

[83] F.P. Preparata and S.J. Hong. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. *Communications of the ACM*, 20(2):87–93, February 1977.

[84] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic. Detecting Trojans Through Leakage Current Analysis Using Multiple Supply Pad I_{DDQs}. *IEEE Transactions on Information Forensics and Security*, 5(4):893–904, December 2010.

[85] R.M. Rad, W. Xiaoxiao, M. Tehranipoor, and J. Plusquellic. Power Supply Signal Calibration Techniques For Improving Detection Resolution To Hardware Trojans. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 632–639, November 2008.

[86] D. Acharyya and J. Plusquellic. Hardware Results Demonstrating Defect Detection Using Power Supply Signal Measurements. In *Proceedings of IEEE VLSI Test Symposium*, pages 433–438, May 2005.

[87] S. Narasimhan, D. Du, R.S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia. Multiple-Parameter Side-Channel Analysis: A Non-Invasive Hardware Trojan Detection Approach. In *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 13–18, June 2010.

[88] S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia. Hardware Trojan Detection by Multiple-Parameter Side-Channel Analysis. *IEEE Transactions on Computers*, 62(11):2183–2195, August 2012.

[89] C. Lamech, R.M. Rad, M. Tehranipoor, and J. Plusquellic. An Experimental Analysis of Power and Delay Signal-to-Noise Requirements for Detecting Trojans and Methods for Achieving the Required Detection Sensitivities. *IEEE Transactions on Information Forensics and Security*, 6(3):1170–1179, September 2011.

[90] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 104–113, London, UK, UK, 1996. Springer-Verlag.

[91] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestre, J.J. Quisquater, and J.L. Willems. A Practical Implementation of the Timing Attack. In *Proceedings of International Conference on Smart Card Research and Applications (CARDIS)*, volume 1820 of *Lecture Notes in Computer Science*, pages 167–182. Springer Berlin Heidelberg, 2000.

[92] J.J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In *Proceedings of International Conference on Research in Smart Cards, Smart Card Programming and Security (E-smart)*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer Berlin Heidelberg, 2001.

[93] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 251–261, London, UK, 2001. Springer-Verlag.

[94] D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The em side-channel(s). In *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer Berlin Heidelberg, 2003.

[95] L. Sauvage, S. Guilley, and Y. Mathieu. Electromagnetic Radiations of FPGAs: High Spatial Resolution Cartography and Attack on a Cryptographic Module. *ACM Transactions on Reconfigurable Technology and Systems*, 2(1):1–24, March 2009.

[96] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4^{th} edition, 2010.

[97] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi. *Low Power Methodology Manual: For System-on-Chip Design*. Springer, 2007.

[98] Agilent Technologies InfiniiVision 6000 Series Oscilloscopes. Data sheet, Agilent Technologies, December 2012. 5989-2000EN.

[99] A. Abdel-Hamid, S. Tahar, and E.M. Aboulhamid. IP Watermarking Techniques: Survey and Comparison. In *Proceedings of IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 60–65, June-July 2003.

[100] A. Abdel-Hamid, S. Tahar, and E.M. Aboulhamid. A Survey on IP Watermarking Techniques. *Design Automation for Embedded Systems*, 9:211–227, 2004.

[101] Q. Li, N. Memon, and H.T. Sencar. Security Issues in Watermarking Applications - A Deeper Look. In *Proceedings of ACM International Workshop on Contents Protection and Security (MCPS)*, pages 23–28, New York, NY, USA, 2006. ACM.

[102] P. Alfke. Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators. Tech. rep., Xilinx Corporation, July 1996. xAPP052 (v1.1).

[103] R.H. Barker. Group Synchronizing of Binary Digital Sequences. In *Communication Theory*, pages 273–287. Butterworth, London, UK, 1953.

[104] M. Al Mamun, M.M. Hossain, M.Z. Alam, M.S.U. Zaman, A. Ahmmed, and T. Kanij. Performance Evaluation of Pseudo-Code of Multi-user Environment

CDMA Technology. *Proceedings of International Conference on Convergence Information Technology*, 0:1887–1892, 2007.

[105] M. Friese. Polyphase Barker Sequences Up To Length 36. *IEEE Transactions on Information Theory*, 42(4):1248–1250, July 1996.

[106] Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet. Product specification, Xilinx Corporation, June 2011. DS083 (v5.0).

[107] R. York. Benchmarking in Context: Dhrystone. ARM, White Paper, March 2002.

[108] G.T. Becker et al. Detecting Software Theft in Embedded Systems: A Side-Channel Approach. *TIFS*, 7(4):1144–1154, 2012.

[109] R.L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. Internet Activities Board, April 1992.

[110] E. Charbon et al. Watermarking Techniques for Electronic Circuit Design. volume 2613 of *Lecture Notes in Computer Science*, pages 147–169. 2003.

[111] J. Kufel, P. Wilson, S. Hill, B.M. Al-Hashimi, P.N. Whatmough, and J. Myers. Clock-modulation based watermark for protection of embedded processors. In *DATE*, pages 1–6, March 2014.

[112] F. Emnett and M. Biegel. Power Reduction Through RTL Clock Gating. In *Proceedings of Synopsys Users Group (SNUG) Conference*, 2000.

[113] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang. A Dual-Core Multi-Threaded Xeon Processor with 16MB L3 Cache. In *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, pages 315–324, February 2006.

[114] V.G. Oklobdzija, V.M. Stojanovic, D.M. Markovic, and N.M. Nedovic. *Digital System Clocking: High-Performance and Low-Power Aspects*. Wiley, 2005.

[115] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, September 1996.

[116] S. Wimer and I. Koren. The Optimal Fan-Out of Clock Network for Power Minimization by Adaptive Gating. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(10):1772–1780, October 2012.

[117] S. Huda, M. Mallick, and J.H. Anderson. Clock Gating Architectures for FPGA Power Reduction. In *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, pages 112–118, September 2009.

[118] Clock Gating Methodology for Power and CTS QoR. Technical report, Synopsys.

[119] J.G. Mueller and R.A. Saleh. Autonomous, Multilevel Ring Tuning Scheme for Post-Silicon Active Clock Deskewing Over Intra-Die Variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(6):973–986, June 2011.

[120] R. Chaturvedi and H. Jiang. Buffered Clock Tree for High Quality IC Design. In *Proceedings of International Symposium on Quality Electronic Design (ISQED)*, pages 381–386, 2004.

[121] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-V Power Supply High-Speed Digital Circuit Technology With Multithreshold-Voltage CMOS. *IEEE Journal of Solid-State Circuits*, 30(8):847–854, 1995.

[122] Cortex-A5: Technical Reference Manual. Technical reference manual, ARM, 2010. r0p1.

[123] Infiniium DSO80000B Series Oscilloscopes and InfiniiMax Series Probes: 2GHz to 13GHz Real-time Oscilloscope Measurement Systems. Data sheet, Agilent Technologies, December 2007. 5989-4604EN.

[124] Steve Gunn. Support Vector Machines for Classification and Regression. Technical report, Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, University of Southampton, UK, May 1998.