

FP7-ICT / FET-OPEN – 309129 / i-RISC

D6.1

## Report on Reliability Aware Synthesis and LDPC Decoders Built with Unreliable Components

Editor:	Sorin Cotofana (TUD)
Deliverable nature:	Public
Due date (DoW):	January 31, 2015
Extended due date:	March 31, 2015
Delivery date:	April 20, 2015
Version	1.0
Total number of pages:	110
Reviewed by:	i-RISC members
Keywords:	LDPC, Stochastic Decoder, Gallager-B, Gradient Descent Bit Flipping (GDBF), Probabilistic GDBF, Min-Sum (MS), Self-Corrected MS, Finite Alphabet Iterative Decoder (FAID), Fault-Tolerance, Voltage Scaling, Fault Injection, Error Profile Characterization.

### Abstract

This deliverable presents an overview of the work carried out in relation to Work Package 6 (WP6) during the second year of the i-RISC project. The first part of the deliverable reports on the implementation of a reliability aware synthesis tool and its evaluation on a set of benchmark circuits (Task 6.3). In particular, we propose an integrated design flow, which combines all the up to date developed i-RISC custom tools together with widely used tools in the circuit design industry. The second part of the deliverable reports on the fault tolerance assessment of state-of-the-art and i-RISC proposed LDPC decoders (Task 6.1). A number of seven LDPC decoders are implemented in VHDL/Verilog and are exposed to external aggression via voltage scaling or simulated fault injection. The error correction performance of the implemented decoders is evaluated for different aggression profiles, substantiating their fault-tolerance capabilities and the resulting benefits in terms of energy consumption and throughput.

## List of Authors

Participant	Author	E-Mail
CEA	Valentin Savin	<a href="mailto:valentin.savin@cea.fr">valentin.savin@cea.fr</a>
ENSEA	David Declercq	<a href="mailto:declercq@ensea.fr">declercq@ensea.fr</a>
	Fakhreddine Ghaffari	<a href="mailto:fakhreddine.ghaffari@ensea.fr">fakhreddine.ghaffari@ensea.fr</a>
	Khoa Le	<a href="mailto:le.khoa@ensea.fr">le.khoa@ensea.fr</a>
TU-Delft	Nicoleta Cucu-Laurenciu	<a href="mailto:N.CucuLaurenciu@tudelft.nl">N.CucuLaurenciu@tudelft.nl</a>
	Thomas Marconi	<a href="mailto:T.Marconi@tudelft.nl">T.Marconi@tudelft.nl</a>
	Sorin Cotofana	<a href="mailto:S.D.Cotofana@tudelft.nl">S.D.Cotofana@tudelft.nl</a>
UPT	Alexandru Amaricai	<a href="mailto:amaricai@cs.upt.ro">amaricai@cs.upt.ro</a>
	Dan Dutescu	<a href="mailto:dan.dutescu@student.upt.ro">dan.dutescu@student.upt.ro</a>
	Ioana Mot	<a href="mailto:ioana.mot@student.upt.ro">ioana.mot@student.upt.ro</a>
UCC	Satish Kumar	<a href="mailto:sagrand@ue.ucc.ie">sagrand@ue.ucc.ie</a>
	Bo Yang	<a href="mailto:bo.yang@uemail.ucc.ie">bo.yang@uemail.ucc.ie</a>
	Emanuel Popovici	<a href="mailto:E.Popovici@ucc.ie">E.Popovici@ucc.ie</a>

## Table of Contents

<b>List of Dissemination Activities</b> .....	<b>5</b>
<b>List of Figures</b> .....	<b>6</b>
<b>List of Tables</b> .....	<b>10</b>
<b>Abbreviations</b> .....	<b>11</b>
<b>1. Executive Summary</b> .....	<b>14</b>
<b>2. Implementation of the i-RISC Reliability-Aware Synthesis Tool</b> .....	<b>17</b>
2.1. Introduction.....	17
2.2. The Tool Chain – Complete CAD Framework .....	18
2.2.1. Circuit Representation and Modification .....	19
2.2.2. Gate Level Logic Simulation.....	20
2.2.3. Reliability Estimation and Analysis .....	21
2.2.4. Multi Objective Optimization .....	22
2.2.5. Fault Tolerant Graph Augmentation .....	23
2.3. A Case Study .....	24
2.4. Conclusion .....	25
<b>3. Evaluation of LDPC Decoders in Fault Inducing Environments</b> .....	<b>26</b>
3.1. Introduction.....	26
3.2. Voltage Scaling Evaluation Framework .....	27
3.3. Fault Injection Evaluation Framework .....	31
3.3.1. Fault Simulation Framework .....	32
3.3.2. Decoder Basic Block Error Profile Characterization.....	33
3.3.3. Fault Map Generation .....	45
3.4. Utilized LDPC Codes.....	49
3.4.1. Protograph Based LDPC Construction .....	49
3.4.2. i-RISC Set of Matrices .....	50
3.5. Decoder Architecture, Organization, and Implementation .....	51
3.5.1. Min-Sum (MS).....	51
3.5.2. Self-Correcting Min-Sum (SCMS).....	55
3.5.3. Finite Alphabet Iterative Decoder (FAID) .....	56
3.5.4. Stochastic Decoder (SD) .....	57
3.5.5. Gradient Descent Bit-Flipping (GDBF) and Probabilistic GDBF (PGDBF).....	62
3.5.6. Gallager-B with Extended Alphabet (GB) .....	68
3.6. Performance Evaluation and Comparison.....	75
3.6.1. Voltage Scaling .....	75
3.6.2. Fault Injection.....	96
3.7. Conclusion .....	104
<b>4. General Conclusions and Next Steps</b> .....	<b>106</b>

**Appendix 1: Quasi-Cyclic Description of the LDPC Codes .....110**

## List of Dissemination Activities

### Published Papers

**[P1]** O. Boncalo, A. Amaricai, A. Hera, and V. Savin, "Cost Efficient FPGA Layered LDPC Decoder with Serial AP-LLR Processing", *International Conference on Field Programmable Logic and Applications (FPL)*, Munich, Germany, September 2014.

**[P2]** T. Marconi, C Spagnol, E Popovici, and S.D. Cotofana, "Towards Energy Effective LDPC Decoding by Exploiting Channel Noise Variability", *22<sup>nd</sup> IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 2014)*, Playa del Carmen, Mexico, 6-8 October 2014.

**[P3]** K. Le, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis, and B. Vasic, "Efficient Realization of Probabilistic Gradient Descent Bit Flipping Decoders", *IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015.

**[P4]** T. Marconi and S.D. Cotofana, "Dynamic Bitstream Length Scaling Energy Effective Stochastic LDPC Decoding", *25<sup>th</sup> edition of ACM's Great Lakes VLSI Symposium (GLSVLSI)*, Pittsburgh, Pennsylvania, USA, May 2015.

**[P5]**, N. Cucu-Laurenciu and S.D. Cotofana, "Low Cost and Energy, Thermal Noise Driven, Probability Modulated Random Number Generator", *IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015.

### Submitted Papers

**[S1]** S. Grandhi, Bo Yang, C. Spagnol, E. Popovici, S.D. Cotofana, "A Systematic Approach for Combinational Circuit Reliability Estimation", *Digital System Design (DSD)*, 2015.

**[S2]** O. Boncalo, A. Amaricai, "Memory efficient FPGA implementation for flooded LDPC decoder", *Field Programmable Logic and Applications (FPL)*, 2015.

### In Preparation

2 papers reporting our findings on LDPC decoder evaluation by means of voltage scaling and simulated fault injection, respectively.

## List of Figures

Figure 2-1: Reliability Aware Synthesis Tool Sub-Branches .....	18
Figure 2-2: Reliability Aware Synthesis Tool - The Complete Framework .....	19
Figure 2-3: Different Formats of Circuit Representation .....	20
Figure 2-4: Timing and Power Analysis Flow .....	21
Figure 2-5: Reliability Computation and Analysis Flow .....	22
Figure 2-6: Reliability Aware Logic Optimization Tool Flow .....	23
Figure 2-7: CPE based Graph Augmentation .....	23
Figure 3-1: Experimental Hardware Platform .....	28
Figure 3-2: Top Level Representation of the LDPC Testbed .....	28
Figure 3-3: Simulated Fault Injection Framework .....	32
Figure 3-4: Saboteur Insertion in Combinational Logic .....	33
Figure 3-5: Fault Tolerance Aware Design Flow at RTL/gate Abstraction Level .....	34
Figure 3-6: SC Propagation Delay PDF .....	35
Figure 3-7: Output Propagation Delay IG PDF .....	36
Figure 3-8: Output Propagation Delay IG CDF .....	36
Figure 3-9: Circuit POs Delay CDFs .....	37
Figure 3-10: MS Decoder CNU Combinational Stage 1 .....	38
Figure 3-11: MS Decoder CNU Combinational Stage 2 .....	38
Figure 3-12: MS Decoder CNU Combinational Stage 3 .....	38
Figure 3-13: MS Decoder VNU Combinational Stage 1 .....	39
Figure 3-14: MS Decoder VNU Combinational Stage 2 .....	39
Figure 3-15: MS Decoder VNU Combinational Stage 3 .....	39
Figure 3-16: MS Decoder VNU Combinational Stage 4 .....	39
Figure 3-17: MS Decoder Barrel Shifter .....	39
Figure 3-18: MS Decoder Memory Block .....	39
Figure 3-19: SCMS Decoder VNU Combinational Stage 1 .....	40
Figure 3-20: SCMS Decoder VNU Combinational Stage 2 .....	40
Figure 3-21: SCMS Decoder VNU Combinational Stage 3 .....	40
Figure 3-22: SCMS Decoder VNU Combinational Stage 4 .....	40
Figure 3-23: SCMS Decoder VNU Combinational Stage 5 .....	40
Figure 3-24: FAID Decoder VNU Combinational Stage 1 .....	41
Figure 3-25: FAID Decoder VNU Combinational Stage 2 .....	41
Figure 3-26: FAID Decoder VNU LUT .....	41
Figure 3-27: FAID Decoder VNU Out Stage .....	41
Figure 3-28: SD Decoder VNU Combinational Stage .....	41
Figure 3-29: SD Decoder CNU Combinational Stage .....	41
Figure 3-30: SD Decoder RNG .....	42
Figure 3-31: Number of Active Fault Locations in Processing Units .....	46
Figure 3-32: Ratio between Active Fault Locations and Total Number of Sabotaged Wires in Considered Blocks .....	47
Figure 3-33: Average Probability of Errors in the Considered Blocks (Logarithmic Scale) .....	47
Figure 3-34: Average Number of per Iteration Injected Faults (Logarithmic Scale) .....	48
Figure 3-35: Average Number of per Iteration Injected Faults (Zoom - Linear Plot) .....	48

Figure 3-36: Protographs for $(dv = 12, dc = 24)$ and $(dv = 12, dc = 48)$ LDPC Codes.....	51
Figure 3-37: Flooded MS Architecture .....	53
Figure 3-38: Flooded SCMS Architecture .....	55
Figure 3-39: Architecture of a Fully Parallel Stochastic LDPC Decoder .....	57
Figure 3-40: Probability to Stochastic Converter (P2SB) .....	58
Figure 3-41: The $VNn, m$ Module.....	59
Figure 3-42: The PCHD Module .....	60
Figure 3-43: The $CNm, n$ Module.....	61
Figure 3-44: The Parity Check $cm$ Module .....	61
Figure 3-45: Global Architecture of PGDBF Compared to the Original GBDF .....	62
Figure 3-46: Architecture of VNU in PGDBF .....	65
Figure 3-47: Architecture of Intrinsic-Valued Random Generator .....	66
Figure 3-48: The Multiplication Coefficient $K$ as a Function of the BSC Crossover .....	67
Figure 3-49: Input Data Format.....	69
Figure 3-50: Block Diagram Representation of the Gallager-B Decoder.....	70
Figure 3-51: Bit Node Block Diagram .....	72
Figure 3-52: Bit Node Architecture .....	73
Figure 3-53: Block A Implementation.....	73
Figure 3-54: Block B Implementation.....	74
Figure 3-55: Check Node Block Diagram .....	74
Figure 3-56: Check Node Architecture .....	75
Figure 3-57: BER of Stochastic Decoder under BSC.....	78
Figure 3-58: FER of Stochastic Decoder under BSC.....	78
Figure 3-59: Average Number of Iterations of Stochastic Decoder under BSC.....	78
Figure 3-60: Energy/Bit Normalized to BER of Stochastic Decoder under BSC.....	78
Figure 3-61: Energy/Bit Normalized to FER of Stochastic Decoder under BSC .....	78
Figure 3-62: Throughput Normalized to BER of Stochastic Decoder under BSC.....	79
Figure 3-63: Throughput Normalized to FER of Stochastic Decoder under BSC .....	79
Figure 3-64: BER of Gallager B Decoder under BSC.....	80
Figure 3-65: FER of Gallager B Decoder under BSC.....	80
Figure 3-66: Average Number of Iterations of Gallager B Decoder under BSC.....	80
Figure 3-67: Energy/Bit Normalized to BER of Gallager B Decoder under BSC.....	80
Figure 3-68: Energy/Bit Normalized to FER of Gallager B Decoder under BSC .....	80
Figure 3-69: Throughput Normalized to BER of Gallager B Decoder under BSC.....	81
Figure 3-70: Throughput Normalized to FER of Gallager B Decoder under BSC .....	81
Figure 3-71: BER of GDBF Decoder under BSC .....	81
Figure 3-72: FER of GDBF Decoder under BSC .....	81
Figure 3-73: Average Number of Iterations of GDBF Decoder under BSC .....	82
Figure 3-74: Energy/Bit Normalized to BER of GDBF Decoder under BSC .....	82
Figure 3-75: Energy/Bit Normalized to FER of GDBF Decoder under BSC.....	82
Figure 3-76: Throughput Normalized to BER of GDBF Decoder under BSC .....	82
Figure 3-77: Throughput Normalized to FER of GDBF Decoder under BSC.....	82
Figure 3-78: BER of PGDBF Decoder under BSC .....	83
Figure 3-79: FER of PGDBF Decoder under BSC .....	83
Figure 3-80: Average Number of Iterations of PGDBF Decoder under BSC .....	83

Figure 3-81: Energy/Bit Normalized to BER of PGDBF Decoder under BSC .....	84
Figure 3-82: Energy/Bit Normalized to FER of PGDBF Decoder under BSC.....	84
Figure 3-83: Throughput Normalized to BER of PGDBF Decoder under BSC .....	84
Figure 3-84: Throughput Normalized to FER of PGDBF Decoder under BSC .....	84
Figure 3-85: BER of MS Decoder under BSC.....	85
Figure 3-86: FER of MS Decoder under BSC .....	85
Figure 3-87: Average Number of Iterations of MS Decoder under BSC.....	85
Figure 3-88: Energy/Bit Normalized to BER of MS Decoder under BSC.....	86
Figure 3-89: Energy/Bit Normalized to FER of MS Decoder under BSC .....	86
Figure 3-90: Throughput Normalized to BER of MS Decoder under BSC.....	86
Figure 3-91: Throughput Normalized to FER of MS Decoder under BSC .....	86
Figure 3-92: BER of MSnoET Decoder under BSC.....	86
Figure 3-93: FER of MSnoET Decoder under BSC .....	86
Figure 3-94: Energy/Bit Normalized to BER of MSnoET Decoder under BSC.....	87
Figure 3-95: Energy/Bit Normalized to FER of MSnoET Decoder under BSC .....	87
Figure 3-96: Throughput Normalized to BER of MSnoET Decoder under BSC.....	87
Figure 3-97: Throughput Normalized to FER of MSnoET Decoder under BSC .....	87
Figure 3-98: BER of SCMS Decoder under BSC.....	88
Figure 3-99: FER of SCMS Decoder under BSC .....	88
Figure 3-100: Average Number of Iterations of SCMS Decoder under BSC.....	88
Figure 3-101: Energy/Bit Normalized to BER of SCMS Decoder under BSC.....	89
Figure 3-102: Energy/Bit Normalized to FER of SCMS Decoder under BSC .....	89
Figure 3-103: Throughput Normalized to BER of SCMS Decoder under BSC.....	89
Figure 3-104: Throughput Normalized to FER of SCMS Decoder under BSC .....	89
Figure 3-105: BER of SCMSnoET Decoder under BSC.....	89
Figure 3-106: FER of SCMSnoET Decoder under BSC .....	89
Figure 3-107: Energy/Bit Normalized to BER of SCMSnoET Decoder under BSC.....	90
Figure 3-108: Energy/Bit Normalized to FER of SCMSnoET Decoder under BSC .....	90
Figure 3-109: Throughput Normalized to BER of SCMSnoET Decoder under BSC.....	90
Figure 3-110: Throughput Normalized to FER of SCMSnoET Decoder under BSC .....	90
Figure 3-111: BER of FAID Decoder under BSC .....	91
Figure 3-112: FER of FAID Decoder under BSC.....	91
Figure 3-113: Average Number of Iterations of FAID Decoder under BSC.....	92
Figure 3-114: Energy/Bit Normalized to BER of FAID Decoder under BSC.....	92
Figure 3-115: Energy/Bit Normalized to FER of FAID Decoder under BSC.....	92
Figure 3-116: Throughput normalized to BER of FAID Decoder under BSC .....	92
Figure 3-117: Throughput Normalized to FER of FAID Decoder under BSC.....	92
Figure 3-118: BER of FAIDnoET Decoder under BSC .....	93
Figure 3-119: FER of FAIDnoET Decoder under BSC.....	93
Figure 3-120: Energy/Bit Normalized to BER of FAIDnoET Decoder under BSC.....	93
Figure 3-121: Energy/Bit Normalized to FER of FAIDnoET Decoder under BSC.....	93
Figure 3-122: Throughput Normalized to BER of FAIDnoET Decoder under BSC .....	93
Figure 3-123: Throughput Normalized to FER of FAIDnoET Decoder under BSC.....	93
Figure 3-124: Average Number of Iterations for Faulty MS under BI-AWGN .....	96
Figure 3-125: BER for Faulty MS under BI-AWGN .....	97

Figure 3-126: FER for Faulty MS under BI-AWGN ..... 97

Figure 3-127: Average Number of Iterations for Faulty SCMS under BI-AWGN with Errors Injected in the Two Additional Memories..... 97

Figure 3-128: BER for Faulty SCMS under BI-AWGN with Errors Injected in the Two Additional Memories ..... 98

Figure 3-129: FER for Faulty SCMS under BI-AWGN with Errors Injected in the Two Additional Memories ..... 98

Figure 3-130: Average Number of Iterations for Faulty SCMS under BI-AWGN with No Errors Injected in the Two Additional Memories..... 98

Figure 3-131: BER for Faulty SCMS under BI-AWGN with No Errors Injected in the Two Additional Memories ..... 99

Figure 3-132: FER for Faulty SCMS under BI-AWGN with No Errors Injected in the Two Additional Memories ..... 99

Figure 3-133: Average Number of Iterations for Faulty MS under BSC ..... 100

Figure 3-134: BER for Faulty MS under BSC ..... 100

Figure 3-135: FER for Faulty MS under BSC..... 100

Figure 3-136: Average Number of Iterations for Faulty SCMS under BSC with Errors Injected in All Memories ..... 101

Figure 3-137: BER for Faulty SCMS under BSC with Errors Injected in All Memories ..... 101

Figure 3-138: FER for Faulty SCMS under BSC with Errors Injected in All Memories ..... 101

Figure 3-139: Average Number of Iterations for Faulty SCMS under BSC with No Errors Injected in the Two Additional Memories ..... 102

Figure 3-140: BER for Faulty SCMS under BSC with No Errors Injected in the Two Additional Memories ..... 102

Figure 3-141: FER for Faulty SCMS under BSC with No Errors Injected in the Two Additional Memories ..... 102

Figure 3-142: Average Number of Iterations for Faulty FAID under BSC..... 103

Figure 3-143: BER for Faulty FAID under BSC..... 103

Figure 3-144: FER for Faulty FAID under BSC ..... 103

## List of Tables

Table 1-1: WP6 Gantt Diagram.....	14
Table 2-1: Case Study - C6288 Reliability Evaluation .....	24
Table 3-1: Decoder Common Interface Port Description .....	30
Table 3-2: SC Propagation Delay IG Distribution Parameters .....	35
Table 3-3: FAID Decoder.....	42
Table 3-4: MS Decoder .....	43
Table 3-5: SCMS Decoder .....	44
Table 3-6: SD Decoder .....	44
Table 3-7: Failure Probabilities for Analyzed Decoders for Different Clock Frequencies .....	45
Table 3-8: Average Values of Failure Probabilities for Considered Components in the Analyzed Decoders.....	46
Table 3-9: Different Types of Protograph for ( $dv = 3, dc = 6$ ) LDPC Codes .....	49
Table 3-10: Employed Set of LDPC Codes.....	50
Table 3-11: MS VNU Ports List.....	54
Table 3-12: MS CNU Ports List.....	54
Table 3-13: SCMS VNU Ports List.....	56
Table 3-14: P2SB Module Ports Description.....	58
Table 3-15: $VNn, m$ Module Ports Description .....	59
Table 3-16: PCHD Module Ports Description .....	61
Table 3-17: Probabilistic Gradient Descent Bit-Flipping Algorithm.....	64
Table 3-18: VNU Module Ports Description .....	64
Table 3-19: IVRG Module Ports Description.....	67
Table 3-20: Hardware and Throughput Estimation for PGDBF with Different RG Implementations and for Offset Min-Sum.....	68
Table 3-21: Alphabet Format .....	69
Table 3-22: Gallager-B with Extended Alphabet Decoding Algorithm .....	69
Table 3-23: LDPC Decoder Ports Description .....	71
Table 3-24: LDPC Decoder Design Parameters.....	71
Table 3-25: Bit Node Port Description.....	72
Table 3-26: Check Node Ports Description.....	74
Table 3-27: FPGA Resources Utilization and Maximum Clock Frequency of Implemented Hardware Decoders.....	76
Table 3-28: PPR and PDR Metrics for the Decoders under Test (Voltage Scaling Scenario) .....	95
Table 3-29: Uncertainty Intervals for $V_{pp}$ and $V_{pd}$ Estimates .....	95
Table 3-30: Energy Savings of Evaluated Decoders Operating at $V_{pp}$ (@BER = $10^{-4}$ ) .....	95
Table 3-31: PPR and PDR Metrics for the Decoders under Test (SFI Scenario) .....	104

## Abbreviations

AIG	And-Inverter Graph
AP-LLR	A-Posteriori LLR
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BF	Bit Flipping
BI-AWGN	Binary Input - Additive White Gaussian Noise (channel)
BLIF	Berkeley Logic Interchange Format
BN	Bit Node
BPSK	Binary Phase Shift Keying
BRAM	Block Random Access Memory
BS	Barrel Shifter
BSC	Binary Symmetric Channel
C2	Two's Complement
CAD	Computer Aided Design
CDF	Cumulative Distribution Function
CMOS	Complementary Metal Oxide Semiconductor
CN	Check Node
CNFET	Carbon Nanotube Field-Effect-Transistor
CNU	Check Node processing Unit
CPE	Codeword Prediction Encoder
DDR	Double Data Rate
DDR3	Double Data Rate type 3
DFF	D Flip-Flop
DSR	Degradation Stochastic Resonance
ECC	Error Correcting Codes
EDA	Electronic Design Automation
EM	Edge Memory
ET	Early Termination
EXIT	EXtrinsic Information Transfer
FAID	Finite Alphabet Iterative Decoder
FER	Frame Error Rate
FPGA	Field Programmable Gate Array
FSS	Frequency Scaling Sensitivity
GDBF	Gradient Descent Bit Flipping

IC	Integrated Circuit
I2C	Inter-Integrated Circuit
INV	Inverter
IVRG	Intrinsic-Valued Random Generator
JTAG	Joint Test Action Group
KDV	Knock Down Voltage
LDPC	Low Density Parity Check
LFSR	Linear Feedback Shift Register
LHCA	Linear Hybrid Cellular Automata
LLR	Log-Likelihood Ratio
MC	Monitoring and Controller
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MS	Min-Sum
MSB	Most Significant Bit
NAND2	2-input NAND
PDF	Probability Density Function
PEG	Progressive Edge Growth
PCC	Parity Check Circuit
PCHD	a-Posteriori Counter update and Hard Decision
PDR	Performance Degradation Region
PGDBF	Probabilistic Gradient Descent Bit Flipping
PLA	Programmable Logic Array
PMBus	Power Management Bus
PO(s)	Primary Output(s)
PPR	Performance Preservation Region
P2SB	Probabilistic-to-Stochastic Bitstream
PVT	Process, Voltage and Temperature
QC	Quasi-Cyclic
RC	Random Connector
RG	Random Generator
RTL	Register Transfer Language
SAIF	Switching Activity Interchange Format
SB	Stochastic Bitstream
SC	Standard Cell
SCMS	Self-Corrected Min-Sum
SD	Stochastic Decoder

SDF	Standard Delay Format
SFI	Simulated Fault Injection
SM	Sign Magnitude
SNR	Signal to Noise Ratio
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus
VLSI	Very Large Scale Integration
VN	Variable Node
VNU	Variable Node processing Unit
VSS	Voltage Scaling Sensitivity



The main contributions presented in this deliverable relate to **Task 6.1** and **Task 6.3** as follows.

### **Task 6.1 - Implementation of LDPC Decoders**

We implemented at Register Transfer Level (RTL) in VHDL/Verilog, debugged, and mapped on a Xilinx Virtex-7 FPGA several state-of-the-art and reliability enhanced LDPC decoders proposed in the WP3 framework for a codeword length  $N=1296$  and  $R=0.5$  as follows: Min-Sum (MS), Self-Corrected Min-Sum (SCMS), Finite Alphabet Iterative Decoder (FAID), Stochastic Decoder (SD), Gradient Descent Bit Flipping (GDBF), Probabilistic GDBF (PGDBF), and Gallager B with Extended Alphabet (GB).

We developed an experimental hardware platform that allows us to simulate the decoder exposure to environmental aggression factors, e.g., temperature, and cosmic radiation, by diminishing the power supply voltage  $V_{dd}$  under its nominal value, which may result in timing faults. In this way we can modulate the fault presence rate by means of the  $V_{dd}$  value, i.e., the lower the  $V_{dd}$  value the higher the fault rate, but we cannot control the fault occurrence location on the decoder real estate.

We developed a hybrid fault injection HDL/C++ simulation framework that allows for decoder simulation in the presence of errors (flipped bits), which location and density are derived according to decoder architectural and implementation details. Related to this we also introduced a methodology to create a fault map meant to guide the fault injection process, which is reflecting the contribution of the internal organization of each decoder basic building block to the fault error rate at its outputs.

We evaluated the implemented decoders under both scenarios, i.e., voltage scaling and fault injection, over different technology and environmental aggression profiles and quantified their figure of merit in terms of: (i) decoding performance, specifically Frame Error Rate (FER) and Bit Error Rate (BER), (ii) average number of iterations, (iii) throughput (Mb/s) normalized to BER/FER, and (vi) energy/bit ( $\mu\text{J}/\text{bit}$ ) normalized to BER/FER.

We introduced two additional metrics: Voltage Scaling Sensitivity (VSS) and Frequency Scaling Sensitivity (FSS), which are meant to capture the way a decoder reacts to voltage and frequency scaling, respectively. VSS provides inside on: (i) the decoder potential to save energy while still delivering its expected performance and (ii) how much performance one can still get in situations when the energy source is confined. FSS provides inside on: (i) the decoder potential to operate at increased clock frequency while providing its expected performance and (ii) how much overclocking one can still resort to if channel conditions permit.

We presented a thorough analysis of the voltage scaling and simulated fault injection results which confirm the fault-tolerance capabilities of the i-RISC proposed decoders and serve as guideline for the selection of a proof-of-concept fault-tolerant decoder class (which exhibits the best trade-off in terms of performance and ability to tolerate/mask faults). In particular our experiments indicate that voltage scaling may result in energy savings between 45% and 67%, while preserving decoder's nominal throughput and error correction performance. Based on simulated fault injection results, the decoder potential to increase throughput by means of overclocking is also estimated to be between 77% and 150%, while preserving the nominal error correction performance.

### Task 6.3 - Reliability-Aware Synthesis Tool

A novel CAD framework is proposed that improves the circuit reliability by re-designing the Boolean network such that reliability is enhanced with little overhead in terms of area, delay, and/or power. In this line of thought a novel set of tools have been developed targeting various levels of abstraction within the VLSI design cycle.

An integrated design flow comprising of industry and academic tools is proposed. Design compiler from Synopsys is used to synthesize a high level description of a circuit specified in Verilog. The resulting gate level netlist is then translated into an AIG data structure, which is then analyzed/optimized/manipulated within the i-RISC custom tool built as a wrapper around the open source tool 'ABC'.

One of the keys for developing an efficient optimization tool is the availability of accurate reliability information as well as efficient/fast algorithms for computing the reliability of logic functions representing partial solutions during the optimization process. A probabilistic and simulation based methodology were developed. Rewriting based local transformation techniques are employed to improve the circuit reliability at little overhead of area. Further, a novel fault tolerant logic augmentation technique is proposed. The resulting optimized netlist is then technology mapped and analyzed using standard industry tools from Synopsys for delay, area and power.

We also presented a reliability improvement case study on the MCNC benchmark circuit 'C6288', which demonstrates that both the optimization algorithm as well as the fault tolerant techniques can contribute to a significant circuit reliability improvement.

Such a flow makes use of a number of state of the art tools within Synopsys complementing them with the custom i-RISC tools for ultra-low power and reliable circuit design.

We note that given that the WP4 investigations in reliable data storage and transport are still in early stages a proof of concept in this direction is not yet at hand. Thus while we already initiated **Task 6.2** related activities the reporting of the results will be done in Deliverable D6.2.

Finally, we'd like to stress out that even though the deliverable could not be finalized in time its objectives have been reached and the late delivery doesn't have any negative effect on the other i-RISC activities. On the contrary, by solving the faced technical hurdles, which is one of the reasons behind the delayed delivery, we created better technical premises and smoothed the way towards the successful completion of the final i-RISC proof of concept. We note that apart of a number of unexpected technical challenges, which usually occur during such a complex design effort, the experiment preparation and execution were very much time consuming, thus substantially contributed to the late delivery.

The deliverable is organized as follows: Section 2 is dedicated to the EDA reliability-aware synthesis tool. Section 3 is concerned with the implementation and evaluation of the i-RISC LDPC decoders in faulty environments. Finally, in view of our findings, we draw some conclusions in Section 4.

## 2. Implementation of the i-RISC Reliability-Aware Synthesis Tool

**Abstract:** The aim of this task is to propose a novel systematic and integrated methodology to address and improve combinational circuit reliability measured in terms of Soft Error Rate (SER). In this line of reasoning, as part of the i-RISC project, a novel set of tools were developed targeting various abstraction levels within the Very Large Scale Integration (VLSI) design cycle. The proposed logic optimization Computer Aided Design (CAD) framework makes use of rewriting techniques, which at their turn employs local transformations. The main idea behind our proposal is to replace parts of the circuit with functionally equivalent but more reliable logically equivalent counterparts. Further, we have developed a number of logic augmentation techniques to improve the circuit fault tolerance. Last but not least we also introduced the following reliability estimation approaches: (i) a simulation based technique, which estimates the circuit reliability via gate-level fault injection and (ii) a fundamentally novel analytical technique which is building upon the conditional probability theory.

### 2.1. Introduction

Traditional logic synthesis methodologies and EDA tools are centred on fulfilling timing, power, and area constraints or on achieving acceptable trade-offs among those [Pedram96] [Mehrotra11]. However, as the Complementary Metal-Oxide-Semiconductor (CMOS) technology entered the nanometer era, such an approach cannot cover any longer all the relevant design aspects. Technology scaling has precipitated higher operating speeds, lower operating voltages, and lower operating noise margins; all of which contribute to reduced switching energies, allowing legitimate logic signals to be readily overwhelmed by single-event-induced charge-collection transients [Dodd03]. Nanotechnology specific issues, e.g., power supply voltage ( $V_{dd}$ ) reduction, higher impact of process parameter and temperature variations, result in increased device failure rates, making CMOS Integrated Circuits (ICs) less reliable [Borkar05] [Constantinescu03].

As power usage is proportional to the square of voltage, operating at very low voltages offers potential for large power savings [Kaeslin08] [Vittoz14]. For example, if a circuit is operated at  $0.1 V$  instead of a process nominal  $1 V$ , a  $100 \times$  power saving is potentially achievable. However this means that the supply voltage is significantly below the transistor threshold voltage and it is well known that in this weak inversion regime, MOSFET transistors exhibit high voltage gain but very low currents. There are many possible ways in which a sub threshold circuit may become unreliable. The simplest is of course due to noise, made worse by leakage induced noise. Stuck at “0” or “1”, or similar persistent faults can occur due to process variations [Borkar05], either statically or dynamically (dependant on temperature and voltage), which are inherent due to the fact that silicon doping is a stochastic process, and in small process geometries a very small number of dopant atoms can be present in a MOSFET transistor channel (10s to 100s). This means that the stochastic process doesn’t necessarily average out, leading to nearby MOSFETs having very different electrical properties and by implication switching behaviour.

Another unreliability mechanism is the unpredictable timing. In [Chen14], it was indicated that sub-powered gate arrival times follow inverse Gaussian distributions, with a long calculation completion time tail. In a practical system, a gate chain has a certain allowed slack, and if individual gate completion times chance this may be exceeded and errors may occur. Even a small error probability at the level of individual gates might result in a large error probability at the circuit final outputs [Choudhury10]. We note that this tendency is not CMOS specific, as even the most promising post

silicon devices, e.g., Carbon Nanotube Field-Effect Transistors (CNFETs) that are considered to eventually replace CMOS, suffer from various amounts of statistical variation in device behaviour, potentially leading to a lack of reliability. As a result, reliability is turning out into a major design metric sharing equal importance with the other existing design metrics. Consequently, design time reliability assessment and optimization is becoming a mandatory IC design flow step which targets the reliability improvement for circuits/systems built with unreliable components.

In this section, all the techniques and methodologies that were developed over the past two years and assembled together into a single package are presented. As depicted in Figure 2-1, the reliability aware synthesis tool comprises three major components. We have developed a multiple set of reliability computation engines that would estimate the probable circuit output error with varying degrees of accuracy and speed. A multiple set of logic optimization techniques has been developed which pre-dominantly operate at gate level. Furthermore, to achieve extra added value, we have embarked on developing a number of logic augmentation techniques to improve the circuit fault tolerance.

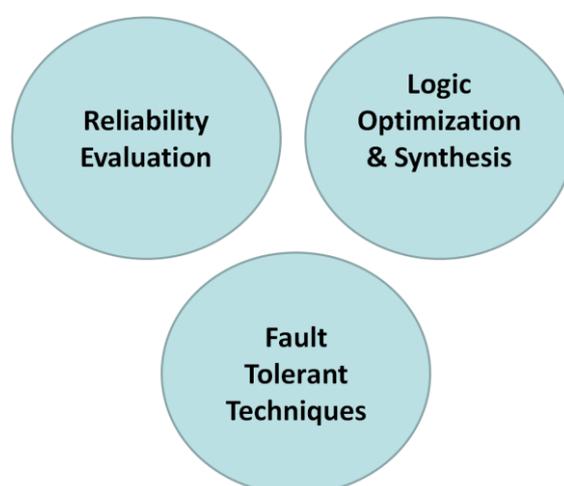


Figure 2-1: Reliability Aware Synthesis Tool Sub-Branches

## 2.2. The Tool Chain – Complete CAD Framework

The design flow consists of several academic tools developed in-house within the i-RISC project scope that are integrated alongside several industrial tools. This integration would be extremely beneficial in the implementation of the final proof of concept. The complete design flow is presented in Figure 2-2 and it outlines the digital circuit design path from the Register Transfer Language (RTL) level to the final error resilient technology mapped gate-level netlists followed by reliability, power, delay, and area reports. Some of the important flow steps are as follows:

**Step 1:** Convert the circuit description into its corresponding And-Inverter Graph (AIG).

**Step 2:** Run Reliability driven logic optimization tools to synthesize gate level netlists.

**Step 3:** Perform Reliability Analysis to compute the achieved improvement in terms of error resilience. Reliability details of every node in the network are stored into the output file.

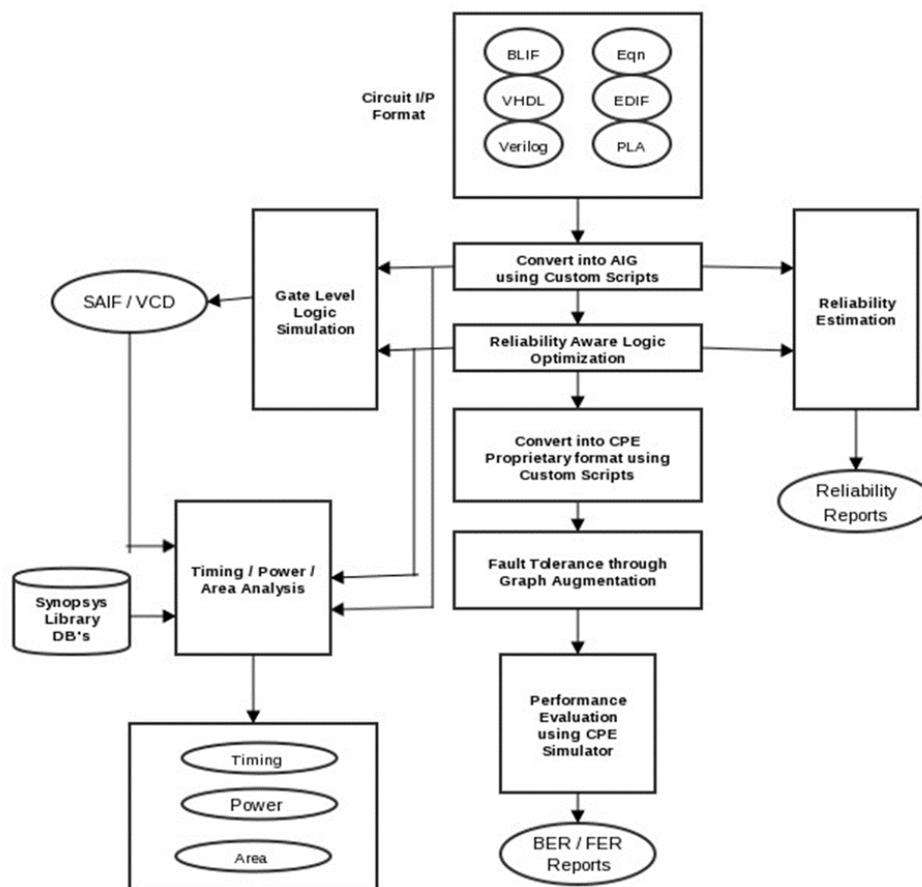


Figure 2-2: Reliability Aware Synthesis Tool - The Complete Framework

**Step 4:** Perform Gate level simulations to collect switching activity details. These are saved in the standard Switching Activity Interchange Format (SAIF) format.

**Step 5:** Using netlists from **Step 1** and **2** and switching reports from **Step 4**, invoke Synopsys Design Compiler to perform the power, area, and timing analysis. Subsequently, perform comparative studies to evaluate the savings/overhead corresponding to the new netlists.

**Step 6:** Implement an LDPC encoding scheme on top of the reliability optimized netlist. The structure of the parity circuitry to augment the optimized circuit is determined based on its functionality.

**Step 7:** Convert the netlists {combinational circuit, parity circuitry, and the LDPC decoder} into the internal proprietary format understood by the Codeword Prediction Encoder (CPE) simulator.

**Step 8:** Invoke the CPE simulator to perform encoding and decoding simulations and generate reports comprising FER/BER analysis, critical node count, etc.

### 2.2.1. Circuit Representation and Modification

Over the years, a number of academic EDA tools [Yanushkevich05] [Sentovich92] [Wu05] have been proposed in the literature. These open source tools provide a programming environment and a solid platform for research in logic synthesis, power estimation and power optimisation as well as for implementing new developments into them. ABC [Brayton10] is a logic synthesis and verification tool which performs scalable logic optimisation based on AIGs [Mishchenko06]. In all of these

academic tools, data structures and algorithms largely determine the tool efficiency in providing support for implementing new capabilities. As reported in Deliverable D5.1 [i-RISC/D5.1], we have decided to use AIG as the data structure and 'abc' as the platform to develop and implement all the reliability related algorithms.

The tool accepts any high level description to generate the netlists of a generic function. It can take any function description in, e.g., BLIF, VHDL, Verilog, PLA, convert it into the ".eqn" intermediate format and then generate its corresponding AIG representation as described in Figure 2-3. We note that during the process of modifying the circuit representation from one format to another it is imperative to maintain the logical equivalence of the original and new circuits, which is guaranteed by adopting different kinds of formal verification techniques.

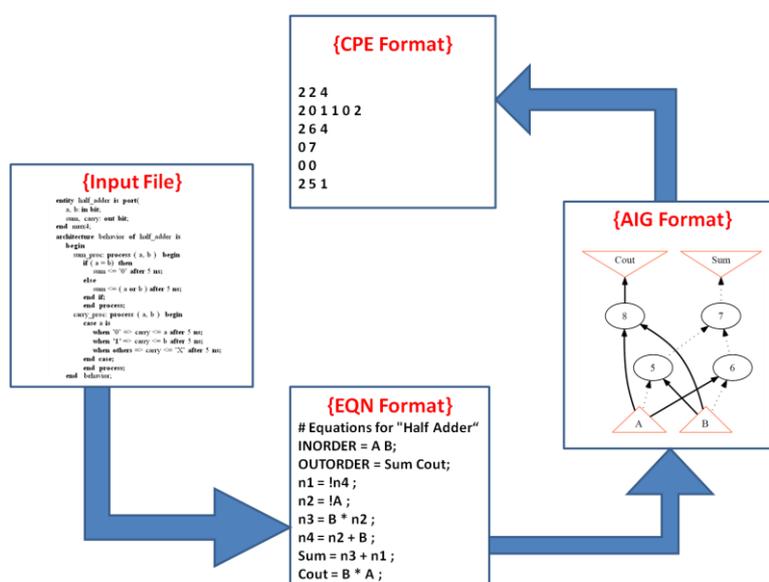


Figure 2-3: Different Formats of Circuit Representation

## 2.2.2. Gate Level Logic Simulation

As depicted in Figure 2-4, RTL simulations are performed using VCS [SYNTOOL], a commercial tool from the Synopsys EDA vendor. The logic description is optimized and then synthesized to a gate-level netlists by employing the internal reliability optimization tool. Formal verification is performed in Synopsys Formality [SYNTOOL], ensuring the equivalence between the RTL description and the resulting gate-level netlists. Reports generated by the synthesis tool detail the silicon area consumed by the logic design. Timing reports are generated using Synopsys Primetime [SYNTOOL] determining the longest paths and the maximum clock frequency. Primetime also generates Standard Delay Format (SDF) [SDF04] data containing delay information for annotation onto the netlists during gate-level simulation. The primary goal of constructing this flow was to perform power analysis on the pre-placement netlist using commercial tools. Switching probability information is recorded by VCS during simulation as SAIF [SYNSAIF]. SAIF is used during power analysis to obtain realistic power figures for the simulated scenario.

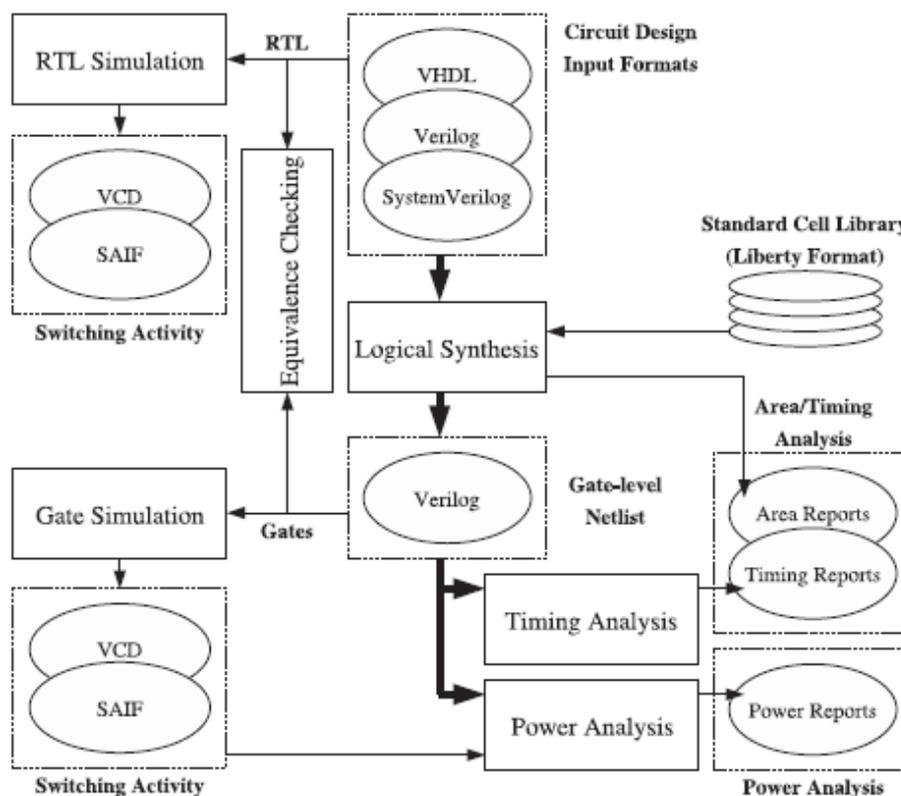


Figure 2-4: Timing and Power Analysis Flow

### 2.2.3. Reliability Estimation and Analysis

Reliability analysis of logic circuits deals with the computation of the impact that gate errors might have on circuit Primary Outputs (PO's). Generally speaking a pure reliability analysis based on HSPICE Monte Carlo simulations is not feasible real life circuits due to its prohibitive computation time and excessive resource requirements. Several analytical approaches were previously proposed [Choudhury09]. As we represent circuits in the AIG format, a novel algorithm based on probability principles is developed, with the prime focus being AND and INVERTER gates. Two different methodologies, simulation based approach and the probabilistic error gate model based approach have been devised. Figure 2-5 depicts the complete flow of the reliability analysis tool.

The circuit under test is passed onto both the probabilistic and simulation based reliability computation algorithms. The probabilistic based methodology emulates all the gates with the probabilistic error models and based on input switching activity, static probability, and gate error values, it computes the expected reliability of the output node. The simulation-based algorithm appends all the gates with extra XOR gates to randomly toggle the output value there, by inserting an error. We use a Mersenne twister to generate highly random test patterns, which we utilize to compute the final output reliability values. The final output reports from both these models are compared to define the accuracy of the probabilistic gate error model. Though very accurate, the simulation methodology is very expensive in terms of execution time, which might preclude its utilization on large circuits. Hence, it is finally a tradeoff between accuracy and speed when it comes to choosing one of these two algorithms.

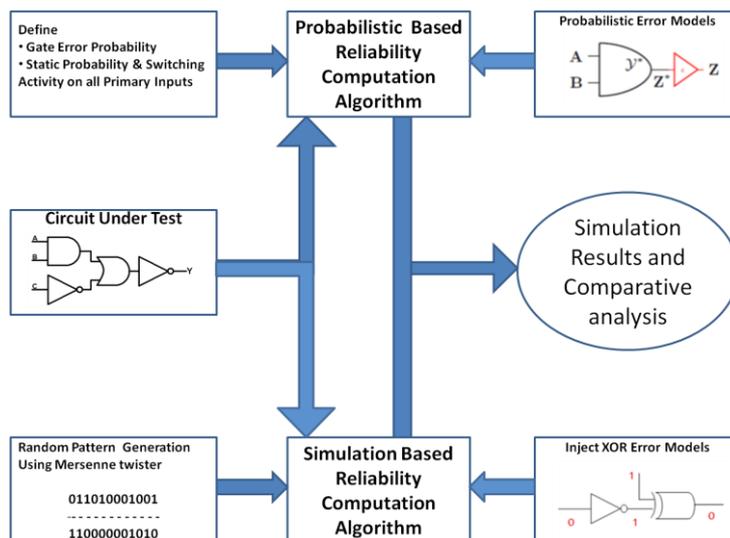


Figure 2-5: Reliability Computation and Analysis Flow

#### 2.2.4. Multi Objective Optimization

Logic optimization and synthesis is the process of taking in a higher level circuit representation and translating it into hardware. We have proposed two different methodologies that were thoroughly articulated in Deliverable D5.2 [i-RISC/D5.2], both based on the rewriting technique. Figure 2-6 describes the multiobjective optimization tool. From the under optimization circuit, which can be for example a reference MCNC benchmark circuit, the output error probabilities of two netlists are initially computed: (i) default circuit without any optimization and (ii) circuit optimized by the best 'abc' synthesis algorithm. It is not always necessary that the second configuration has higher reliability. The synthesis algorithms in 'abc' (or in general) are mainly targeting delay reduction which can affect adversely the reliability. After selecting the initial circuit configuration, we apply the developed optimisation methodology transformation rules, we perform Boolean matching to pick the matching rule. The one which provides the highest reliability improvement is selected based on the results of the reliability evaluation function. Further, this process continues until no more rules can be applied on this node, that can improve the circuit reliability. We perform similar set of operations on all the nodes in the circuit. Using this method of optimisation, the number of inputs and outputs of a particular logic function is not modified. The logic network describing the function is updated during each iteration of the optimisation algorithm. The delay/area optimised initial logic network is transformed iteratively for improved reliability (delay/area driven reliability optimisation).

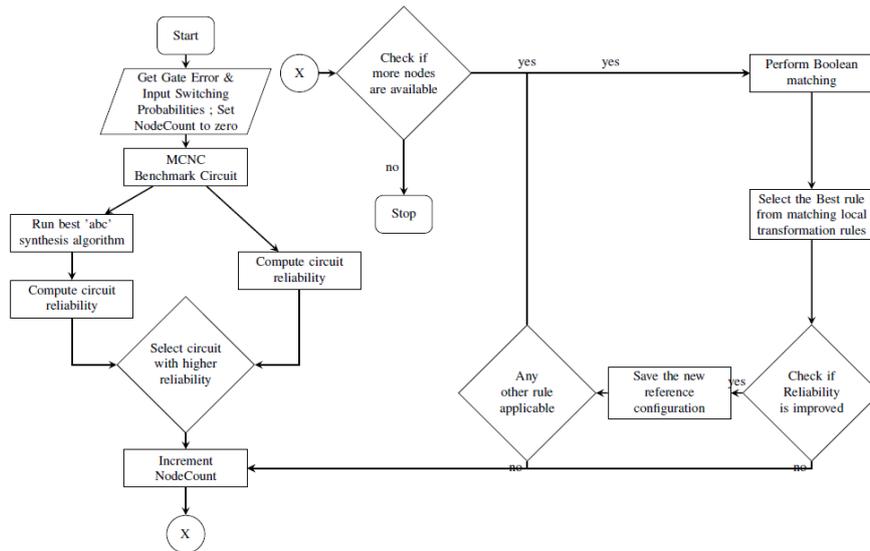


Figure 2-6: Reliability Aware Logic Optimization Tool Flow

### 2.2.5. Fault Tolerant Graph Augmentation

The aim of fault tolerant techniques is to systematically encode the input function under consideration. The focus is not on altering logic but on augmenting it to add redundancy. In this instance, additional logic network is added to the original network describing the function which results in the an increase number of outputs for the resulting network.

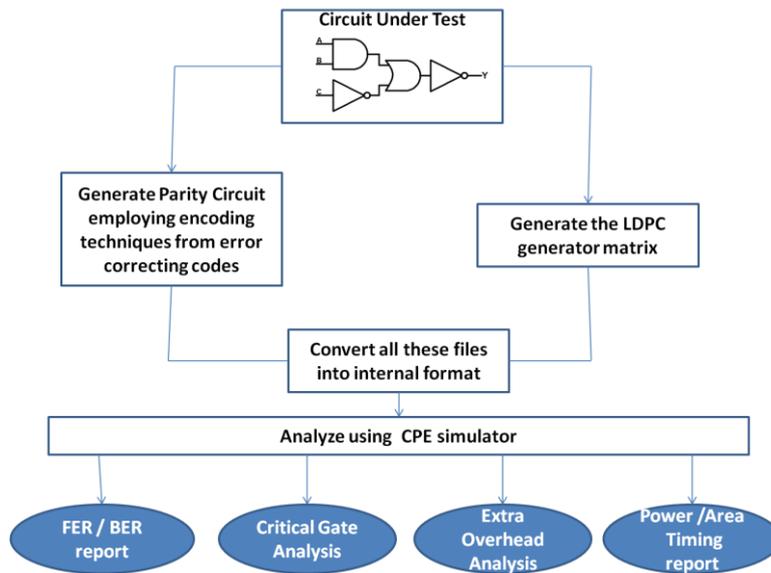


Figure 2-7: CPE based Graph Augmentation

By using Error Correcting Codes (ECC) based architectures redundant logic is added to enable retrieving the correct output thereby improving the combinatorial circuit reliability. This approach takes the input function netlist and translates it into an AND Invert set of equations for further manipulation and analysis of the number of gates and longest path modifications. The logic network annotation is informed by a particular ECC scheme. The two classes of logic functions identified in our study are linear or non-linear functions. The resulting annotated logic network is then decoded

using an additional logic network associated to the chosen ECC. Based on where we perform decoding, we can have the following scenarios:

- Symmetric: encoding and decoding processes are on chip, and hence affected by same faulty conditions;
  - iRISC developed LDPC decoder architectures are valid candidates for the ECC schemes.
- Asymmetric: the decoding process is performed offline and hence it is assumed that it is not affected by faults.
  - Allows the use of generic ECC.

The complete methodology of Codeword Protection Encoding (CPE) is detailed in Deliverable D5.2 [i-RISC/D5.2] and the CPE based graph augmentation flow is depicted in Figure 2-7.

### 2.3. A Case Study

In this section, the potential practical implications of the proposed reliability aware synthesis tool are evaluated. The total improvement achieved in terms of circuit reliability is evaluated by performing a set of simulations. The proposed reliability aware synthesis algorithm is applied on the MCNC benchmark circuit 'C6288', which implements a 32-input 32-output logic function. Simulation results comparing the average circuit output errors corresponding to the original, the optimized configuration, and the CPE based approach obtained from our tool for different technologies, i.e., basic gate error probability, are reported in Table 2-1. The individual gate error is given in column 1, while column 2 and 3 presents the output error probability of the original and the optimized circuit, respectively. Column 4 list the reliability improvement in % achieved by means of the optimization procedure. Columns 5(6), 7(8), and 9(10) summarize the output error probability and the error probability improvement achieved by employing the CPE approach, for different gate Critical Threshold (CT) values.

Table 2-1: Case Study - C6288 Reliability Evaluation

Gate_Err	Original	Optimized	%Imp	CPE					
				CT==00	%Imp	CT==05	%Imp	CT==10	%Imp
0.10	0.48	0.47	0.63	0.48	-1.39	0.44	7.96	0.46	2.29
0.05	0.46	0.46	0.22	0.48	-4.24	0.39	14.63	0.41	11.45
0.02	0.45	0.44	2.01	0.46	-1.92	0.31	30.65	0.34	23.51
0.01	0.42	0.42	0.94	0.45	-7.17	0.28	33.00	0.31	26.75
0.01	0.36	0.37	4.20	0.42	-18.26	0.17	53.31	0.25	28.68
0.00	0.25	0.26	1.99	0.34	-37.33	0.06	76.86	0.14	46.06
0.00	0.17	0.16	5.42	0.29	-72.05	0.02	89.56	0.04	76.13
0.00	0.09	0.10	14.88	0.19	-120.35	0.00	94.71	0.02	75.40
0.00	0.04	0.04	4.04	0.07	-57.20	0.00	97.31	0.01	87.83

0.00	0.02	0.02	7.91	0.03	-55.25	0.00	97.01	0.00	84.53
0.00	0.01	0.01	7.72	0.02	-94.50	0.00	97.72	0.00	89.43
0.00	0.00	0.00	15.70	0.01	-71.85	0.00	98.08	0.00	88.57
0.00	0.00	0.00	5.56	0.00	-43.15	0.00	98.03	0.00	90.70

---

From the table, it is clear that significant SER reduction can be achieved (at the expense of negligible area overhead) by employing the optimisation algorithms (columns 2, 3, 4). A very good reliability performance is observed in case of CPE implementation for large CT values. But the current limitations with the CPE are the large extra gate count due to the parity circuit and the number of gates that have to be safeguarded (represented by CT in %).

## 2.4. Conclusion

In this section, we summarized the activities related to the development of the reliability aware synthesis tool. We proposed an integrated design flow, which combines all the up to date developed i-RISC custom tools together with widely used tools in the circuit design industry. We also presented a case study, which demonstrates that both the optimization algorithm as well as the fault tolerant techniques can contribute to a significant circuit reliability improvement. While the tools are integrated in a complete flow with proven functionality, there a number of issues, which still need to be addressed. Going forward, this flow will be used:

- To improve upon the initial work of the reliability computation techniques and the graph optimization algorithms.
- To systematically optimize the circuits to improve reliability within a multi-objective optimization framework.
- To validate and to characterize the circuits proposed as part of proof of concept within i-RISC.

### 3. Evaluation of LDPC Decoders in Fault Inducing Environments

**Abstract:** A desideratum of computing circuits built out of current technology nodes unreliable devices is fault tolerance. A circuit, which is robust to environmental aggression (e.g., supply voltage variation, temperature, and cosmic radiation), enables an improved dependability profile and potentially extended useful lifetime expectancy. Moreover it provides a certain noise immunity, which one may trade for energy consumption by means of power supply scaling. In this section we perform a fault tolerance assessment of state-of-the-art and i-RISC proposed LDPC decoders introduced in WP3. In particular we evaluate Min-Sum (MS), Self-Corrected Min-Sum (SCMS), Finite Alphabet Iterative Decoder (FAID), Stochastic, Gradient Descent Bit Flipping (GDBF), Probabilistic GDBF (PGDBF), and Gallager B decoders for a codeword length  $N=1296$  and  $R=0.5$  as described in Section 3.4.2. All decoders are implemented in VHDL/Verilog and are exposed to external aggression via two approaches, which emulate in different ways in-field real-life scenarios, namely voltage scaling and judicious fault injection. The decoder performance expressed in terms of Bit-Error-Rate (BER), Frame-Error-Rate (FER), energy consumption/bit, throughput, average number of iterations, maximum operation frequency, area, is evaluated for each LDPC decoder architecture, over different scenarios (i.e., different CMOS process and voltage corners, communication channel types, i.e., Binary Symmetric Channel (BSC) and Additive White Gaussian Noise (AWGN) channel). Based on performance and reliability profile of the comprising basic building blocks, the considered decoders architectures are compared, towards substantiating a fault-tolerant proof-of-concept LDPC decoder architecture.

#### 3.1. Introduction

Nowadays, the shrinking of transistor sizes has reached a level where it is extremely difficult, if not impossible, to provide reliable transistors that can properly work all the time without experiencing faults. In this condition, building reliable chips out of unreliable transistors has been a major topic in the cutting edge VLSI research where the reliability is the main issue. Therefore, the evaluation of reliable systems in the presence of faulty components is critically important. In this section, we evaluate state of the art LDPC decoders built out of unreliable components. Different methods to evaluate faulty LDPC decoders exist, and C-simulation based evaluations were reported in **[i-RISC/D3.1]** and **[i-RISC/D3.2]** for simple theoretical error models. In this section we target real hardware based evaluations. Given that LDPC decoder ASIC design and fabrication is out of the i-RISC project scope we rely on FPGA based fast prototyping. We implement at Register Transfer Level (RTL) in VHDL/Verilog, debug and map on a Xilinx Virtex-7 FPGA seven types of LDPC decoders, i.e., Min-Sum (MS), Self-Corrected Min-Sum (SCMS), Finite Alphabet Iterative Decoder (FAID), Stochastic, Gradient Descent Bit Flipping (GDBF), Probabilistic GDBF (PGDBF), and Gallager B, for a codeword length  $N=1296$  and  $R=0.5$ . To have realistic results, we develop an experimental hardware platform (described in Section 3.2), which allows for decoder evaluations under different channel conditions and timing faults induced by diminishing the power supply voltage  $V_{dd}$  under its nominal value. In this way we can modulate the fault presence rate by means of the  $V_{dd}$  value, i.e., the lower the  $V_{dd}$  value the higher the fault rate. The voltage scaling method is quite effective in inducing timing faults all over the circuit but does not provide us the means to control their occurrence location on the decoder real estate. Given that in real life situations fault density and location are related to architectural and implementation details we also performed a specific fault injection as described in

Section 3.3. In this endeavor we create a fault map reflecting the contribution of the internal organization of each basic building blocks to the fault error rate of its outputs. As augmenting the FPGA decoder implementation with fault injection specific extra circuitry results in a large area overhead and the FPGA fault injection process is very slow we decide to make use of simulated fault injection by means of a mixed simulation environment. In this way we can still fully control the fault location, according to the fault map reflecting the reliability of each internal decoder component, and collect performance data faster than by means of a hardware only fault injection approach.

The implemented decoders are evaluated in both scenarios in terms of: (i) decoding performance, specifically Frame Error Rate (FER) and Bit Error Rate (BER), (ii) average number of iterations, (iii) throughput (Mb/s) normalized to BER/FER, (vi) energy/bit (pJ/bit) normalized to BER/FER.

For the voltage scaling we introduce an additional metric the Voltage Scaling Sensitivity (VSS), which is meant to capture the way a decoder reacts to the voltage scaling process and provides inside on: (i) the decoder potential to save energy while providing its expected performance and (ii) how much performance one can still get in situations when the energy source is confined. Our experiments suggest that voltage scaling may result in energy savings between 45% and 67%, while preserving the nominal throughput and error correction performance.

Similarly, for the second scenario, we introduce the Frequency Scaling Sensitivity (FSS) metric, which provides inside on: (i) the decoder potential to increase throughput by means of overclocking, while providing its expected performance and (ii) how much overclocking one can still resort to if channel conditions permit. In this case our experiments indicate that decoder overclocking may result in throughput increase between 77% and 150%, while preserving the nominal error correction performance.

### 3.2. Voltage Scaling Evaluation Framework

To facilitate the real hardware based LDPC decoder evaluation, we develop an experimental hardware platform [Marconi15] that consists of a laptop and a Xilinx VC707 board as depicted in Figure 3-1. The laptop is dedicated to the following activities:

- Designing the hardware platform targeting Xilinx Virtex-7 FPGA: XC7VX485TFFG1761-2 inside the Xilinx board and generating the bitstream files.
- Downloading the bitstream files for FPGA hardware and the software files for the MicroBlaze through the USB JTAG interface.
- Monitoring/capturing the number of iterations and decoding outcomes, FER, and BER through the USB UART.
- Measuring Energy/bit using the Fusion Digital Power Designer from Texas Instrument through Texas Instrument USB Interface adapter by reading PMBus, and accessing Power Supply Monitor and Controller inside the board.

To build the evaluation hardware support, i.e., the AWGN/BSC channel emulator, the Binary Phase Shift Keying (BPSK) modulator, any other functionality for evaluation purpose, and the decoders, the Xilinx board is utilized. The MicroBlaze processor, resident on the Virtex-7 FPGA fabric, in collaboration with the LDPC Monitoring and Controller (MC) module mainly acts as the evaluation hardware support by accessing software and data stored in FPGA memory (BRAM) and external DDR3 memory. MC monitors the decoding outcome and conveys quantized probabilities or Log-Likelihood Ratios (LLRs) from the MicroBlaze to the evaluated decoder. The two possible outcomes of

the decoding process are: (i) “success” and (ii) “give up”. If all decoder check nodes are satisfied, the outcome is “success”. In the case that not all check nodes can be satisfied after the maximum number of iterations, the system reports “give up”. The MicroBlaze processor utilizes these outcomes for computing the statistical results of the experiments, i.e., BER, FER, and average number of decoding iterations. The laptop displays the experimental results received from FPGA board through USB UART interface.

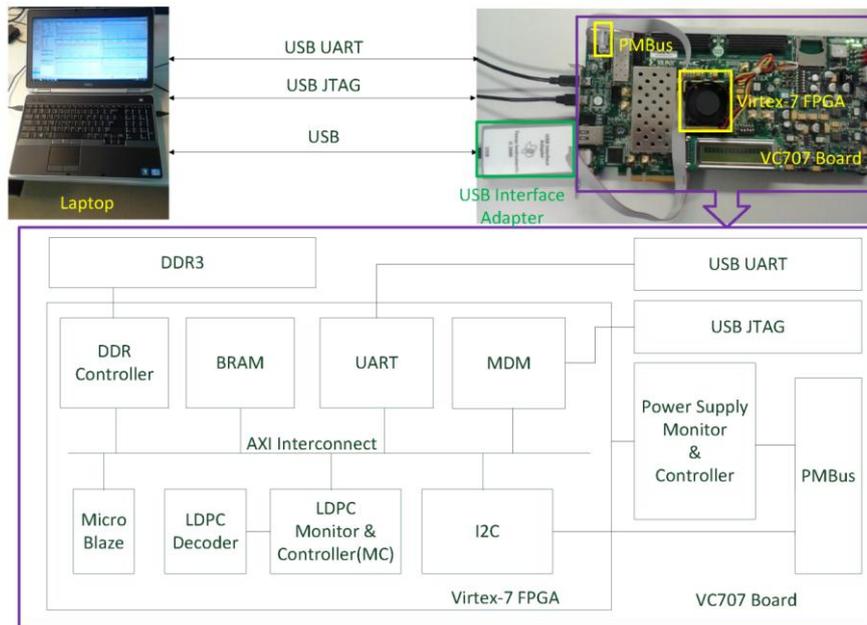


Figure 3-1: Experimental Hardware Platform

The current platform provides these features: (i) it is configurable for various LDPC decoders (e.g., MS, SCMS, FAID, Stochastic), (ii) allows for easy integration of any evaluated decoders due to a common interface approach, (iii) provides voltage scaling support and enable power/energy measurement due to its ability to access directly the PMBus of Power Supply Monitor and Controller through the I2C interface, and (iv) can evaluate decoders under both AWGN and BSC channels.

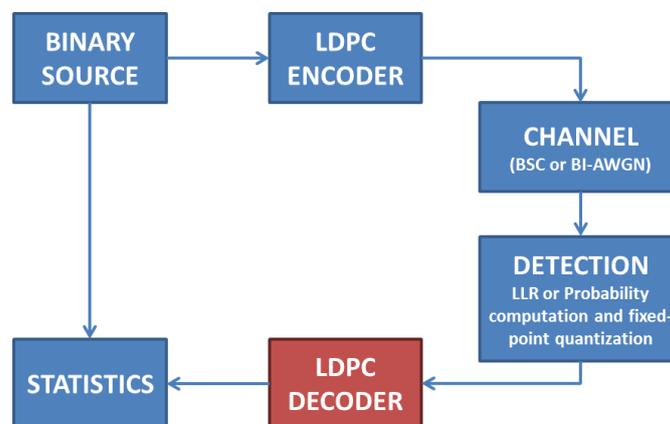


Figure 3-2: Top Level Representation of the LDPC Testbed

Figure 3-2 presents organization of the testbed we utilize to evaluate the performance of the FPGA implemented LDPC decoders, which contains the following blocks:

- Binary Source: Generates the source bit-stream (information word).
- LDPC encoder: Generates the encoded bit-stream (codeword).
- Channel: Emulates the transmission channel (either BSC or Bi-AWGN channel).
- Detection: Computes the quantized LLR or probability values. We note that the number of quantization bits depends on the LDPC decoder, as follows:
  - 1-bit LLR quantization for Gallager-B, GDBF, and PGDBF,
  - 3-bit LLR quantization for FAID,
  - 4-bit LLR quantization for MS and SCMS,
  - 6-bit probability quantization for Stochastic.
- LDPC decoder: Implements the LDPC decoding; supported decoders are Stochastic, Gallager-B, GDBF, PGDBF, FAID, MS, and SCMS.
- Statistics: Computes BER/FER and the average number of decoding iterations.

All testbed blocks except the LDPC decoder are implemented in C and are executed on the MicroBlaze. The LDPC decoders are implemented on the Virtex-7 FPGA. The decoders under test are evaluated in terms of: (i) decoding performance, specifically FER and BER, (ii) average number of iterations, (iii) throughput (Mb/s) normalized to BER/FER, and (iv) energy/bit (pJ/bit) normalized to BER/FER.

Moreover, specially tailored for voltage scaling based evaluation of LDPC decoders we introduce a new metric called Voltage Scaling Sensitivity (VSS), which is meant to capture the way a decoder reacts to the voltage scaling process. To this end we propose to capture two aspects:

- Performance Preservation Region (PPR), i.e., the voltage interval starting down from the nominal  $V_{dd}$  value in which the decoder preserves its performance, i.e.,  $PPR = V_{dd} - V_{pp}$ , where  $V_{pp}$  is the power supply value at each performance degradation starts occurring.
- Performance Degradation Region (PDR), i.e., the voltage interval starting down from  $V_{pp}$  value in which the decoder performance degrades but it still provides some useful results, i.e.,  $PDR = V_{pp} - V_{pd}$ , where  $V_{pd}$  is the power supply value at which the decoder is not functional any longer (FER gets almost 1).

PPR tells us about the decoder potential to save energy while providing its expected performance. This is the region in which one can save energy by means of voltage scaling if channel conditions permit. PDR tells us about how much performance one can still get in situations when the energy source is confined. This operation region is meant for situations when the system is in energy shortage but given that some minimum service is required it should not be shut down if still possible.

We note that both PPR and PDR values depend on the channel conditions thus we can compare only the decoders operating on the same channel type per each SNR/crossover probability value. Additionally, when determining the  $V_{pp}$  and  $V_{pd}$  values based on the voltage scaling based evaluation results we should recall that FER/BER figures are obtained by means of Monte Carlo simulations and we have to treat them as such.

Thus the  $V_{pp}$  and  $V_{pd}$  calculations have to be done in a way that takes into consideration this aspect and in a coherent way for all decoders. Given that small FER/BER differences may not hold true or even worse change the sign between different simulation runs we should go for a certain uncertainty

interval, let us say the performance is still nominal within an  $x\%$  margin and the decoder stops functioning if FER in an  $y\%$  vicinity of 1. Details on the  $V_{pp}$  and  $V_{pd}$  calculations are provided in Section 3.6.1.

We should note that while VSS can provide great insight into the decoder behavior one couldn't draw any generally valuable conclusion by comparing decoders in terms of PPR and PDR only. They put things into the right perspective for the evaluated FPGA decoder implementations under scrutiny but the ranking may not hold true for ASIC or on different decoder designs.

To facilitate the easy integration of different decoder types into the evaluation framework we define a common decoder entity described in Table 3-1.

Table 3-1: Decoder Common Interface Port Description

Port Name	In/Out	Size [bits]	Function
clk	I	1	clock signal
rst	I	1	reset active high
start	I	1	'1' to start decoding
load_data_in	I	1	'1' to load <i>load_data_in(n_value)</i>
n_value	I	11	a current bit position
data_in	I	6	soft messages
max_iter	I	11	the maximum number of iterations
done	O	1	'1' -> decoding is done
give_up	O	1	'1' -> the decoder gives up
data_out	O	1296	the decoded codeword
iteration	O	11	the number of iterations

Similar to [Marconi14], the voltage scaling and power/energy measurement are performed by accessing directly the PMBus of Power Supply Monitor and Controller through the I2C interface. The MicroBlaze communicates with the evaluated decoders via the proposed common interface as follows:

1. The MicroBlaze sets the maximum number of iterations to the *max\_iter* input.
2. The MicroBlaze resets the decoder by applying a pulse '1' to the *rst* input.
3. The MicroBlaze puts serially the quantized soft messages (i.e., LLRs or probabilities) in *data\_in* input. Every time the evaluated decoder receives a pulse '1' at the *load\_data\_in* input initiated by the MicroBlaze, the decoder needs to fetch the data to the specific bit of the messages to its internal memory addressed by the *n\_value* input coming from the MicroBlaze.
4. The decoder starts decoding when it receives a pulse '1' at *start* input initiated by the MicroBlaze.
5. After the decoding is done, the decoder sends (i) the done signal by putting '1' at the *done* input, (ii) the decoding outcome to the *give\_up* output (i.e., '1' if the decoder gives up, otherwise '0'), (iii) the number of iterations to *iteration* output, and (iv) the decoded codeword to *data\_out* output.
6. The information from step 5 is utilized by the MicroBlaze to compute the statistical results of the experiment (i.e., BER, FER, average number of decoding iterations, energy/bit, and throughput).

### 3.3. Fault Injection Evaluation Framework

As explained in Section 3.1 Simulated Fault Injection (SFI) presents the advantage of better observability and fault insertion selective control when compared with voltage scaling, which creates time errors all over the circuit. SFI based methods allow to alter specific locations in the decoder, while other blocks may still exhibit a correct behavior. Thus, SFI can be also utilized to determine which decoder components have the most significant impact on their reliability.

Regarding the SFI methodology, we employ a multi-level evaluation procedure for Register Transfer Level (RTL) descriptions, similar to the one described in Deliverable D2.2 [i-RISC/D2.2]. In the first phase, reliability measures are derived for each and every decoder building block. These measures are computed by using standard cell statistical timing characterization and Probability Density Function (PDF) propagation. The second phase consists in applying saboteur based SFI on the LDPC decoder RTL description. By employing this type of hierarchical analysis, we aim to combine the accuracy of circuit-level analysis with the low simulation overhead characteristic to RTL based evaluations.

Regarding the fault locations, we decided to only alter data path element outputs, and to let control units, as well as input-output interfaces error-free. This relates to the fact that injecting faults into the control unit can create severe disruptions in the LDPC decoder's data flow, such as reading/writing messages from/to incorrect memory addresses or routing messages to the inappropriate processing units, which might make the decoder unable to perform the LDPC decoding algorithms. We note that this is a realistic assumption as in real-life designs it is rather common to take design measures in order to make controllers more robust than data-paths.

The decoders under test are evaluated in terms of decoding performance, specifically FER and BER, and average number of iterations. Moreover, similar to the voltage-scaling scenario, we introduce a new metric called Frequency Scaling Sensitivity (FSS), which is meant to capture the way a decoder reacts to the frequency scaling process. To this end we capture the following aspects:

- Performance Preservation Region (PPR), i.e., the timing interval starting down from the nominal  $T_{clk}$  value in which the decoder preserves its performance, i.e.,  $PPR = T_{clk} - T_{pp}$ , where  $T_{pp}$  is the clock period value at each performance degradation starts occurring.
- Performance Degradation Region (PDR), i.e., the timing interval starting down from  $T_{pp}$  value in which the decoder performance degrades but it still provides some useful results, i.e.,  $PDR = T_{pp} - T_{pd}$ , where  $T_{pd}$  is the clock period value at which the decoder is not functional any longer (FER gets almost 1).

PPR tells us about the decoder potential to increase clock frequency while providing its expected performance. This is the region in which one can increase throughput by means of overclocking, without any degradation of the decoding performance. PDR tells us about how much overclocking one can still resort to if channel conditions permit.  $T_{pp}$  and  $T_{pd}$  calculations are further detailed in Section 3.6.1.8. While FSS can provide great insight into the decoder behavior, we note however that  $T_{pp}$  and  $T_{pd}$  values are determined based on simulated fault injection and not by actual overclocking of the design, and therefore we have to treat them as such.

### 3.3.1. Fault Simulation Framework

Our main goal is the evaluation of the faulty decoders' error correction capability (measured in Bit Error Rate (BER) and Frame Error Rate (FER)) under different channel parameters (Signal-to-Noise Ratio (SNR) for Binary Additive White Gaussian Noise (BI-AWGN) channel model and crossover probability for Binary Symmetric Channel (BSC) channel model), rather than the failure rate with respect to the correct LDPC decoder implementation. In order to perform this type of analysis we developed a dedicated System Verilog framework. We perform the hardware simulations using Modelsim 10.02.c commercial HDL simulator, while the DPI-C interface is utilized in order to perform RTL Verilog/VHDL description – transmission chain C++ model co-simulations.

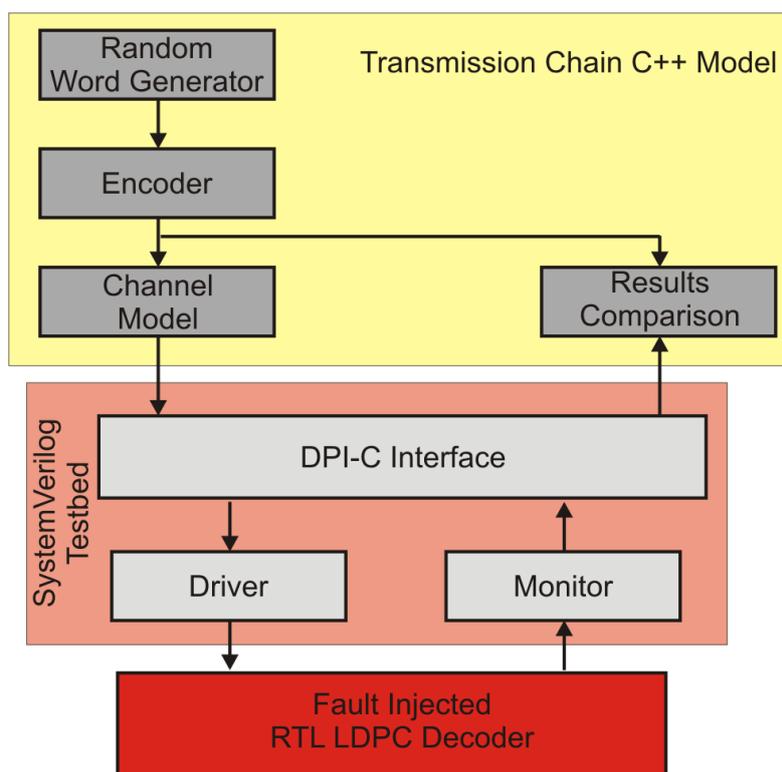


Figure 3-3: Simulated Fault Injection Framework

The SFI framework is described in Figure 3-3 and consists of:

1. Transmission chain C++ model – it is used to generate the appropriate input data frames for different channel noise models and parameters, compare the decoder output with the correct codewords, and compute the BER and FER figures.
2. System Verilog wrapper and interface – it has the role to extract the inputs frameworks generated by the C++ decoder simulator and feed them to the RTL LDPC decoder description, as well as to capture the output of the RTL description of the LDPC decoder and transmit it to the C++ simulator.
3. Fault Injected RTL description of LDPC decoders.

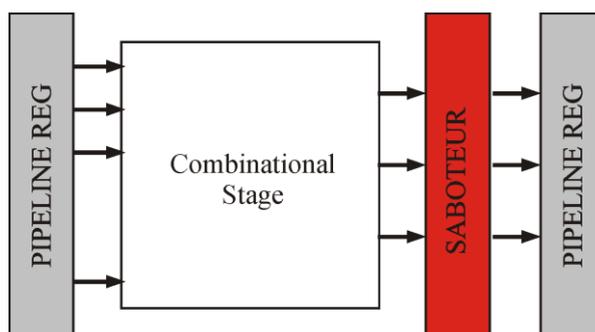


Figure 3-4: Saboteur Insertion in Combinational Logic

The chosen fault injection methodology is based on probabilistic saboteurs as indicated in Figure 3-4. Given that we are mainly targeting timing faults, which manifest only at output transitions, the employed saboteurs use a switch detection signal and a random number generator (Verilog `$random` system call). The latter is required to generate probabilistic faults. Regarding the sabotaged modules, we alter the outputs of combinational components and of memory blocks in the evaluated decoders.

### 3.3.2. Decoder Basic Block Error Profile Characterization

This section is concerned with the error profile characterization of the basic building blocks utilized to construct the LDPC decoders under investigation. The main goal is to derive the error profile block Primary Outputs (POs), which further serves as guidance mean for judiciously performing fault injection. In this endeavor we assume that decoder basic blocks are implemented by means of a standard cell CMOS technology and derive their POs error profiles and their direct implications on performing fault injection, by traversing the following steps:

- Perform standard cells statistical timing characterization when exposing them to process and voltage variations.
- Identify the worst propagation path for each PO by means of timing analysis.
- Derive each PO delay distribution (statistical moments) based on the timing profile of the afferent path standard cells constituents.
- Assess each PO error profile for based on its delay distribution.

By following this procedure we present the simulation results obtained for the main combinational and sequential logic blocks for four varieties of LDPC decoders: Min-Sum (MS) decoder, Self-Corrected Min-Sum (SCMS) decoder, Finite Alphabet Iterative Decoder (FAID), and Stochastic Decoder (SD).

Our approach relies on the RTL/gate abstraction level fault tolerance aware design flow illustrated in Figure 3-5. We note that in the reign of silicon structures fundamental randomness, a device operation is better described as a stochastic process. For process and voltage variations likely to be encountered by a given circuit during run-time, a circuit path delay is a random variable and therefore a primary objective to enable error profiling and further fault-tolerance analysis, is to compute this random variation characteristics (e.g., distribution, statistical moments). To this end, first a Standard Cells (SCs) library is augmented with appropriate delay statistical characteristics. A circuit specified at the RTL level for instance, is synthesized using the library SCs.

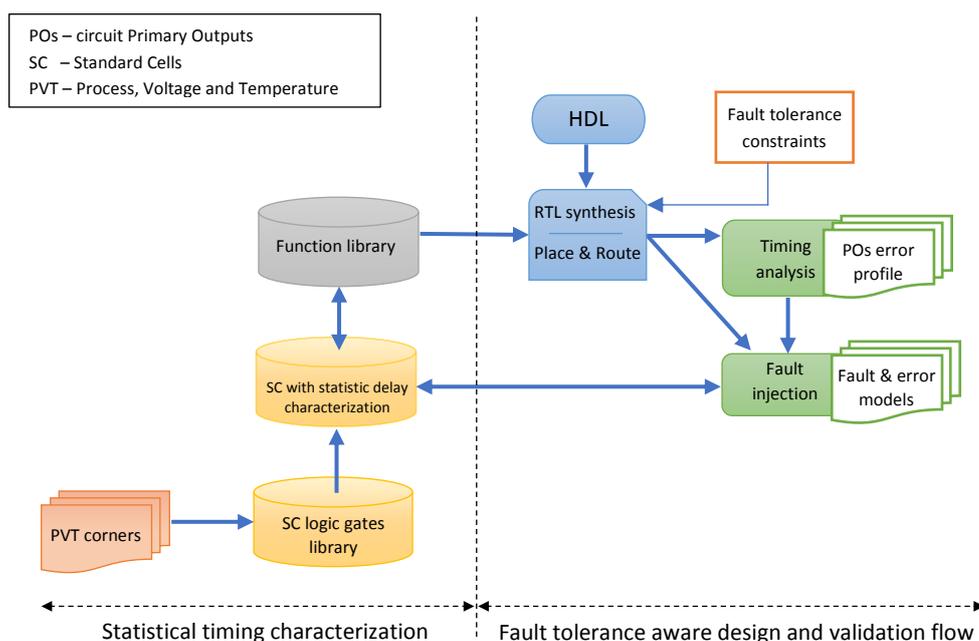


Figure 3-5: Fault Tolerance Aware Design Flow at RTL/gate Abstraction Level

A static timing analysis is subsequently performed to determine the worst timing path for each primary output of the circuit. A primary output delay statistics can then be derived based on the delay statistics of the SCs composing the primary output path. Based on the delay statistics, the probability profile for the output to be in an erroneous logic state easily follows. The error profile for each circuit primary output is of interest, as it enables to judiciously inject faults in the circuit for subsequent circuit fault-tolerance aware optimizations.

### 3.3.2.1. Standard cells statistical timing characterization

The first step in the flow depicted in Figure 3-5, consists in the statistical timing characterization of the standard cells in a technology library. Specifically, each standard cell is augmented with its propagation delay probability distribution over different process and voltage corners, according to the methodology presented in Deliverable D2.1 [i-RISC/D2.1].

Generally speaking, each standard cell operating in sub-threshold regime is exposed to normally distributed process (e.g., the cell comprising CMOS transistors oxide thickness and threshold voltage) and voltage (e.g., the supply voltage) variations. For each sampling set of process and voltage variation data, the cell propagation delay is derived as a mean between the measured rising and falling propagation delays which correspond to the two possible output switching situations, i.e., the output undergoing transition from logic “1” to logic “0”, and viceversa. The set of cell propagation delay values obtained for all process and voltage corner cases, is found to exhibit the same Probability Density Function (PDF) trend for all standard cells, namely, it follows an Inverse Gaussian (IG) distribution [i-RISC/D2.1].

The set of standard cells we utilize in this section is comprised of 45nm CMOS {NAND2, INV, DFF} cells (corresponding to a NAND gate, an inverter gate, and a D flip-flop, respectively), with driving strength X1. Instead of the commonly employed set of logic gates, i.e., NOT, AND, OR, XOR, NAND, NOR, XNOR, we opted for the universal NAND gates as they best serve the purpose of our

investigations, but also due to their salient features such as modularity, regularity, and logic flexibility to perform a variety of logic functions, as well as their potential in RTL circuit optimization, which may lead to a faster and/or more compact circuit.

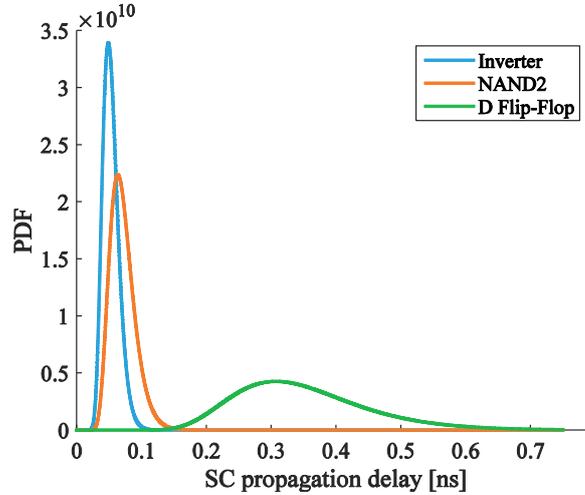


Figure 3-6: SC Propagation Delay PDF

Figure 3-6 depicts the obtained propagation delay Probability Density Function (PDF), which is given by an Inverse Gaussian distribution with the parameters summarized in Table 3-2, for each standard cell in our library.

Table 3-2: SC Propagation Delay IG Distribution Parameters

Standard Cell	IG parameters	
	Mean $\mu$ [ $10^{-11}$ ]	Shape $\lambda$ [ $10^{-10}$ ]
INV	5.3	9.5
NAND2	7.1	9.5
DFF	35	40.3

### 3.3.2.2. Circuit primary outputs error profile characterization

As graphically illustrated in Figure 3-5, the second step is first concerned with the error profile characterization for each primary output of a given circuit, based on: (i) each of its primary outputs worst timing path and (ii) the SCs propagation delay PDFs obtained at the previous step described in Section 3.3.2.1.

For a given circuit, specified at RTL level for instance, logic synthesis is first performed using a commercial tool, e.g., Synopsys DC compiler, Cadence RTL Encounter, and the circuit is mapped into the {NAND2, INV, DFF} standard cells. A static timing analysis is then carried out in order to determine for each circuit PO its worst timing path, i.e., the longest sensitizable path from a primary input to the primary output under consideration.

For illustrative reason, we employ as discussion vehicle a sole primary output of a certain circuit: the output corresponding to the circuit timing critical path (under the assumption that the circuit has a single critical path). The delay profile of the output can then be derived by a linear superposition of the individual path SC constituents delay profiles [i-RISC/D2.1]. For instance, for a critical path being

composed of 2 INV cells and 3 NAND2 cells, the mean  $\mu_o$  and shape  $\lambda_o$  parameters of the IG output propagation delay distribution are approximated as:

$$\begin{cases} \mu_o = 2 \cdot \mu_{INV} + 3 \cdot \mu_{NAND2} \\ \lambda_o = 2 \cdot \lambda_{INV} + 3 \cdot \lambda_{NAND2} \end{cases}$$

The Probability Density Function (PDF) and Cumulative Distribution Function (CDF) associated to the output propagation delay are depicted in Figure 3-7 and Figure 3-8, respectively, for the critical path output case discussed above.

Note that (CDF) and PDF of a random variable – the output propagation delay in this case – can be derived from one another by means of integration and differentiation. In the sequel we opt for CDF, as it conveys more straightforwardly the probability figures for subsequent derivation of the output delay error profile. Given the IG statistical moments of the output delay random variable, the probability of the delay to have a value less than or equal to a certain value  $\tau_{pL}$ , can be straightforwardly determined from the associated CDF. The direct implication is that based on a CDF, a timing error profile can be easily deduced, for given path delay constraints. Specifically, the probability  $CDF(\tau_{pL})$  that a  $\tau_{pL}$  timing constraint is satisfied (i.e., the output path delay is smaller than  $\tau_{pL}$ ) reflects the probability of the output signal to be in error. A smaller CDF probability for a delay constraint implies a higher probability that the output signal is in erroneous logic state. The inverse proportional relation between the output delay CDF probability and the output signal error probability is graphically illustrated in Figure 3-8 by the CDF curve gradient color (from red – higher error probability, to green – lower error probability).

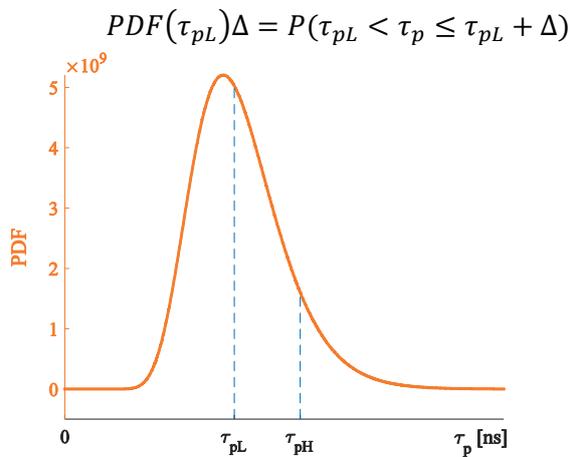


Figure 3-7: Output Propagation Delay IG PDF

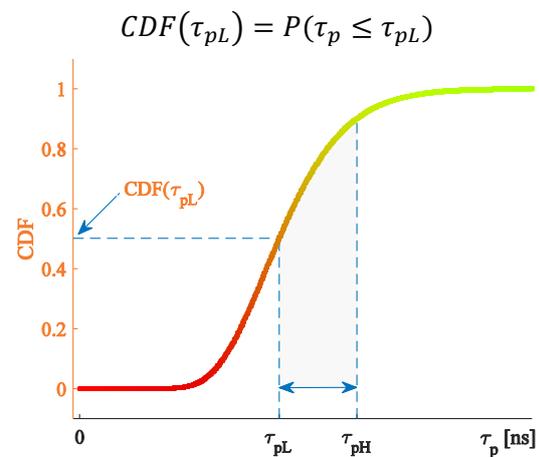


Figure 3-8: Output Propagation Delay IG CDF

The error profile of each circuit PO serves further as guidance mean for the fault injection techniques. For instance, in Figure 3-8, for  $\tau_p \in [0, \tau_{pL})$  the output signal probability to be in an erroneous state is very high, which implies that in this case the path exhibits an increased degree of susceptibility to faults and errors, and thus this situation should be avoided in the context of fault injection scenarios. Following the same line of reasoning, for  $\tau_p > \tau_{pH}$  the output signal error probability is very low, which implies that the path in this situation is reliable over process and voltage variations, and thus its suitability for fault tolerance related analysis and optimization is precluded. Hence, the interval of interest for judicious fault injection is given by  $\tau_p \in [\tau_{pL}, \tau_{pH}]$ .

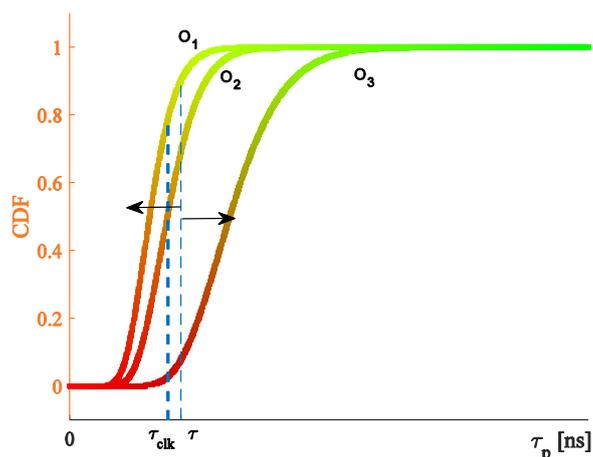


Figure 3-9: Circuit POs Delay CDFs

Figure 3-9 reflects the error profile of a circuit with three primary outputs, denoted by  $O_1$ ,  $O_2$ , and  $O_3$ , which has a maximum delay (clock period) equal to  $\tau_{clk}$ . Based on the POs error profiles the circuit fault tolerance status can be assessed for varying propagation delays (as a result of faults/errors), which may or may not satisfy the maximum clock constraint. For each sample value  $\tau$ , it can be deduced from the POs error profiles which are the best suited outputs for fault injection – in this case, it can be observed in Figure 3-9, that  $O_2$  is better suited for fault injection, as  $O_3$  and  $O_1$  have a low and respectively a high error probability, and thus shall be disregarded for fault injection. Thus, based on POs error profiles, for its given maximum clock period  $\tau_{clk}$ , the most suitable output for fault injection can be selected.

Using the above methodology, the basic building blocks of four LDPC decoders are characterized in the following subsection.

### 3.3.2.3. Error profile simulation results for LDPC decoders

Four LDPC decoder architectures specified at RTL level are considered, namely: MS, SCMS, FAID, and SD. Note that all decoders are synchronous and mainly divided into the following logic blocks: Variable Node processing Unit (VNU), Check Node processing Unit (CNU), barrel shifter, memory blocks, and finite state machine for the control signals. The decoders' architecture synopsis and the ports description are presented in detail in Section 3.5.

Blocks, which are proprietary to decoder architecture, e.g., VNU, are individually characterized for each decoder. For convenience, the logic blocks which are common to all decoders, i.e., the CNU block, the barrel shifter and the memory blocks, will have their error profile results presented only once – in the MS decoder.

For each decoder architecture statistical delay and error profile figures are presented in a hierarchical manner, i.e., block-wise, for each output. Specifically, we analyze the following:

- For the MS decoder: 3 combinational stages of the CNU logic, 4 combinational stages of the VNU logic, barrel shifter, and memory block.
- For the SCMS decoder: 5 combinational stages of the VNU unit.
- For the FAID decoder: 3 combinational stages and LUT of the VNU logic.

- For the SD decoder: 1 combinational stage CNU, 1 combinational stage VNU, and Random Number Generator (RNG) comparator.

For each basic building block of the considered decoders, all output signals are characterized with respect to their error profile over voltage and CMOS process variations and their afferent CDF is obtained. Table 3-3 to Table 3-6 summarize the worst timing path constituent standard cells for each output signal. Using the statistical characterization of the standard cells over processes and voltage variations, the CDF curve for each output is derived, according to the method described above. Figure 3-10 to Figure 3-30 depict the CDF curves for each output, decoder building block wise.

Based on these CDF curves, the outputs more prone to a faulty behavior provide the guidelines for performing a judicious fault injection, as presented in Section 3.3.3.

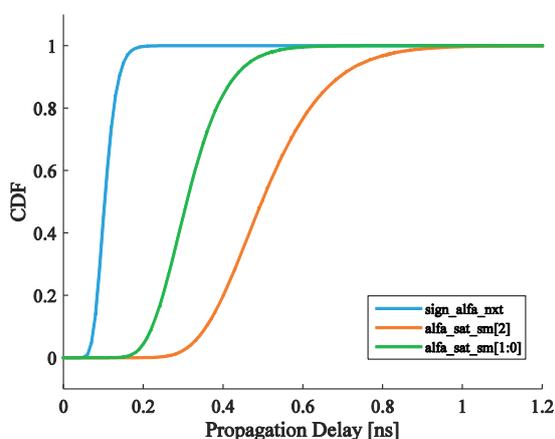


Figure 3-10: MS Decoder CNU Combinational Stage 1

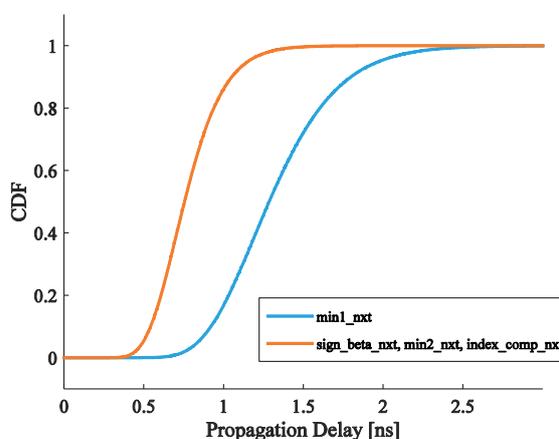


Figure 3-11: MS Decoder CNU Combinational Stage 2

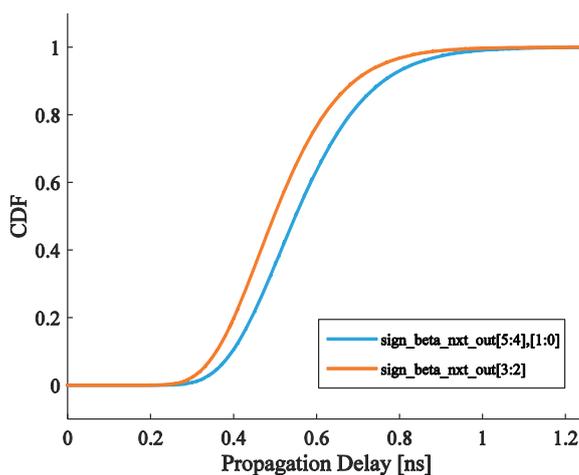


Figure 3-12: MS Decoder CNU Combinational Stage 3

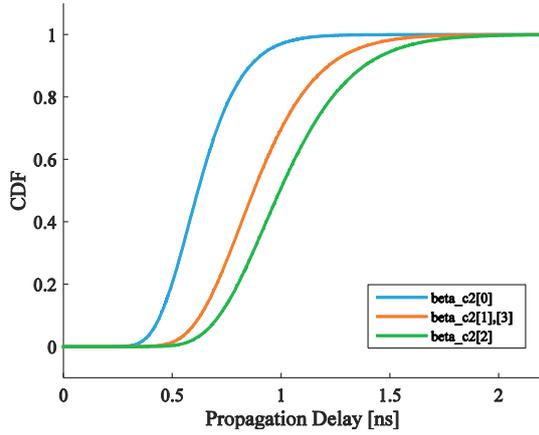


Figure 3-13: MS Decoder VNU Combinational Stage 1

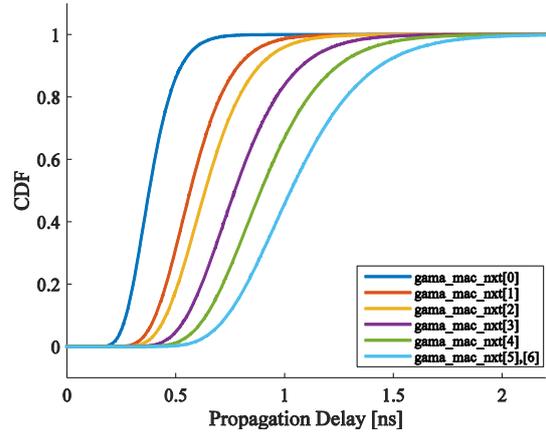


Figure 3-14: MS Decoder VNU Combinational Stage 2

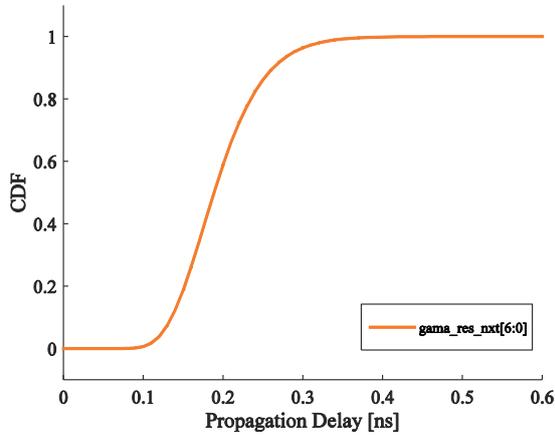


Figure 3-15: MS Decoder VNU Combinational Stage 3

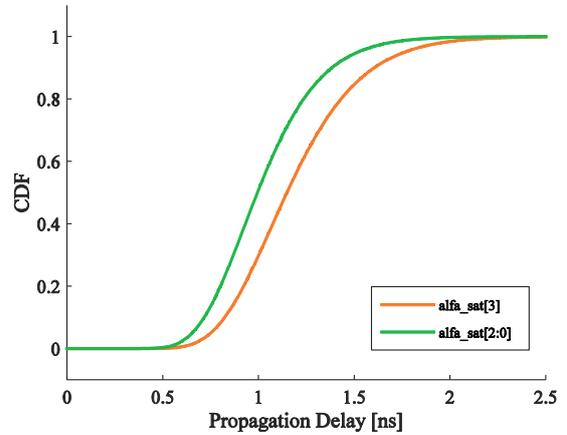


Figure 3-16: MS Decoder VNU Combinational Stage 4

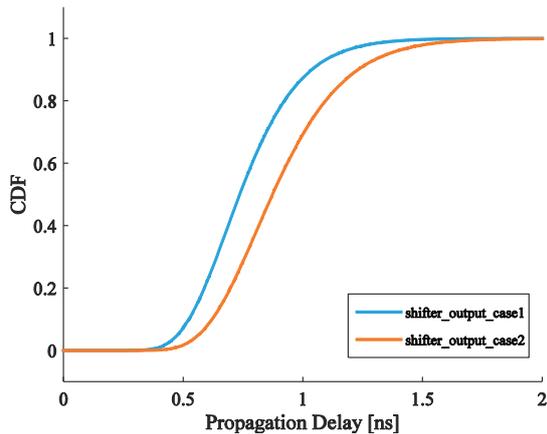


Figure 3-17: MS Decoder Barrel Shifter

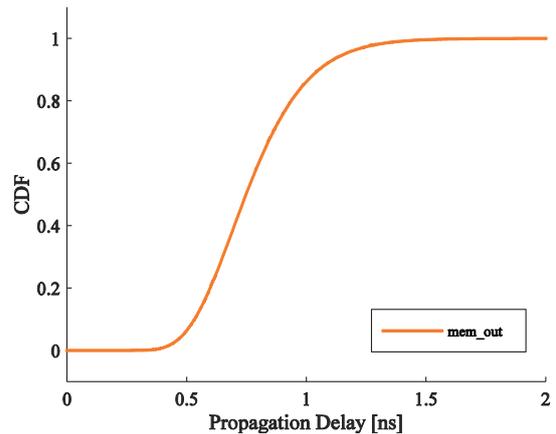


Figure 3-18: MS Decoder Memory Block

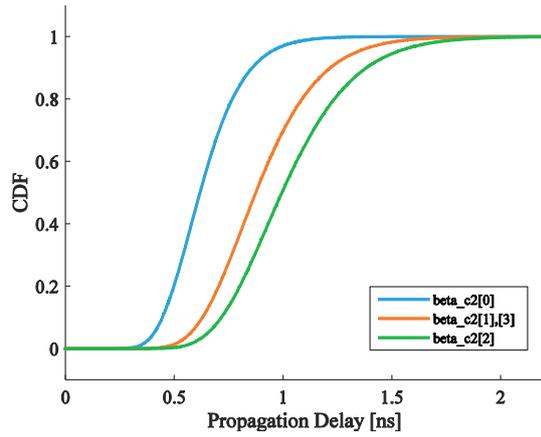


Figure 3-19: SCMS Decoder VNU Combinational Stage 1

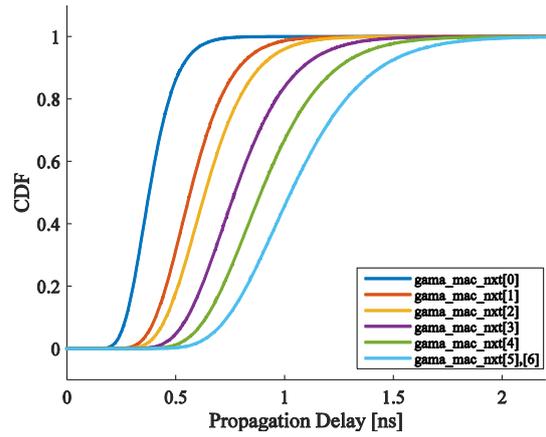


Figure 3-20: SCMS Decoder VNU Combinational Stage 2

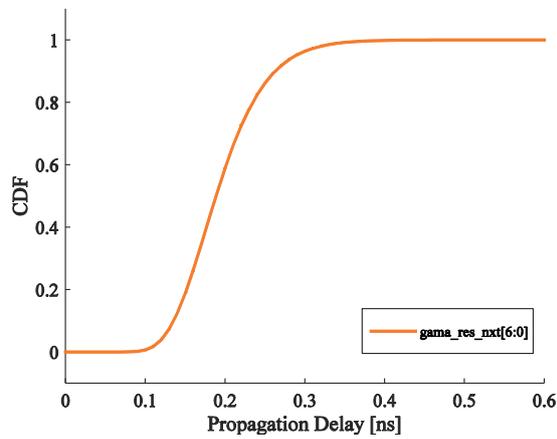


Figure 3-21: SCMS Decoder VNU Combinational Stage 3

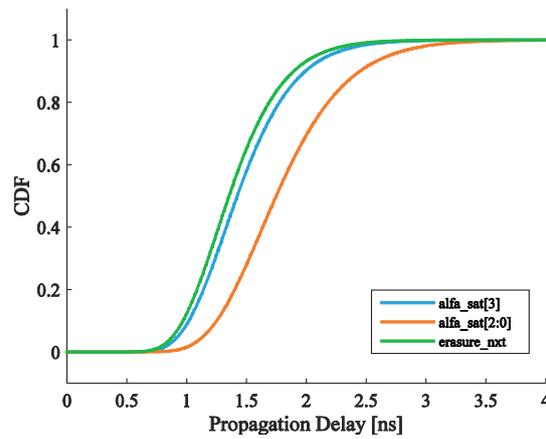


Figure 3-22: SCMS Decoder VNU Combinational Stage 4

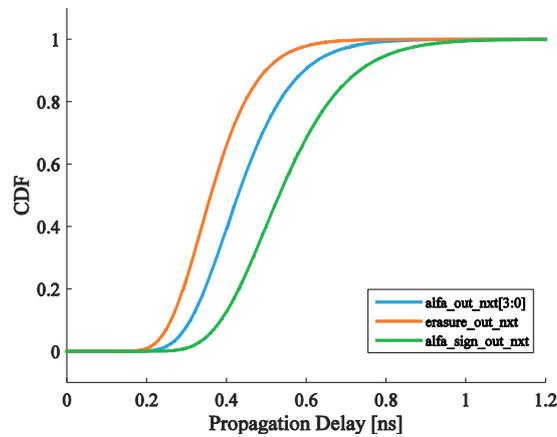


Figure 3-23: SCMS Decoder VNU Combinational Stage 5

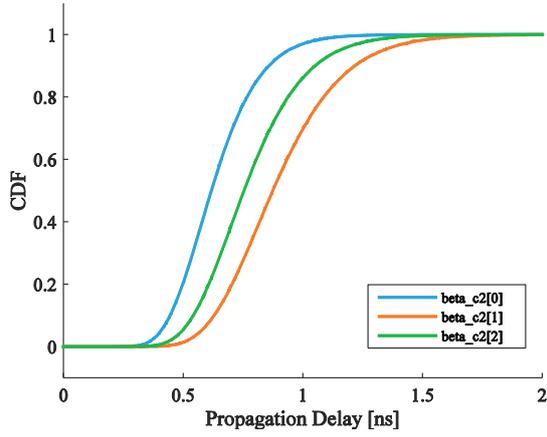


Figure 3-24: FAID Decoder VNU Combinational Stage 1

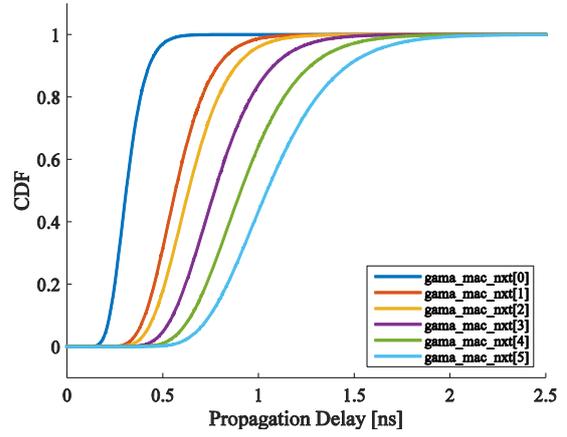


Figure 3-25: FAID Decoder VNU Combinational Stage 2

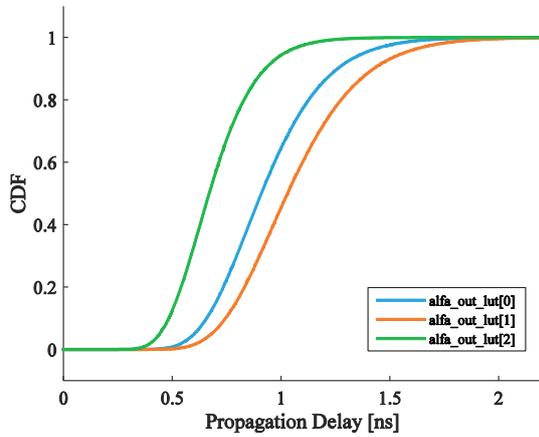


Figure 3-26: FAID Decoder VNU LUT

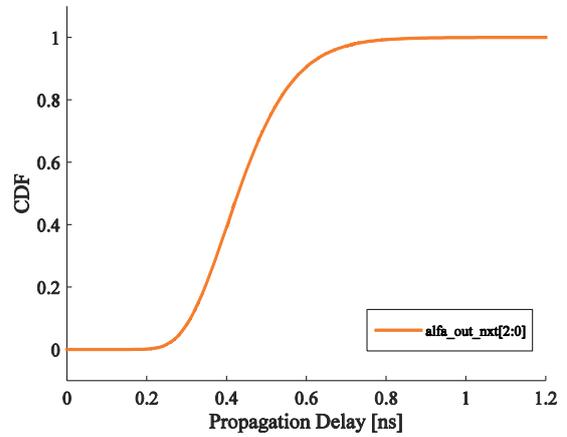


Figure 3-27: FAID Decoder VNU Out Stage

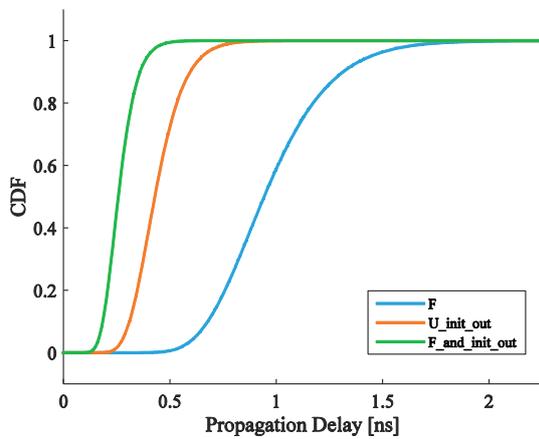


Figure 3-28: SD Decoder VNU Combinational Stage

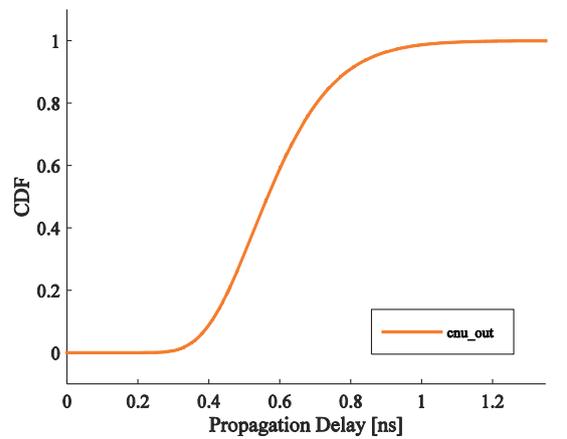


Figure 3-29: SD Decoder CNU Combinational Stage

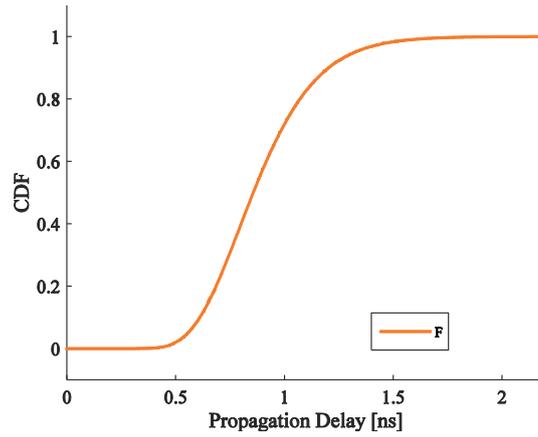


Figure 3-30: SD Decoder RNG

Table 3-3: FAID Decoder

FAID Decoder	Output signal	# INV	# NAND2	Legend
FAID VNU combinational Stage 1				
	beta_c2[0]	4	6	—
	beta_c2[1]	5	9	—
	beta_c2[2]	4	8	—
FAID VNU Combinational Stage 2				
	gama_mac_nxt[5]	3	13	—
	gama_mac_nxt[4]	3	11	—
	gama_mac_nxt[3]	3	9	—
	gama_mac_nxt[2]	3	7	—
	gama_mac_nxt[1]	3	6	—
	gama_mac_nxt[0]	2	3	—
FAID VNU LUT				
	alfa_out_lut[0]	7	8	—
	alfa_out_lut[1]	8	9	—
	alfa_out_lut[2]	5	6	—
SCMS VNU Out Stage				
	alfa_out_nxt[2:0]	3	4	—

Table 3-4: MS Decoder

MS Decoder	Output signal	# INV	# NAND2	#DFF	Legend
<b>MS CNU combinational stage 1</b>					
	sign_alfa_nxt	2	-	-	—
	alfa_sat_sm[2]	3	5	-	—
	alfa_sat_sm[1:0]	2	3	-	—
<b>MS CNU combinational stage 2</b>					
	sign_beta_nxt[5:0]	4	8	-	—
	min2_nxt[2:0]	4	8	-	—
	min1_nxt[2:0]	5	15	-	—
	index_comp_nxt[2:0]	4	8	-	—
<b>MS CNU combinational stage 3</b>					
	sign_beta_nxt_out[5:4] [1:0]	4	5	-	—
	sign_beta_nxt_out[3:2]	3	5	-	—
	sign_beta_nxt_out[1:0]	4	5	-	—
<b>MS VNU Combinational Stage 1</b>					
	beta_c2[3], [1]	5	9	-	—
	beta_c2[2]	6	10	-	—
	beta_c2[0]	4	6	-	—
<b>MS VNU Combinational Stage 2</b>					
	gama_mac_nxt[6:5]	4	12	-	—
	gama_mac_nxt[4]	4	10	-	—
	gama_mac_nxt[3]	3	9	-	—
	gama_mac_nxt[2]	3	7	-	—
	gama_mac_nxt[1]	3	6	-	—
	gama_mac_nxt[0]	2	4	-	—
<b>MS VNU Combinational Stage 3</b>					
	gama_res_nxt[6:0]	1	2	-	—
<b>MS Decoder VNU Combinational Stage 4</b>					
	alfa_sat[3]	3	8	-	—
	alfa_sat[2:0]	5	13	-	—
<b>MS Decoder Barrel Shifter</b>					
	shifter_output_case2 (*)	1	12	-	—
	shifter_output_case1 (*)	1	10	-	—
<b>MS Decoder Memory Block</b>					
	mem_out	4	3	1	—

(\*) The signal shifter\_output\_case1 corresponds to the following bits of the shifter output signal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53 while the signal shifter\_output\_case2 corresponds to the following bits of the shifter output signal: 10, 12, 34, 27, 31.

Table 3-5: SCMS Decoder

SCMS Decoder	Output signal	# INV	# NAND2	Legend
SCMS VNU combinational stage 1				
	beta_c2[3], [1]	5	9	
	beta_c2[2]	6	10	
	beta_c2[0]	4	6	
SCMS VNU Combinational Stage 2				
	gama_mac_nxt[6:5]	4	12	
	gama_mac_nxt[4]	4	10	
	gama_mac_nxt[3]	3	9	
	gama_mac_nxt[2]	3	7	
	gama_mac_nxt[1]	3	6	
	gama_mac_nxt[0]	2	4	
SCMS VNU Combinational Stage 3				
	gama_res_nxt[6:0]	1	2	
SCMS VNU Combinational Stage 4				
	alfa_sat[3]	5	17	
	alfa_sat[2:0]	6	21	
	erasure_nxt	5	16	
SCMS VNU Combinational Stage 5				
	alfa_out_nxt[3:0]	3	4	
	erasure_out_nxt	3	3	
	alfa_sign_out_nxt	5	4	

Table 3-6: SD Decoder

SD Decoder	Output signal	# INV	# NAND2	Legend
SD VNU combinational Stage				
	F	5	10	
	U_init_out	3	4	
	F_and_init_out	1	3	
SD CNU Combinational Stage				
	cnu_out	3	6	
SD RNG				
	F	2	11	

### 3.3.3. Fault Map Generation

We recall that the SFI based evaluation purpose is to analyze the LDPC decoder error correction capability under timing violations, which are due to the overclocking at very low supply voltages. To this end we introduced saboteurs at the outputs of any combinational component located between two registers (a pipeline stage) and at the memory blocks outputs. To let saboteurs capture as accurate as possible real life situations we have now to deduce their error probabilities setting based on the CDFs derived in the previous section.

We note that the LDPC decoders have been designed for a single clock domain. Therefore, combinational stages with lower latencies are expected to present lower failure probability when compared with stages with higher latencies for a given clock period. Furthermore, as indicated by the CDFs for each stage presented in Figure 3-10 to Figure 3-30 the error probabilities of individual outputs for a specific combinational stage differ due to the different latency from the input to that particular output. Thus, the failure probabilities for each output of each stage for a given clock period can be extracted from the CDFs in the Figures. By considering this type of error distribution and probability, dependent on clock frequency, we simulate in an accurate and realistic way the timing error occurrence across the entire design.

Table 3-7: Failure Probabilities for Analyzed Decoders for Different Clock Frequencies

Clock Period (ns)	Min-Sum		Self-Corrected Min-Sum		FAID	
	Min*	Max	Min*	Max	Min*	Max
5.50	1.06E-09	1.06E-09	1.07E-09	1.07E-09	1.06E-09	1.06E-09
4.00	3.46E-09	2.92E-06	2.92E-06	5.78E-09	3.46E-09	2.92E-06
2.50	2.53E-09	5.04E-03	5.04E-03	2.53E-09	2.53E-09	5.04E-03
2.20	2.72E-09	1.93E-02	1.93E-02	2.72E-09	2.72E-09	1.93E-02
1.90	1.43E-08	6.73E-02	6.73E-02	1.43E-08	1.43E-08	6.73E-02
1.70	2.40E-07	1.43E-01	1.43E-01	4.02E-09	2.40E-07	1.43E-01

\* non-zero

Table 3-7 presents the minimum and the maximum failure probabilities for the 3 decoders evaluated by means of simulated fault injection. We note that even though the Stochastic Decoder basic building blocks have been evaluated (see Figure 3-28 to Figure 3-30) we could not complete its SFI based evaluation due to technical hurdles in its integration in the HDL/C++ co-simulation framework and due to time shortage (SFI experiments are extremely time consuming even we carried experiments in parallel on 6 workstations). Table 3-8 summarize the average failure probabilities for the components used in the 3 evaluated decoders, i.e., MS, SCMS, and FAID, as derived for the CDFs of each combinational stage and memory block outputs, for different clock frequency values.

Table 3-8: Average Values of Failure Probabilities for Considered Components in the Analyzed Decoders

Clock Period (ns)	Memories	Barrel Shifters	VNU MS	VNU SCMS	VNU FAID	CNU MS/SCMS	CNU FAID
5.50	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.28E-10	9.70E-11
4.00	0.00E+00	0.00E+00	3.14E-08	2.47E-08	2.26E-09	3.51E-07	2.66E-07
2.50	9.64E-07	2.79E-06	1.92E-04	1.51E-04	4.61E-05	6.05E-04	4.58E-04
2.20	1.28E-05	2.71E-05	9.94E-04	7.81E-04	3.12E-04	2.33E-03	1.76E-03
1.90	1.60E-04	2.62E-04	4.80E-03	3.77E-03	1.98E-03	8.13E-03	6.18E-03
1.70	8.28E-04	1.17E-03	1.30E-02	1.02E-02	6.48E-03	1.75E-02	1.33E-02

For the considered technology and supply voltage, we have performed simulations corresponding to clock periods from 5.5ns down to 1.7ns, which correspond to error probabilities higher than  $10^{-9}$ . As expected increasing the clock frequency leads to an increase in output failure probabilities but this failure rate increase is not uniform across the circuit, as combinational stages with higher latency will have higher failure rates.

Figure 3-31 depicts the number of active fault locations (signals with a non-zero error probability) for the processing units of the analyzed LDPC decoders. It can be observed that this number increases with the decrease in the considered clock period. The figure indicates that for a clock period of 2ns, the MS decoder CNU has the highest number of active fault locations, while its VNU has the lowest.

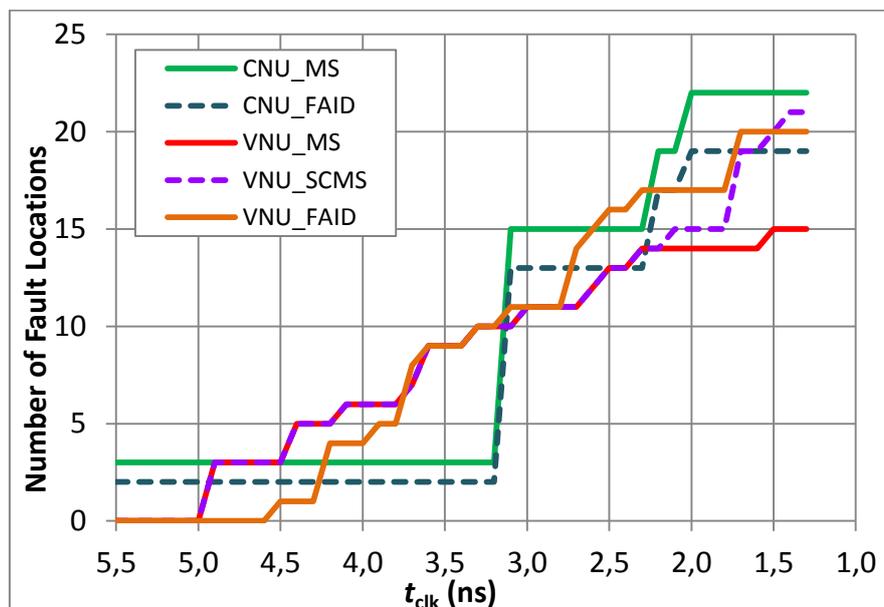


Figure 3-31: Number of Active Fault Locations in Processing Units

Figure 3-32 depicts the ratio between the per-block number of active fault locations (signals subject to fault insertion by means of saboteurs) and the total number of sabotaged wired.

One can conclude from the figure that: (i) FAID and MS/SCMS CNU's experience errors with a probability higher than  $10^{-9}$  for a clock period of 5.5ns or lower, (ii) MS/SCMS VNU's errors appear for a clock period of 5ns or lower while for FAID VNU they appear for a clock periods of 4.5ns or lower, and (iii) for barrel shifters, errors with a probability higher than  $10^{-9}$  start to occur at a clock period of 3.7ns. We also note that for a clock period of 3.9ns, all modules have more than half of their sabotaged wires affected by probabilistic errors with occurrence probability higher than  $10^{-9}$ .

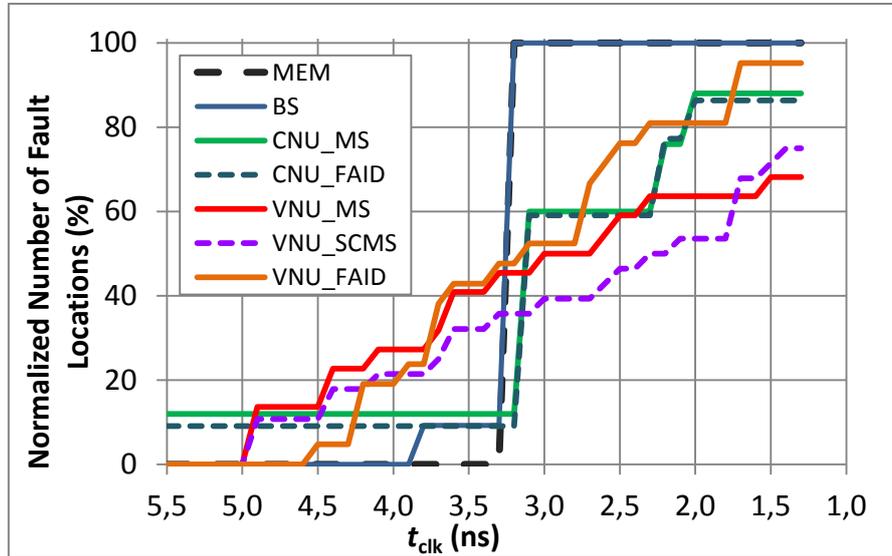


Figure 3-32: Ratio between Active Fault Locations and Total Number of Sabotaged Wires in Considered Blocks

Figure 3-33 depicts the per block average error probability on a logarithmic scale. This figure indicates that FAID VNU has a lower average error probability with respect to the MS and SCMS VNU for the same clock period. In particular, for a clock period of 1.9ns the average error probability for FAID VNU is almost half with respect to the MS and SCMS VNU. We can also observe that CNU units present the highest average error probability while memory blocks, followed by barrel shifters, have one order of magnitude lower error probabilities than the processing units.

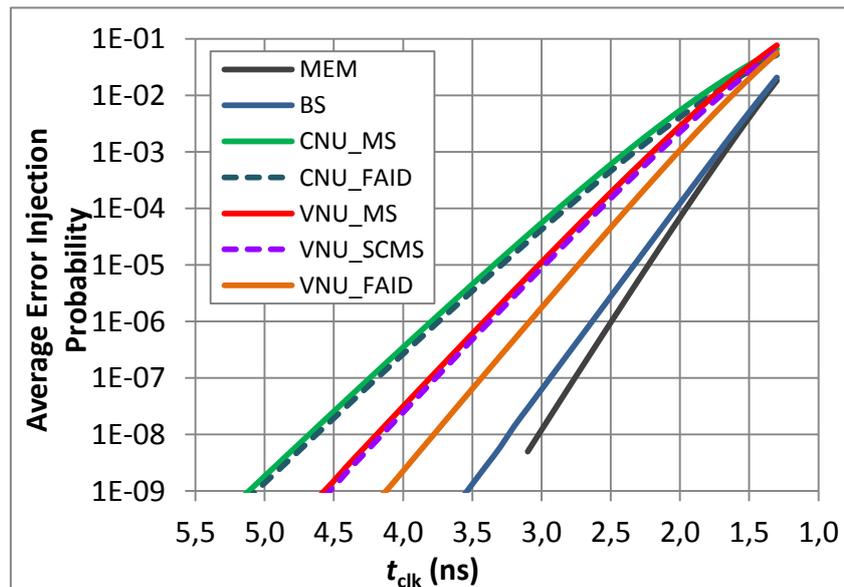


Figure 3-33: Average Probability of Errors in the Considered Blocks (Logarithmic Scale)

Figure 3-34 depicts the estimated number of per iteration fault locations, which might be activated in the entire decoder on a logarithmical scale. Figure 3-35 depicts the estimated number of possible per iteration active fault locations in the entire decoder, for clock periods between 1.9ns and 2.5ns. These two figures indicate that the FAID decoder has significant less active faults per iteration with respect to MS and SCMS decoders. For example, for a clock period of 1.9ns, the number of active faults in FAID decoder is half of the active faults to be injected in MS or SCMS decoder.

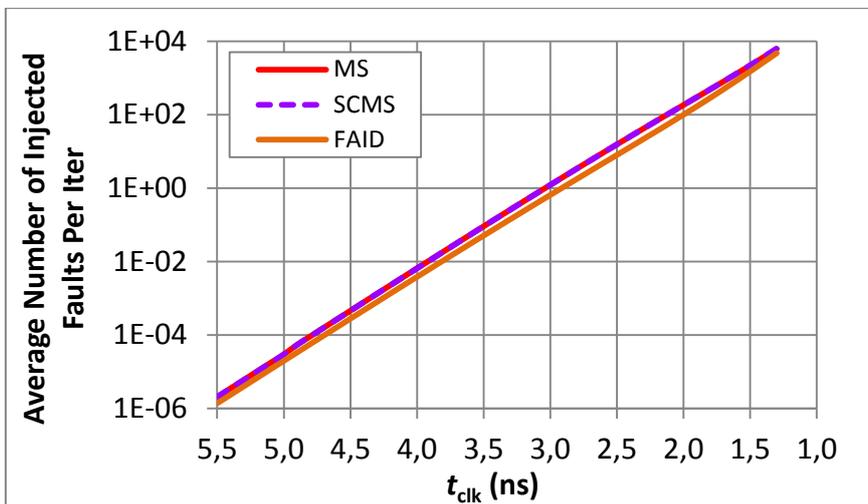


Figure 3-34: Average Number of per Iteration Injected Faults (Logarithmic Scale)

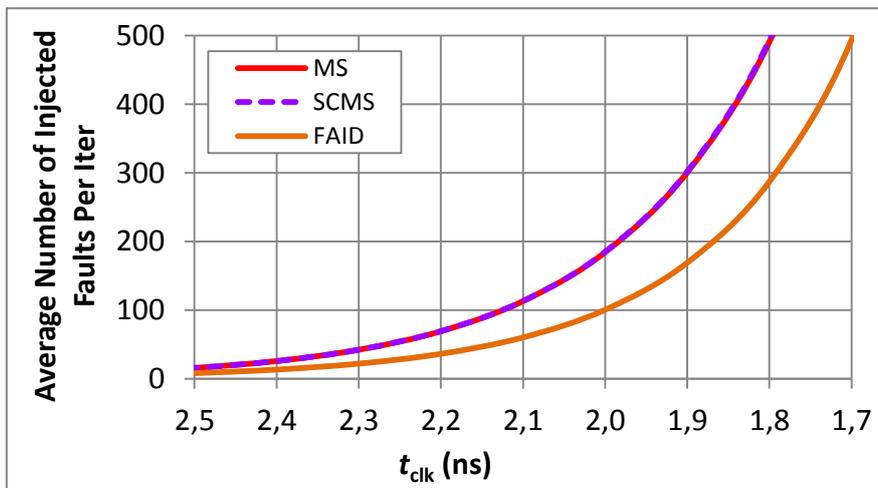


Figure 3-35: Average Number of per Iteration Injected Faults (Zoom - Linear Plot)

The generated fault maps provide the timing probabilistic faults, which take into account the architectural and implementation details of the analyzed decoders. Based on these maps, we can observe the error conditions under which the decoders will show decoding performance degradation. In our experiments we consider various clock periods, starting from 5.5ns (when errors start occurring in the processing units), 3.1ns (when errors appear in the memories), then 2.5ns, 2.2ns, 1.9ns and 1.7ns, the last values corresponding to the region where the average number of injected faults per iterations start increasing at a higher pace (as one can observe in Figure 3-35).

### 3.4. Utilized LDPC Codes

In this section we describe the codes that sit behind the LDPC decoders implemented and evaluated in this deliverable. Before presenting them into details we introduce an LDPC construction methodology.

#### 3.4.1. Protograph Based LDPC Construction

First introduced by [Thorpe03], a binary protograph is defined as a small bipartite graph from which a larger graph is obtained by a so-called “copy-and-permute” procedure. The procedure can be summarized as follows:

- *Copy step*: first, the protograph is copied  $L$  times to obtain  $L$  replicas.  $L$  is arbitrary defined to achieve the targeted codeword length.
- *Permute step*: then, the edges of the individual replicas are permuted among the replicas to obtain a single but larger graph, but under some restrictions in order to obtain an LDPC code with good enough error correction performance.

Finally, the permuted edge connections specify the non-zeros entries of the parity-check matrix associated with the resulting graph.

The protograph itself is generally described using its *adjacency matrix*  $H_B$  also called base matrix [Liva06]. A protograph is then a  $(M_b, N_b)$  matrix filled with integer values, which represent the number of edges between the  $i$ -th check node  $C_i$  of the protograph and the  $j$ -th variable node  $V_j$ . Note that using this representation enables to consider parallel edges, i.e., two nodes (a variable node and a check node) can be connected with more than one edge. These parallel edges however must be eliminated when building the larger graph using the copy-and-permute procedure to yield to a suitable representation of the code using a parity-check matrix.

In fact, the code ensemble defined by the protograph can be viewed as a structured sub-ensemble of the Low-Density Parity-Check (LDPC) code ensemble defined using the equivalent edge distribution. As an example, we consider the regular ( $d_v = 3$ ,  $d_c = 6$ ) LDPC code ensemble (variable nodes with degree 2 and check nodes with degree 4). The two following adjacency matrices are associated with a particular structured sub-ensemble of the regular ( $d_v = 3$ ,  $d_c = 6$ ) LDPC code ensemble:

2	1	0	3
1	2	3	0

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

Table 3-9: Different Types of Protograph for ( $d_v = 3$ ,  $d_c = 6$ ) LDPC Codes

Throughout the rest of this section, we denote by *type-I protograph* a protograph which contains only ‘0’ and ‘1’, and by *type-II protograph*, a protograph which could contain values greater than ‘1’. Usually, the protograph or the base matrix is issued from some previous optimization procedures based on density evolution [Thorpe03] or on a multidimensional EXIT chart analysis [Liva06].

Once the protograph (or equivalently the base matrix) has been selected, one aims at building a larger graph. The optimum way of selecting the permutations among the different replicas of the protograph is still an open issue, and one has to rely on heuristics and sub-optimum procedures. To simplify both the encoding and decoding storage, it is convenient to choose circulant permutations:

the final parity-check matrix  $H$  of the LDPC code is described as an  $(M, N)$  array of  $(L, L)$  (weight-one) circulant permutation matrices or  $(L, L)$  zeros matrices. The assignment of the circulants should be done carefully to avoid short cycles and it can be done using an instance of the Progressive-Edge-Growth (PEG) algorithm [Hu05] sometimes referred to as circulant-PEG or lift-PEG [Venkiah08]. We will employ in the rest of this section the following lifted graphs/codes definition:

*The operation consisting in replacing each non-zero entry with value  $d$  in the protograph base-matrix by a set of  $d$  non-overlapping circulant matrices is called “lifting”. The order of the lift  $L$  corresponds to the size of the circulant matrices.*

### 3.4.2. *i*-RISC Set of Matrices

For best performance in the error floor region, the Tanner graph should have the best possible topological properties in terms of cycles. That is, one should aim at the maximum possible girth, and minimum multiplicity of the number of cycles with minimum length. This is important because the trapping sets topologies are obtained by combination of several cycles, so the larger the cycles, the larger are the trapping sets [Liva06]. In the *i*-RISC project, we have adapted the Random-PEG algorithm proposed in [Venkiah08] in order to perform the lifting operation. This instance of the PEG algorithm is called *Lift-PEG* algorithm and it can be briefly described as follows:

For each non-zero entry in the protograph from column 1 to column  $N_b$ , choose the circulant, which maximizes the “local” girth of the graph and minimizes the number of small cycles created. The computation of the “local” girth and multiplicity is done with the help of the computation tree.

The principle of the Random-PEG algorithm is kept, but applied block by block, for each  $(L, L)$  circulant. To keep the complexity of the hardware decoder implementations within implementable bounds, a constant codeword length has been chosen,  $N=1296$  coded bits, for two different rates,  $R=0.5$  and  $R=0.75$ . Also, in order to verify the robustness of the proposed fault-tolerant decoders on different coding situations, 3 kinds of LDPC families have been proposed. The set of 6 codes considered in the project is presented in Table 3-10.

Table 3-10: Employed Set of LDPC Codes

Code Rate R	Regular $d_v = 3$			Regular $d_v = 4$			Irregular		
	$N$	$M$	$L$	$N$	$M$	$L$	$N$	$M$	$L$
R=1/2	1296	648	54	1296	648	54	1296	648	54
R=3/4	1296	324	27	1296	324	27	1296	324	27

The corresponding protographs have therefore size  $(12, 24)$ , for the rate  $R=1/2$ , and  $(12, 48)$ , for the rate  $R=3/4$  as presented in Figure 3-36 and the corresponding matrices are described in Appendix 1. The LDPC codes have all good properties in terms of girth, as they all have girth  $g=8$ , which means that the size of the minimum cycle is 8. This last property ensures good error correction performance for the designed LDPC codes.

We note however that in view of the fact that both voltage scaling and SFI experiments are very expensive in terms of execution time we had to limit our evaluations to LDPC decoders corresponding to  $R = 1/2$ ,  $d_v = 3$ ,  $N = 1296$ ,  $M = 648$ ,  $L = 54$ .

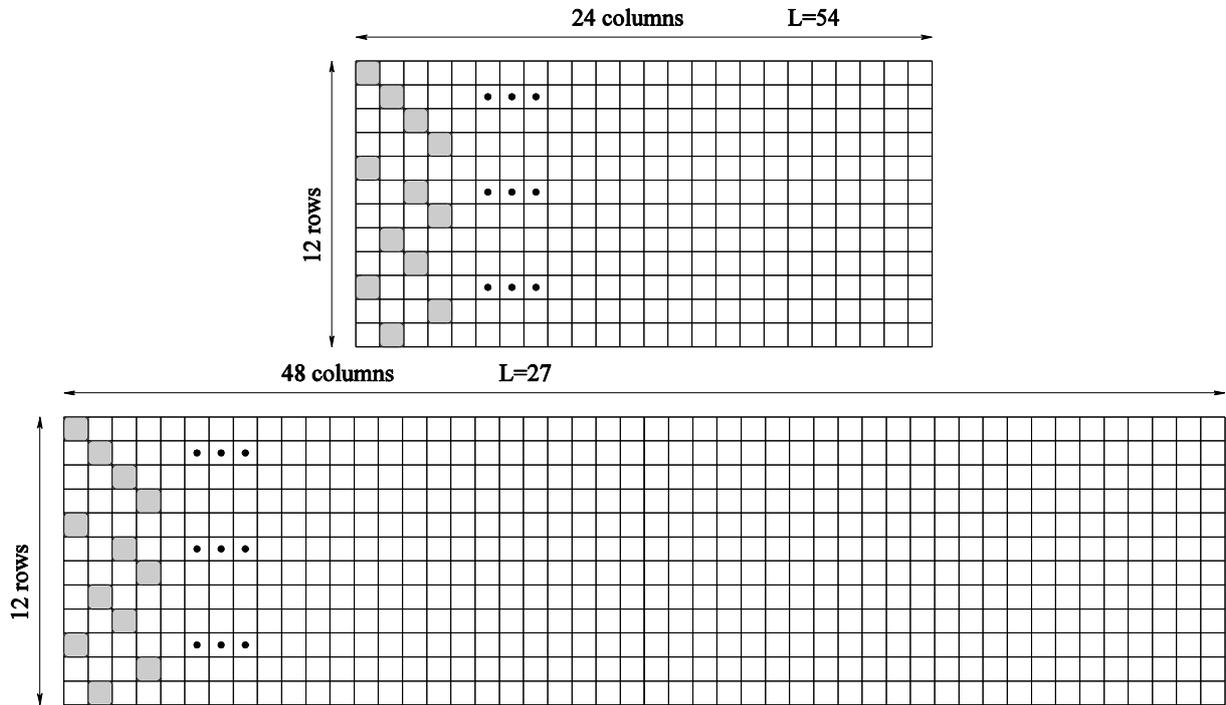


Figure 3-36: Protographs for  $(d_v = 12, d_c = 24)$  and  $(d_v = 12, d_c = 48)$  LDPC Codes

### 3.5. Decoder Architecture, Organization, and Implementation

In this section we describe the LDPC decoders we investigate by means of voltage scaling and fault injection from the hardware prospective. We give special attention to implementation relevant aspects by we also briefly discuss their basic operation principle when required for understanding architectural and implementation decisions.

#### 3.5.1. Min-Sum (MS)

Min-Sum decoding has been thoroughly investigated in WP3 [i-RISC/D3.1], where its robustness to simple theoretical error models has been assessed both analytically and by C simulations.

In flooded min-sum decoding is performed according to the Tanner graph associated to the  $H$  matrix: the Check Node Units (CNU) compute the check node messages (denoted as  $\beta$ ) based on the messages received from the Variable Node Units (VNU) (denoted as  $\alpha$ ); these updated  $\beta$  messages are passed back to the VNUs, which will update  $\alpha$  messages. The flooded MS decoding has as inputs the channel LLR messages, and consists of the following steps:

1. Initialization  $\alpha_i = \gamma_i$
2. Check Node Update  $\beta_{z,i} = \left( \prod_{j \in H(z) \setminus i} \text{sgn}(\alpha_{z,j}) \right) \times \min(\alpha_{z,j})$
3. Variable Node Update  $\alpha_{z,i} = \gamma_i + \sum_{j \in H(z) \setminus i} \beta_j$

#### 4. A-posteriori Update $\gamma_i = \gamma_i + \sum_{j \in H(z)} \beta_j$

Steps 2-4 are repeated until a codeword is found or the maximum number of iterations is reached. As indicated by the description above the MS algorithm has low hardware complexity as it mainly consists of additions and comparisons on a small number of bits.

The fully parallel flooded decoder represents the faithful implementation of the Tanner graph. It has a number of VNUs/CNUs equal to the number of columns/rows in the parity check matrix. It has the advantage of very high throughput but has a very high implementation cost, mainly due to the interconnection network [Stimming12] [Chandrasetty12]. Serialization is used in order to reduce the decoder cost at the expense of throughput. Because serial solutions have a limited number of processing units, messages are temporarily stored in memories. Several approaches of implementing such flooded architecture have been presented in [Chen11] [Park14].

We implemented an MS decoder developed for Quasi-Cyclic (QC) LDPC codes, which include the codes presented in Section 3.4, and has a number of processing units (both VNU and CNU) equal to the size of the circulant matrix (54). At both individual CNU and VNU level, messages are processed serial while the rows and the columns of the  $B$  matrix are processed in a serial manner. Due to the serialization, at both processing unit level, and at the  $B$  matrix level, the developed flooded decoder uses memories for message storage.

##### 3.5.1.1. Decoder General Description

The architecture of the flooded Min-Sum (MS) LDPC decoder is depicted in Figure 3-37. It consists of the following modules:

1. *Input Log Likelihood Ratio (LLR) memory* – this memory stores the channel messages, which will be used in the decoding process for variable node computations; the memory word size is equal to input LLR quantization (4 bits) multiplied by circulant size (54 for the given code); the depth of the memory is equal to the number of columns in the  $B$  matrix.
2. *Variable Node Unit (VNU) block* – it contains a number equal to the circulant size (54 for the analyzed QC-LDPC code) individual units; the 54 VNUs compute the corresponding variable-to-check messages ( $\alpha$ ) for a column in the  $B$  matrix, as well as the A-Posteriori LLR (AP-LLR).
3. *Variable-to-check message memory* – it stores the  $\alpha$  messages, which will be used in the check node computations; the memory word size is equal to the  $\alpha$  message size (4 bits) multiplied by the circulant size ( $54 \times 4$ ); the depth of this memory is equal to variable node degree multiplied by the number of columns in the  $B$  matrix ( $3 \times 24$ ).
4. *Variable-to-check message Barrel Shifter (BS)* – it provides the corresponding interconnection scheme between the VNU outputs to check node unit inputs; it has 6 multiplexer (MUX) levels and a number of circulant size multiplied by  $\alpha$  word size ( $54 \times 4$ ) MUXes per level.
5. *Check Node Unit (CNU) block* – it contains a circulant size individual check node units; the 54 units compute the corresponding check-to-variable messages ( $\beta$ ) for a row in the  $B$  matrix.
6. *Check-to-variable message memory* – it stores the  $\beta$  messages, which will be used in the variable node computations; the  $\beta$  message is stored in a compressed form; the size of the compressed  $\beta$  message is 15 bits; the memory word size is equal to circulant size multiplied by compressed  $\beta$  message size ( $54 \times 15$ ), while the memory depth is equal to the number of rows in the  $B$  matrix (12).

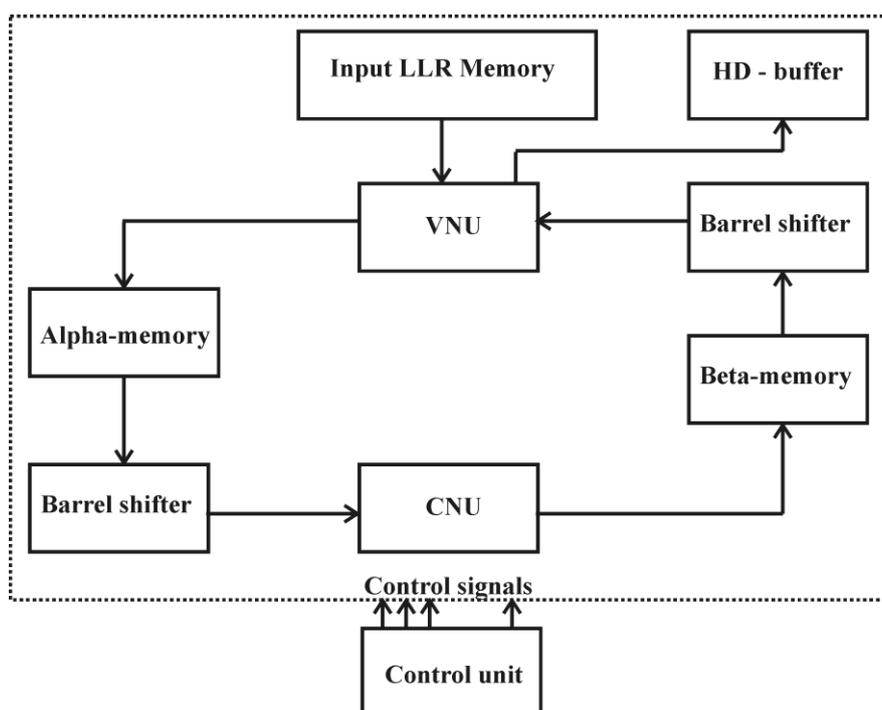


Figure 3-37: Flooded MS Architecture

7. *Check-to-variable message BS* – it provides the corresponding interconnection scheme between the CNU outputs to VNU inputs; it has 6 multiplexer (MUX) levels and a number of circulant size multiplied by compressed  $\beta$  word size ( $54 \times 15$ ) MUXes per level.
8. Hard-decision buffer and early-termination circuit – after each iteration it evaluates the parity check equations based on the signs of AP-LLR computed within the VNUs; furthermore, it provides the output of the decoder.
9. Control unit – it has the role of providing the appropriate sequence of operations during the decoding process; it provides: (i) the addresses for the 3 memories; (ii) the shift amounts for the 2 BS, (iii) the indexes of the  $\beta$  messages (iv) the corresponding control signals for both processing units (VNUs and CNUs) and the memories (read and write enables).

The  $\beta$  messages have been compressed in order to reduce the number of memory locations. Six  $\beta$  messages corresponding to a row in  $H$  matrix ( $6 \times 4$  bits/message = 24 bits) are compressed in the following way: 3 bits are used for the absolute value of the first minimum, 3 bits for the absolute value of the second minimum, 3 bits for the index of the first minimum, and 6 bits for the 6  $\beta$  messages' signs. Thus, 15 bits are required for the  $\beta$  messages corresponding to a row in the  $H$  matrix.

### 3.5.1.2. Decoder Modules Description

In order to reduce the number of memory read/write ports and to reduce the BS number serialization has been applied to both VNU and CNU. Thus, the 3  $\beta$  messages corresponding to a VNU and the 6  $\alpha$  messages corresponding to a CNU are read in 3 (for VNU) /6 (for CNU) consecutive clock cycles. Therefore, both the VNU module and the CNU module have one port for the input message and one port for the output message.

*VNU Module:* The VNU consists of 4 combinational stages, which perform the following operations:

- The first stage performs the conversion of  $\beta$  messages from the compressed form to the uncompressed Two's Complement (C2) representation; the corresponding beta to a VNU is computed as follows:
- The second stage performs the computation of the AP-LLR, using a 6-bit accumulator.
- The third stage is used for synchronization purposes and performs a freeze on the result of the AP-LLR.
- The fourth stage computes the  $\alpha$  message by subtraction the corresponding  $\beta$  message from the AP-LLR; in order to subtract the correct  $\beta$  message, a FIFO buffer is used for the C2 version of  $\beta$ ; in order to reduce size of the variable-to-check message memory, the 6-bit  $\alpha$  message obtained after the subtraction is reduced to 4 bits by applying a saturation.

The VNU outputs the 3  $\alpha$  messages corresponding to a column in the  $H$  matrix in a serial manner, in 3 consecutive clock cycles. Table 3-11 presents the VNU port description.

Table 3-11: MS VNU Ports List

Port name	In/Out	Size [bits]	Function
beta_in	I	15	Compressed $\beta$ message input
beta_index_in	I	3	Index of the current processed $\beta$ message
gama_in	I	4	Input LLR
start_proc_in	I	1	Control signal – start a new VNU operation
hard_dec_out	O	1	Hard decision output
alfa_out	O	4	Output $\alpha$ message
end_proc_out	O	1	Status signal

*CNU Module:* The CNU consists of 3 combinational stages, which perform the following operations:

- The first stage performs the conversion of input  $\alpha$  messages from C2 representation to Sign-Magnitude (SM) representation;
- The second stage performs the comparisons with previous absolute vales of the first and second minima, as well as the computation of the index for the first minimum;
- The third stage performs updates on the 6  $\beta$  messages signs.

The CNU has as input port the  $\alpha$  message (4 bits) and it outputs a single compressed  $\beta$  message (15 bits) corresponding to the 6  $\beta$  messages of a row in the  $H$  matrix. Table 3-12 presents the CNU port list.

Table 3-12: MS CNU Ports List

Port name	In/Out	Size [bits]	Function
alfa_in	I	4	Input $\alpha$ message
start_proc_in	I	1	Control signal – start a new VNU operation
beta_out	O	15	Compressed $\beta$ message output
end_proc_out	O	1	Status signal

### 3.5.2. Self-Correcting Min-Sum (SCMS)

Self-Corrected Min-Sum (SCMS) has been proposed by [Savin08] and targets the improvement of the LDPC error correction capability, mainly in the error floor region. It has been proven to be the most reliable MS based algorithm in Deliverable 3.1 [i-RISC/D3.1]. With respect to the MS algorithm, SCMS has a modified variable node update: if the new  $\alpha$  message has a different sign than the previous  $\alpha$  message, it is erased. In order to implement the variable node update for the SCMS, two more bits are required for each  $\alpha$  message update: the sign of the previous  $\alpha$  message and the erasure bit. The erasure bit is used in order to avoid two consecutive erasures on the same  $\alpha$  message.

#### 3.5.2.1. Differences With Respect to MS Decoder

The architecture of the flooded Self-Corrected Min-Sum (SCMS) LDPC decoder is depicted in Figure 3-38. With respect to the MS implementation the following differences are in place:

1. VNU is modified in order to perform the self-correction ( $\alpha$  message erasure) operation; two more 1-bit inputs and 1-bit outputs are added, which correspond to the previous sign of  $\alpha$  message and the erasure bit; a fifth combinational stage is added to the VNU; this stage performs the self-correction.
2. The erasure bits and the signs of the previous  $\alpha$  messages are stored in two separate memories with a word size equal to the circulant size (54), and a depth equal to the number of columns multiplied by variable node degree ( $24 \times 3$ ); thus, the SCMS decoder has increased memory requirements with respect to MS.

These two differences require the modification of the control unit in order to generate the corresponding read and write addresses for the erasure bit memory and the previous  $\alpha$  messages' signs memory.

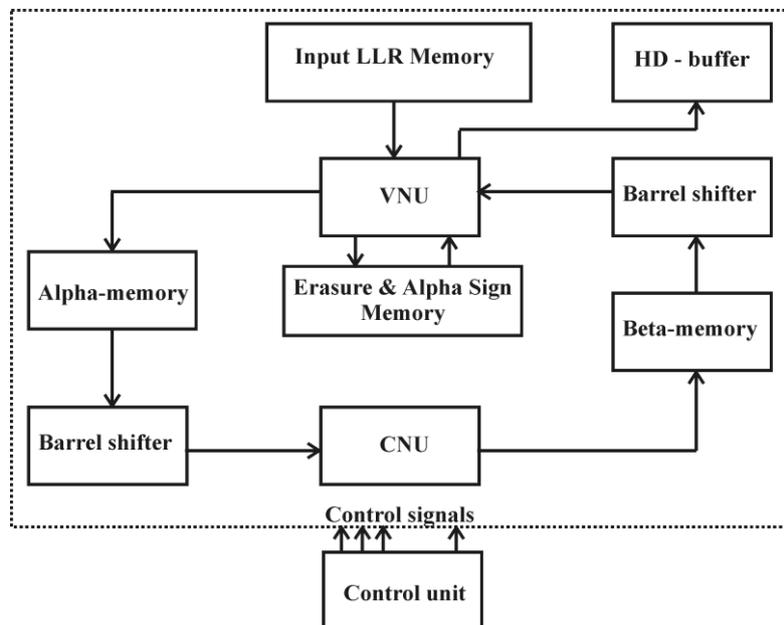


Figure 3-38: Flooded SCMS Architecture

The SCMS VNU list of ports for is summarized in Table 3-13.

Table 3-13: SCMS VNU Ports List

Port name	In/Out	Size [bits]	Function
beta_in	I	15	Compressed $\beta$ message input
beta_index_in	I	3	Index of the current processed $\beta$ message
gama_in	I	4	Input LLR
alfa_sign_in	I	1	Previous $\alpha$ message sign
erasure_in	I	1	Erasures bit in
start_proc_in	I	1	Control signal – start a new VNU operation
hard_dec_out	O	1	Hard decision output
alfa_out	O	4	Output $\alpha$ message
end_proc_out	O	1	Status signal
alfa_sign_out	O	1	Previous $\alpha$ message sign
erasure_out	O	1	Erasures bit in

### 3.5.3. Finite Alphabet Iterative Decoder (FAID)

FAID algorithm [Planjery10] uses a different variable node update than MS based algorithms. The  $\alpha$  message updates are performed using dedicated Look-Up Tables (LUT) for the input  $\beta$  messages and the input LLR value. The main FAID advantage relates to the utilization of 3 bits only for the quantization of both  $\alpha$  and  $\beta$  messages – 3 bits. Another advantage is that the VNU LUT can be optimized such as to improve FAID robustness to faulty hardware as suggested in Deliverable 3.2 [i-RISC/D3.2]. We note however that FAID decoders can only be utilized over Binary Symmetric Channels and only for regular LDPC codes with variable node degree 3 or 4. Regarding the CNU, the  $\beta$  messages are computed in the same way as in the MS algorithm. The FAID decoder data flow is the same as in the one of the MS decoder.

#### 3.5.3.1. Differences With Respect to MS Decoder

The FAID decoder has the same architecture as the MS decoder. Furthermore, the flooded FAID decoder has the same data flow as the MS decoder presented in Section 3.5.1. This leads to both decoders having identical control units. The differences between the FAID and MS decoders are:

- *Different VNUs* – the FAID VNU consists of 3 LUTs which are used to compute the  $\alpha$  messages; the input  $\beta$  messages are read from memory in a serial manner (one message per clock cycle); the  $\alpha$  messages are outputted also in a serial manner; two buffers are used: one to store the input  $\beta$  messages and one to store the newly computed  $\alpha$  messages; the  $\beta$  buffer provides inputs for the 3 LUTs; the inputs for the  $\alpha$  buffer are provided by the outputs of the 3 LUTs.
- *Differences due to different quantization* – the  $\alpha$  and  $\beta$  message memories have smaller memory word size ( $54 \times 3$  for  $\alpha$  message memory and  $54 \times 13$  for  $\beta$  message memory), while the two BS have smaller number of MUXes per level ( $54 \times 3$  for  $\alpha$  read BS and  $54 \times 13$  for  $\beta$  read BS).

### 3.5.4. Stochastic Decoder (SD)

In stochastic LDPC decoding, the channel probabilities are converted into randomly generated binary bit streams, called Bernoulli sequences [Gaines69] [Poppelbaum67] [Ribeiro67]. In a Bernoulli sequence, each comprising bit is generated independently of the other bits, and it is equal to 1 with a probability equal to the to be transformed channel probability. For instance, a stochastic stream of 10 bits with 5 comprising bits being equal to 1, encodes a channel probability of  $5/10 = 0.5$ . This number representation has two main advantages: (i) arithmetic computations can be done with low complexity hardware and (ii) has intrinsic high fault tolerant capability. This representation allows for the evaluation of complex arithmetic operations by using simple logic gates. For instance, we can use an AND gate to implement a multiplication. Moreover, by relying on an identically-weight representation, Stochastic Bitstreams (SB) which are the de facto data representation for Stochastic Computing (SC) are more error tolerant by construction, e.g., flipping one bit at position  $n$  of an SB with length  $L_s$  causes only an  $1/L_s$  difference irrespective of the  $n$  value while the same bit flip in a traditional binary representation creates a  $2^{n-1}$  difference.

#### 3.5.4.1. Decoder General Description

A general fully parallel implementation of an Edge Memory (EM) [Tehrani08] stochastic LDPC decoder is depicted in Figure 3-39.

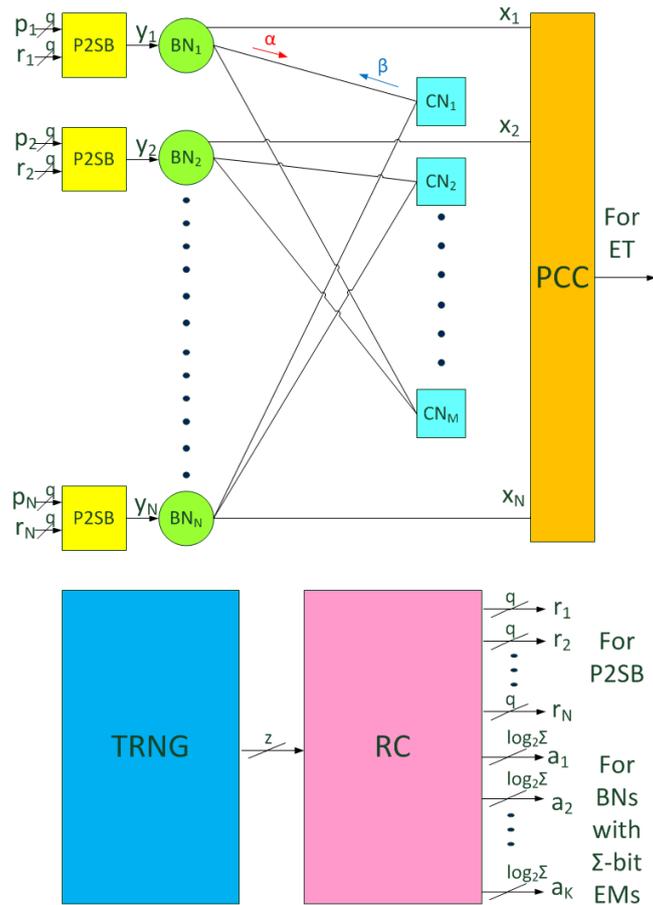


Figure 3-39: Architecture of a Fully Parallel Stochastic LDPC Decoder

Each channel probability  $p_n$  represented as a  $q$ -bit binary number is first converted to a Stochastic Bitstream (SB)  $y_n$  via a Probability to Stochastic Bitstream converter (P2SB) module. Each Bit Node (BN) module consists of: (i)  $N_{ccn}$  units of Variable Node  $VN_{n,m}$  module to generate  $\alpha_{n,m} \in \{0,1\}$ , message to be sent from  $BN_n$  to  $CN_m$  where  $N_{ccn}$  is the number of connected Check Nodes (CNs), and (ii) one *A Posteriori Counter Update and Hard Decision* (PCHD) module to generate hard value bit  $x_n$  for Early Termination (ET) and decoding output. Each  $CN_m$  in Figure 3-39 consists of  $N_{cbn}$  units of  $CN_{m,n}$  module where  $N_{cbn}$  is the number of connected BNs. The output hard value bits are then used by the Parity Check Circuit (PCC) to verify the parity constraints. Edges messages in form of SBs are exchanged iteratively between BNs and CNs, until either a codeword is found, or the maximum number of iterations is reached. The random numbers for P2SBs and  $VN_{n,m}$  modules are fed by the  $z$ -bit True Random Number Generator (TRNG) module via the Random Connector (RC) module. The TRNG and RC are needed by (i) the P2SB modules for generating random SBs with controllable probabilities of being “1” and (ii) the  $VN_{n,m}$  modules for randomly choosing a bit from EMs when the incoming messages are not in agreement.

The SD decoder VHDL code is produced by a specially developed in-house tool we designed for the automatic generation of LDPC decoder fully parallel IP cores starting from: (i) an  $H$  matrix, regular or irregular,  $H = (h_{m,n})_{\substack{m=1,\dots,M \\ n=1,\dots,N}}$ , which is a binary matrix with  $M$  rows and  $N$  columns, (ii) the number of bits  $q$  to represent the probabilities of channel messages, and (iii) the EM number of bits  $\Sigma$ .

### 3.5.4.2. Decoder Modules Description

*Probability to Stochastic Bitstream converter (P2SB)*: The implementation of the P2SB unit is based on a  $q$ -bit comparator, and its block scheme is graphically illustrated in Figure 3-40. The P2SB module generates an output bit which is equal to logic “1” with a probability given by the  $q$ -bit quantized channel probability.

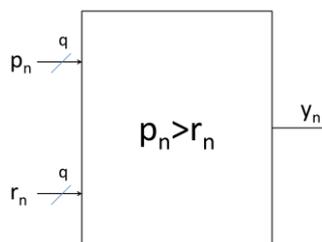


Figure 3-40: Probability to Stochastic Converter (P2SB)

$p_n$  is a vector of  $q$ -bit quantized probability of bit  $n$  to be equal to logic “1”. To randomly generate the bit  $y_n$  with a probability of being “1” equal to  $\frac{p_n}{2^q}$ , we need to give first a  $q$ -bit random number  $r_n$ , randomly generated by TRNG and RC modules, as an input to each P2SB module. The output bit of the P2SB module,  $y_n$ , is “1” when  $p_n > r_n$  and “0” otherwise.

The module ports description is summarized in Table 3-14.

Table 3-14: P2SB Module Ports Description

Port Name	In/Out	Size [bits]	Function
p	I	q	Probability

r	l	q	Random number
y	0	1	SB

*VN Module*: The block scheme of the variable node processing for variable node  $VN_n$  module is presented in Figure 3-41.

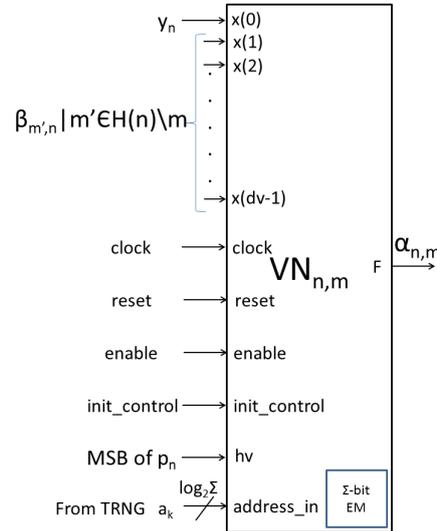


Figure 3-41: The  $VN_{n,m}$  Module

The initialization of  $VN_{n,m}$  module is done by giving logic “1” to the *init\_control* input. During the initialization, (i)  $\alpha_{n,m}$  is loaded with the Most Significant Bit (MSB) of the channel probability  $p_n$  (i.e.,  $\alpha_{n,m} = x_n$ ) and (ii) the EM is filled with a  $\Sigma$ -bit SB of  $y_n$  from the P2SB.

The main function of  $VN_{n,m}$  module is to check the agreement of messages (in form of SBs) from the channel and from all the CNs connected to  $VN_n$ , excluding the message from the check-node  $CN_m$  (i.e.,  $y_n = \beta_{m',n} \forall m' \in H(n) \setminus m$  where  $H(n) = \{m \mid h_{m,n} = 1\}$  is a set of check-nodes connected to the variable node  $VN_n$ , and  $\beta_{m,n} \in \{0,1\}$  is a message sent from  $CN_m$  to  $VN_n$ ). If the messages have an identical logic value, the  $VN_{n,m}$  module (i) passes the common value to its output F (i.e.,  $\alpha_{n,m} = y_n$ ) and (ii) stores the value in the EM inside the module by a single bit shifting operation. However, when the messages are not in agreement, the  $VN_{n,m}$  module selects randomly one bit from the EM addressed by the *address\_in* input and passes this single bit to its output F (i.e.,  $\alpha_{n,m} = EM(address\_in)$ ). The *address\_in* input is fed by the  $z$ -bit TRNG module via the RC module.

The ports description is presented in Table 3-15.

Table 3-15:  $VN_{n,m}$  Module Ports Description

Port Name	In/Out	Size [bits]	Function
$y_n$	l	1	SB from the channel with probability $p_n$
$\beta_{m',n}$	l	$d_v - 1$	Messages from all check nodes connected to $VN_{n,m}$ , excluding the message from check node $CN_m$

clock	l	1	Clock signal
reset	l	1	Reset active high
enable	l	1	"1" -> enable the $VN_{n,m}$ module
init_control	l	1	"1" -> initialize the $VN_{n,m}$ module
MSB of $p_n$	l	1	MSB of channel probability $p_n$
address_in	l	$\log_2 \Sigma$	Edge memory address for selecting 1-bit of data
$\alpha_{n,m}$	O	1	Message from $VN_{n,m}$ to be sent to check node $CN_m$

*TRNG Module:*

The TRNG is designed using Linear Hybrid Cellular Automata (LHCA) [Cattell95] [XilinxTRNG]. The connections from the RC inputs to its output are randomly predetermined at design time by our VHDL generator tool in such a way that each  $r_n$  for feeding the P2SB modules and each  $a_k$  for selecting random bit of EM has no common bit within itself (i.e.,  $r_n(0) \neq r_n(1) \neq r_n(2) \dots \neq r_n(q - 1)$  and  $a_n(0) \neq a_n(1) \neq a_n(2) \dots \neq a_n(\log_2 \Sigma - 1)$ ). In this way, correlations among the  $q$  random bits of  $r_n$  and the  $\log_2 \Sigma$  random bits of  $a_k$  are avoided.

*PCHD Module:*

Figure 3-42 depicts the PCHD<sub>n</sub> module block scheme.

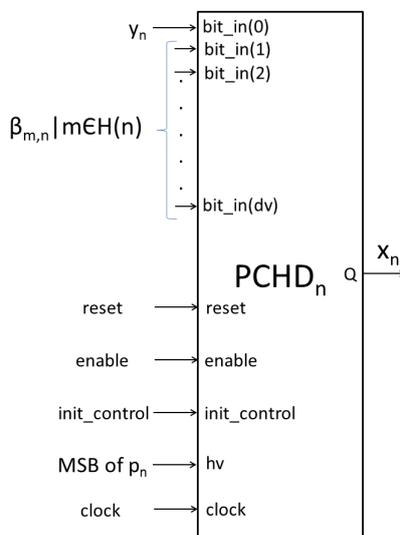


Figure 3-42: The PCHD Module

The module has a saturated signed counter called as  $\gamma_n$  for bit  $n$ . During initialization state when the *init\_control* input is "1", (i) the counter is filled with a value based on the MSB of its corresponding channel probability  $p_n$  (which is connected to its *hv* input) and (ii) the hard bit value is set to the MSB. The counter is initialized to +1 when the MSB is one (i.e.,  $x_n = 1$ ) and -1 otherwise. If the *bit\_in* input is "1", the counter is increased by 1; otherwise, it is decreased by 1. The *bit\_in(0)* is connected to the output bit of the corresponding P2SB  $y_n$  for counting the number of zero and one of SB messages from the channel. The other bits of the *bit\_in* input (i.e., *bit\_in(1)* to *bit\_in(dv)*) for a

regular LDPC code with variable node degree  $d_v$ ) are designed for counting messages from all connected CNs (i.e.,  $\beta_{m,n}, \forall m \in H(n)$ ) in the same manner. The hard value bit is determined by the number stored in the counter. If the stored value is positive, the hard value bit is set to one (i.e.,  $x_n = 1$ ), otherwise it is set to zero.

The module ports are described in Table 3-16.

Table 3-16: PCHD Module Ports Description

Ports Name	In/Out	Size [bits]	Function
$y_n$	I	1	SB from the channel with probability $p_n$
$\beta_{m,n}$	I	$d_v$	Messages from all check nodes connected to $VN_n$
reset	I	1	Reset active high
enable	I	1	"1" -> enable the PCHD module
init_control	I	1	"1" -> initialize
MSB of $p_n$	I	1	MSB of channel probability $p_n$
clock	I	1	Clock signal
$x_n$	O	1	Hard value

*CN and PCC Module:*

Each check node  $CN_{m,n}$  is implemented using a XOR gate as depicted in Figure 3-43.

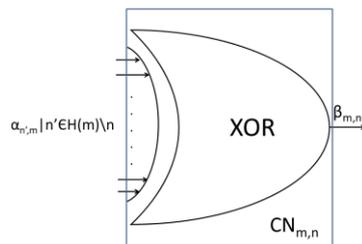


Figure 3-43: The  $CN_{m,n}$  Module

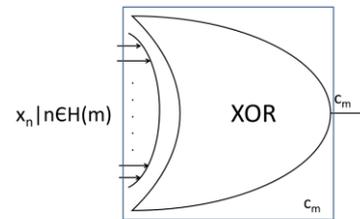


Figure 3-44: The Parity Check  $c_m$  Module

The inputs of the  $CN_{m,n}$  module are messages from all corresponding connected BNs except from  $BN_n$  (i.e.,  $\alpha_{n',m} \forall n' \in H(m) \setminus n$  where  $H(m) = \{n \mid h_{m,n} = 1\}$ , which is a set of VNs connected to the  $CN_{m,n}$ ). The inputs of the module are messages from BNs and its output  $\beta_{m,n}$  is passed back to connected BNs. This iterative message passing process stops either when all parity check constraints are satisfied determined by PCC or when the pre-determined maximum number of iterations is surpassed. The PCC module is built using  $N_{pcc}$  units of XORs where  $N_{pcc}$  is the number of parity check constraints. The inputs of the parity check  $m$ ,  $c_m$ , in Figure 3-44 are from PCHDs of BNs connected to  $CN_m$ . In our experiment, we use a regular (1296,648) LDPC code with these parameters:  $d_v = 3, d_c = 6, q = 6, \Sigma = 16, z = 210$ .

### 3.5.5. Gradient Descent Bit-Flipping (GDBF) and Probabilistic GDBF (PGDBF)

The Gradient Descent Bit Flipping (GDBF) and Probabilistic GDBF (PGDBF) are both hard decision, Bit Flipping (BF)-based, algorithms used in iterative LDPC decoders. Due to their simple computation units, bit flipping decoders significantly reduce the hardware resources needed for implementation. The shortcoming of this simplification is a significant performance loss compared to Belief Propagation (BP) decoders and their variants Min-Sum (MS), Normalized MS etc.

The GDBF algorithm is derived from gradient descent formulation and its theory consists in finding the best suitable bit (or group of bits) to be flipped in the Variable Nodes processing in order to maximize a predefined objective function. GDBF algorithm shows error correction performance far better than other BF variants and very close to normalized MS algorithm. The Probabilistic GDBF (PGDBF) is inspired from both GDBF algorithm and the Probabilistic BF algorithm. PGDBF has a better performance than the original GDBF. Instead of flipping all bits satisfying the gradient descent condition, PGDBF takes the flipping decision according to a probabilistic value.

#### 3.5.5.1. Decoder General Description

The top-level architecture of the decoder is presented in Figure 3-45. This architecture differs from the generic LDPC decoder architecture in several aspects: (i) it contains a global block that takes inputs from all VNUs (the Lambdas) and computes the maximum, and (ii) embeds binary random generators.

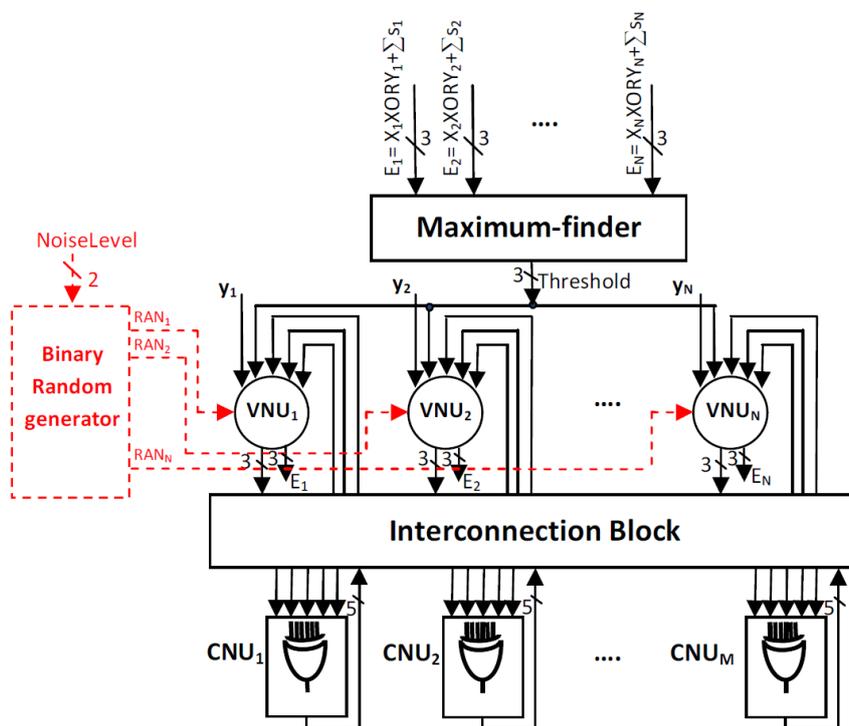


Figure 3-45: Global Architecture of PGDBF Compared to the Original GDBF

Several implementations of the PGDBF decoder for LDPC codes are proposed: a conventional implementation of the random generator through LFSR (Linear Feedback Shift Registers) as a first design, and a new approach [Le15] using binary sequences that are produced by the LDPC decoder, named IVRG, as second design.

The LFSR approach can be seen as a distributed Random Generator (RG) due to the fact that it is implemented inside every VN unit. The complexity of the interconnection network depends on the type and size of LDPC code used as well as on the chosen level of parallelism.

These aspects have been widely discussed in the literature [Darabiha08] and are not discussed here. Implementation of the RG will be discussed in Section 3.5.5.3.

An LDPC code is a linear block code defined by a sparse parity-check matrix  $H$  with size of  $(M, N)$ , where  $N > M$ . A codeword is a vector  $x = (x_1, x_2, \dots, x_N) \in \{0, 1\}^N$  which satisfies  $H \cdot x^T = 0$ , where  $x^T$  denotes the transposed vector of  $x$ .

We denote by  $y = (y_1, y_2, \dots, y_N) \in \{0, 1\}^N$  the output of a Binary Symmetric Channel (BSC), in which the bits of the transmitted codeword  $x$  have been flipped with crossover probability  $p_0$ . The decoders presented here are dedicated for BSC channel.

Let  $N(v(i))$  denote the set of CNs connected to the VN  $v(i)$ , with connection degree  $d_{v(i)}$ . Let also define  $N(c(j))$  as the set of VNs connected to the CN  $c(j)$ , with connection degree  $d_{c(j)}$ .

In Bit Flipping (BF) decoders, the value of variable nodes can change over the iterations, and we denote here by  $v^{(k)}(i)$  the value of the variable node at the  $k$ -th iteration. We correspondingly denote by  $c^{(k)}(j)$  the value of the parity checks at iteration  $k$ . The CN calculation in BF algorithms is defined by checking whether the parity check is satisfied or not. It can be written as:  $c^{(k)}(j) = XOR_{v(i) \in N(c(j))} v^{(k-1)}(i)$ , (XOR is the bit-wise Exclusive-OR operation). In the case of gradient descent BF algorithms, a function called *inversion function*, is defined for each VN unit, and used to evaluate that the value  $v^{(k)}(i)$  should be flipped or not.

The original GDBF is designed for the Additive White Gaussian Noise (AWGN) channel and the inversion function is defined as in (3-1). In GDBF [Wadayama10], only the VN having the smallest inversion function's value will be flipped, and sent for the next iteration.

$$\Lambda_{v(i)}^{(k)} = (1 - 2v(i)^{(k)})\gamma_i + \sum_{c(j) \in N(v(i))} (1 - 2c(j)^{(k)}) \quad (3-1)$$

where  $\gamma_i$  is the received value from AWGN channel. In [Rasheed14], the authors proposed an inversion function to apply GDBF algorithm for the Binary Symmetric Channel (BSC). The inversion function for BSC is modified, and the bits having the maximum value of  $\Delta_{v(i)}^{(k)}$  in (3-2) are flipped.

$$\Delta_{v(i)}^{(k)} = v(i)^{(k)} XOR_{\gamma_i} + \sum_{c(j) \in N(v(i))} (c(j)^{(k)}) \quad (3-2)$$

In [Rasheed14], the inversion function's value is an integer and varies from 0 to  $d_{v(i)} + 1$ . Due to the integer representation of inversion function, many bits can be flipped in the same iteration. This fact may induce a negative impact on the convergence of the algorithm as the analysis of [Rasheed14] shows. To avoid this effect, the PGDBF has been proposed with the idea that, instead of flipping all the bits with maximum inversion function value, only a random fraction of those bits are flipped. The random fraction is fixed by a pre-defined probability  $p_i^{(k)}$ , which could be different for each VN and each iteration. The PGDBF algorithm is explained in Table 3-17.

Table 3-17: Probabilistic Gradient Descent Bit-Flipping Algorithm

<p><b>Initialization</b> <math>v_i^{(init)} \leftarrow y_i, i \in [1, N]</math></p> <p><b>syndrome</b> = <math>H \cdot v^{(init)}</math></p> <p><b>While</b> <b>syndrome</b> <math>\neq \mathbf{0}</math> <b>and</b> <math>k &lt; I_{max}</math> <b>do</b></p> <p><math>\forall i \in [1, N]</math></p> $\Delta_{v(i)}^{(k)} = v(i)^{(k)} \text{ XOR } y_i + \sum_{c(j) \in N(v(i))} \text{ XOR }_{v_u \in N(c(j))} v_u^{(k)}$ <p><b>Threshold</b> = <math>\max(\Delta_{v(i)}^{(k)})</math></p> <p><b>if</b> <math>\Delta_{v(i)}^{(k)} = \text{Threshold}</math> <b>then</b></p> <p>    <b>if</b> <math>R_i^{(k)} = \mathbf{1}</math> <b>then</b></p> <p>        <math>x_i^{(k)} = \text{NOT}(v(i)^{(k)}); \{p(R_i^{(k)} = \mathbf{1}) = p_i^{(k)}\}</math></p> <p>    <b>end if</b></p> <p><b>end if</b></p> <p><b>syndrome</b> = <math>H \cdot v^{(k)}</math></p> <p><math>k = k + 1</math></p> <p><b>end While</b></p> <p><b>Outputs:</b> <math>v^{(k)}</math></p>
---

### 3.5.5.2. Decoder Modules Description

For the hardware implementation, it can be seen that the non-probabilistic GDBF and PGDBF have the same structure for the Check Node (CN) units and also for the maximum-finder module.

The maximum-finder (which deliver Threshold signal in Figure 3-46) is in charge of finding the maximum value of inversion functions. In this work, we follow the conventional method, which uses the binary comparator tree to implement the maximum-finder. Figure 3-46 shows the VN units of PGDBF. In this architecture, an extra block, which generates sequences of random bits, denoted as  $R_i^{(k)}$  in algorithm 1 (signal "ran" in Figure 3-46), are needed. Those blocks are the main difference between PGDBF and non-probabilistic GDBF and are required in order to improve the error correction performance.

In [Rasheed14], it is also shown that the optimum probability mass function for the random binary sequence is  $p' = 0.9$ . Two solutions for the analysis and design of random generators having a fixed value of  $p'$  are presented in the next section.

Table 3-18 presents the ports description of VNU module.

Table 3-18: VNU Module Ports Description

Port Name	In/Out	Size [bits]	Function
Ena	I	1	Enable VNU
Clk	I	1	Clock signal
Data_load	I	1	Data from channel will be loaded to internal register if data_load='0'
Receive_data	I	1	Data from channel

C2V	I	3	inputs from CNU
threshold	I	3	Threshold from maximum finder module
ran_in	I	1	Random number from random generator
sum_out	O	3	Energy function output
VNU_out	O	1	Output bit flipped or not

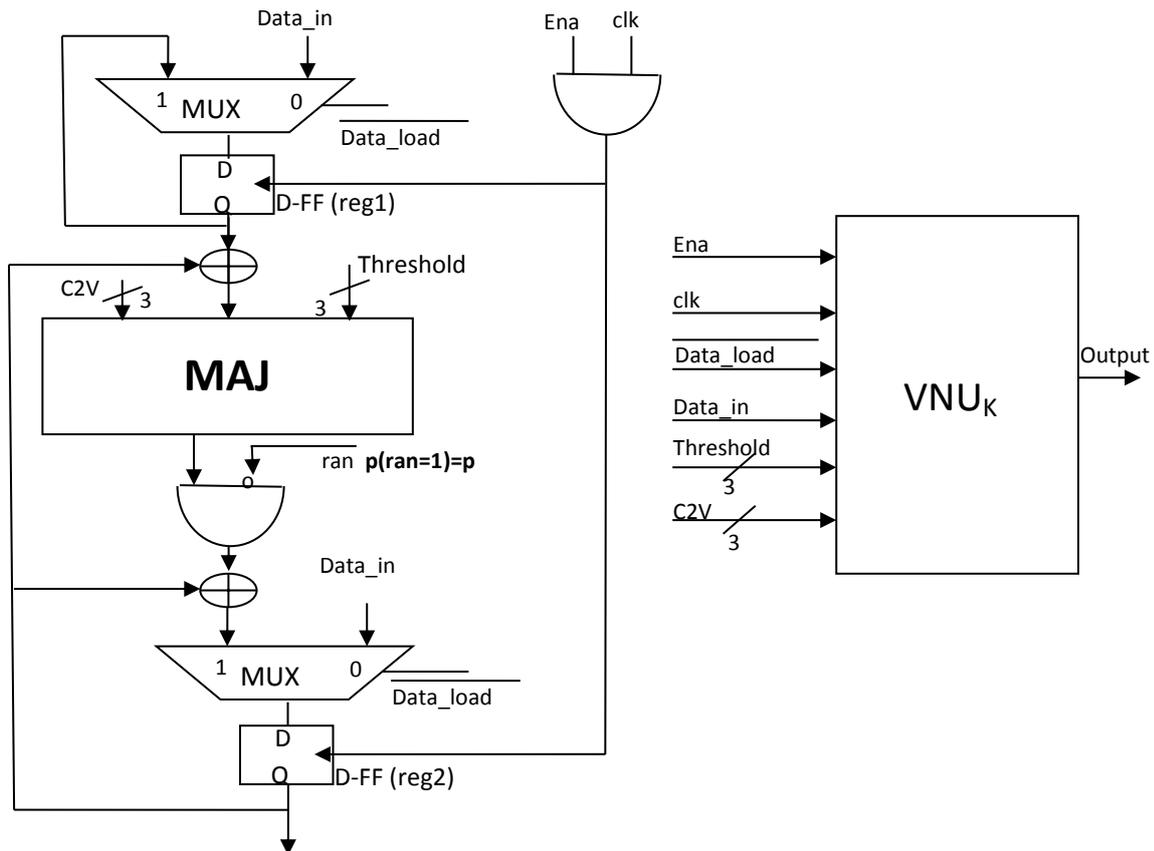


Figure 3-46: Architecture of VNU in PGDBF

### 3.5.5.3. Analysis and Design of Random Binary Generators

#### a) LFSR random generator

The first design proposed is based on Linear Feedback Shift Registers (LFSR), with controlled probability of getting zero or ones, we consider for inclusion in each instantiated VNU. We make use of LFSR with maximum length feedback polynomial to generate an integer number, and the generated number is compared with a threshold to decide if the new bit in the random sequence should be a 0 (higher than threshold) or 1 (lower than threshold). Two aspects are of interest when designing a variable threshold random binary sequence generator: first the period of the random sequence, second the granularity with which the threshold can be programmed.

b) *Intrinsic-Value Random Generator (IVRG)*

A new approach is presented in this section. This approach reduces the cost of generating random binary sequences by means of substituting all local Random Binary Sequence Generators (one per VNU) by a global one. We name this new method Intrinsic-Value Random Generator (IVRG), which makes use of the value of the CNs inside decoder as inputs. In an LDPC iterative decoder, the values of the CNs depend both on the BSC crossover probability  $p_0$ , the degree of check nodes  $d_c$  and the iteration number. Typically, the number of CN which are unsatisfied (value = '1') is large during the first iterations, while it becomes smaller as the iteration number increases. We denote by  $p(c_j^{(k)} = 1) = F(p_0, k, d_c)$  the probability that a CN is unsatisfied, as a function of the three mentioned parameters.

In this work, we will use only the CN values produced at the first iteration  $k = 0$  in order to generate sequences of random bits.

At the first iteration, the probability mass function is given by:

$$p(c_j^{(0)} = 1) = F(p_0, 0, d_c) = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{d_c}.$$

The results in [Rasheed14] have showed that the optimal value of  $p(c_j^{(0)} = 1)$  for the Tanner code (155, 64) is around 0.9, and the range of crossover probability is from  $10^{-3}$  to around  $3 \cdot 10^{-2}$ .

In order to control the random generator probability  $p_0$ , we propose to use a function  $G$  of the CN values  $G(c_{j_1}^{(0)}, c_{j_2}^{(0)}, \dots), j_1, j_2 \in [1, M]$  that controls the desired probability  $p'$ . We briefly describe the function  $G$  as follows:

Let  $c_{j_1}$  and  $c_{j_2}$  be two binary random variables with  $p(c_{j_1} = 1) = p(c_{j_2} = 1) = p$ , it can be proved that  $p(c_{j_1} \text{ OR } c_{j_2} = 1) = 2p + p^2 > p$  and  $p(c_{j_1} \text{ AND } c_{j_2} = 1) = p^2 < p$ .

More specifically:

$$p(c_{j_1} = 1) = p(c_{j_2} = 1) = p = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{d_c}, \quad p(c_{j_1} \text{ XOR } c_{j_2} = 1) = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{2d_c}.$$

Using these transformations of probability, and a function  $G$  implemented as described in Figure 3-47, we can transform the CN output sequence into a longer binary pseudo-random sequence with a desired probability  $p'$ .

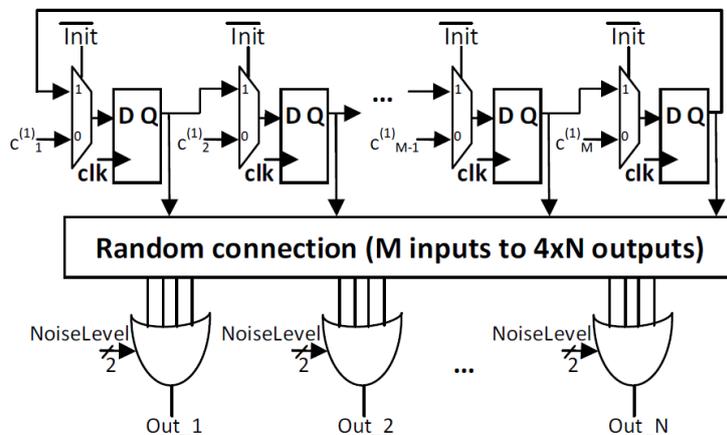


Figure 3-47: Architecture of Intrinsic-Valued Random Generator

Let define  $p' = C \cdot p_0$  where  $p_0$  is the probability of the first iteration CN value equal to '1'. The multiplication coefficient  $C$  is a function of  $p_0$  with the condition is  $p' = 0.1$  as in the Figure 3-48. This figure shows the values of multiplication coefficients  $C$  and its approximation using 4 levels, which represent 4 levels of noise in BSC channel (4 levels of crossover probability).

The Table 3-19 presents the port description of the IVRG module.

Table 3-19: IVRG Module Ports Description

Port Name	In/Out	Size [bits]	Function
Ena	I	1	Enable VNU
Clk	I	1	Clock signal
Init	I	1	Initial step for loading data from CNU
Data_in	I	93	Input from CNU
Crossover	I	2	Crossover value: level of noise
Ran_out	O	155	Random value: 0 or 1 for each VNU

The larger number of levels the more precision in output probability it offers. The BSC noise level is represented as in 2-bit Noise-Level in the final architecture. The CN's first iteration values are stored in the chain of Flip-Flops at the initialization phase. They will be rotated at each iteration. They are also assigned to be the inputs of the selectable-OR gates (Figure 3-47) through the random connection network.

As an example, if the BSC crossover probability is 0.01, from Figure 3-47, multiplication coefficient  $C$  is 2, the output of OR gates will be bitwise-OR results of 2 from 4 inputs. As expected, the desired probability is around 0.1 and the results, which is depicted as a scatter plot in Figure 3-48, which indicates that the output sequences have the probability to be 1 very close to the expected value. On this figure, the blue curve is an approximation that has been used in the FPGA implementation.

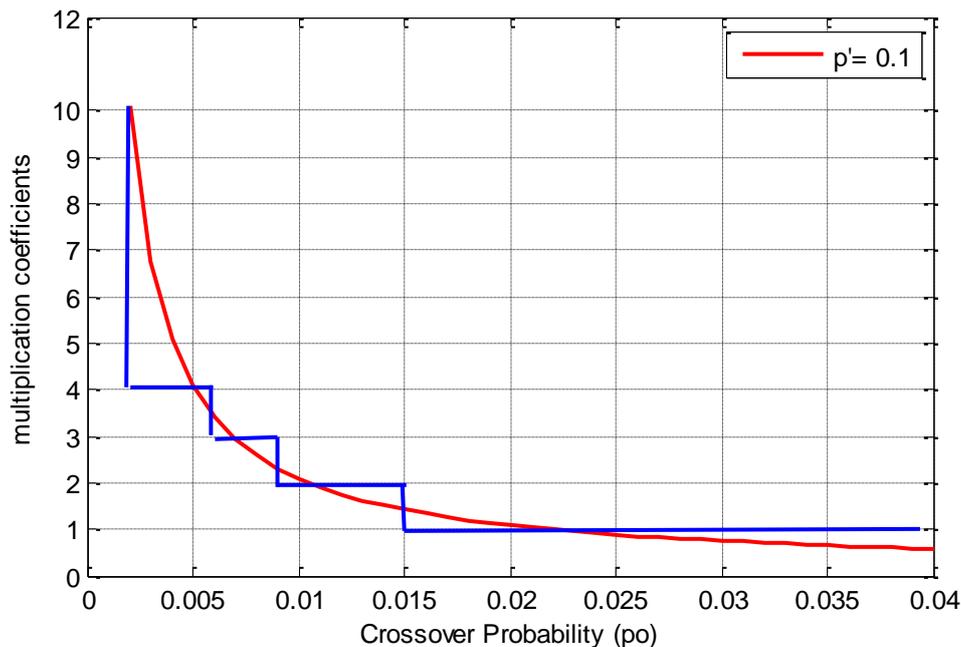


Figure 3-48: The Multiplication Coefficient  $K$  as a Function of the BSC Crossover

We have first made the synthesis of the different solutions for the case of a small LDPC code that has been proposed in the literature [**Tanner01**]: a regular, quasi-cyclic LDPC code with regular connection degrees  $d_v = 3$  and  $d_c = 5$ , with codeword length  $N = 155$ , called the Tanner code.

Table 3-20 shows the hardware resources needed to implement the two different PGDBF structures. As benchmarks, the resources for the non-probabilistic GDBF and 6 bits Min-Sum decoder are shown as well. The maximum frequency and the estimated throughput have been obtained from an implementation using FPGA Xilinx Virtex 6 of 40nm technology, after place and route.

We can notice that the IVRG-PGDBF needs an additional 92 1-bit registers (9.7% overhead) compared to the non-probabilistic while the LFSR-PGDBF needs 8215 1-bit registers overhead (868.4% overhead). This large overhead emphasizes the advantage of IVRG over LFSR in terms of implementation.

Comparing the Slice LUTs required, the IVRGPDBF requires 261 more slices than the non-probabilistic (12.1%) and this number for LFSR-PGDBF is 1394 (64.8%).

The extra complexity brought by the RG implementation has moreover a negligible impact on the obtained throughput (less than 2%) in all PGDBF implementations. We can also see that the offset min-sum decoder is far more complex than the BF type decoders, and cannot compete in terms of decoding speed.

Table 3-20: Hardware and Throughput Estimation for PGDBF with Different RG Implementations and for Offset Min-Sum

	1-bit Register	Slice LUTs	Fmax (Mhz)	Throughput (Mbps)
GDBF	946	2151	132.721	4114.3
IVRG-based PGDBF	1038	2412	132.721	4114.3
LFSR-based PGDBF	9161	3545	135.56	4202.36
Offset Min-Sum (6bits)	13694	15350	237.185	197.5

### 3.5.6. Gallager-B with Extended Alphabet (GB)

The Gallager B decoder was described by Robert Gallager in his doctoral dissertation at the MIT [**Gallager62**]. This is a hard decision decoder that manipulates bits, not probabilities. This decoder has several advantages insofar as it requires little resources and does not use a MAC (Multiply and Accumulate) unit. We implemented a parallel Gallager-B LDPC decoder with extended alphabet, which is a hard decision Message Passing (MP) decoder. The Gallager B decoder operates on the bit flipping principle, i.e., it changes bits in order to validate all the parity check. The block diagram of the decoder core architecture is depicted in Figure 3-50.

In the following we introduce the notation we utilise later on in the section:

- $v_n$  : variable node (bit node)  $n$
- $c_m$  : check node  $m$
- $\hat{x}$  : corrected information
- $y$  : vector containing the data
- $\alpha_{m,n}$  : message sent by the bit node  $n$  to check node  $m$
- $\beta_{m,n}$  : message from check node  $m$  to bit node  $n$

- $\gamma_n$  : a priori information vector
- $\tilde{\gamma}_n$  : a posteriori information
- $s_n$  : subtotal whose formula is given by the following
- $s_{m,n}$  : subtotal whose formula is given by the following
- $H(c_m)$  : all neighbour variable nodes of check node  $c_m$
- $H(v_n)$  : all neighbour check nodes of variable node  $v_n$
- $H(c_m) \setminus \{v_n\}$ :  $H(c_m)$  except for  $v_n$
- $H(v_n) \setminus \{c_m\}$ :  $H(v_n)$  except for  $c_m$
- $t$  : qualified majority threshold value

The extended alphabet is used in the implementation of this Gallager-B decoder, which means we use  $\{-1, x, +1\}$  alphabet. When there is no qualified majority of votes, meaning  $|s_{m,n}| < t$ , a variable-to-check message ( $\alpha_{m,n}$ ) will take on the value 0 (instead of replacing the initial  $\gamma_n$  value) [Savin14]. To implement this feature, using two bits message was preferred to the use of a 3 states implementation. Therefore, each message is composed of two bits whose MSB codes the sign while the LSB codes the presence of mistaken information as indicated in Figure 3-49.

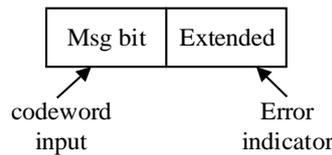


Figure 3-49: Input Data Format

Thus the alphabet used in the design is summarized in Table 3-21:

Table 3-21: Alphabet Format

Alphabet	Msg format
+1	01
-1	11
x	x0

The extended alphabet version of Gallager-B decoding algorithm is presented in Table 3-22.

Table 3-22: Gallager-B with Extended Alphabet Decoding Algorithm

<b>Input:</b> $y = (y_1, \dots, y_N) \in \{0,1\}^N$
<b>Output:</b> $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N) \in \{0,1\}^N$
<b>Initialization</b>
<b>for all</b> $\{v_n\}_{n=1,\dots,N}$ <b>do</b> $\gamma_n = 1 - 2y_n$
<b>for all</b> $\{v_n\}_{n=1,\dots,N}$ <b>and</b> $c_m \in H(v_n)$ <b>do</b> $\alpha_{m,n} = \gamma_n$
<b>Iteration Loop</b>
<b>for all</b> $\{c_m\}_{m=1,\dots,M}$ <b>and</b> $v_n \in H(c_m)$ <b>do</b>
$\beta_{m,n} = \begin{cases} x, & \text{if at least one message is } x \\ \prod_{v_{n'} \in H(c_m) \setminus \{v_n\}} \alpha_{m,n'}, & \text{otherwise} \end{cases}$
<b>for all</b> $\{v_n\}_{n=1,\dots,N}$ <b>and</b> $c_m \in H(v_n)$ <b>do</b>

$$s_{m,n} = \gamma_n + \sum_{\substack{c_m \in H(v_n) \setminus \{c_m\} \\ \beta_{m',n} \neq x}} \beta_{m',n}$$

$$\alpha_{m,n} = \begin{cases} x, & \text{if } s_{m,n} = x \\ \gamma_n, & \text{if } |s_{m,n}| < t \\ [\text{sign}(s_{m,n}), 1], & \text{otherwise} \end{cases}$$

**for all**  $\{v_n\}_{n=1,\dots,N}$  **do**

$$s_n = \gamma_n + \sum_{\substack{c_m \in H(v_n) \\ \beta_{m,n} \neq x}} \beta_{m,n}$$

$$\tilde{\gamma}_n = \begin{cases} \gamma_n, & \text{if } s_n = 0 \\ [\text{sign}(s_n), 1], & \text{otherwise} \end{cases}$$

**for all**  $\{v_n\}_{n=1,\dots,N}$  **do**

$$\hat{x} = (1 - \tilde{\gamma}_n)/2$$

**if**  $\hat{x}$  **is codeword** **then** **exit the iteration loop**

**End Iteration Loop**

Figure 3-50 captures the block diagram of the decoder core. As suggested in the figure, the decoder core is composed of bit nodes block, check nodes block connected by the interconnection network  $\alpha_{m,n}$  and  $\beta_{m,n}$ . Both of the bit nodes and check nodes are parallel structured, therefore the size of datain and dataout is  $2 \times \text{NB\_COLUMN}$ .

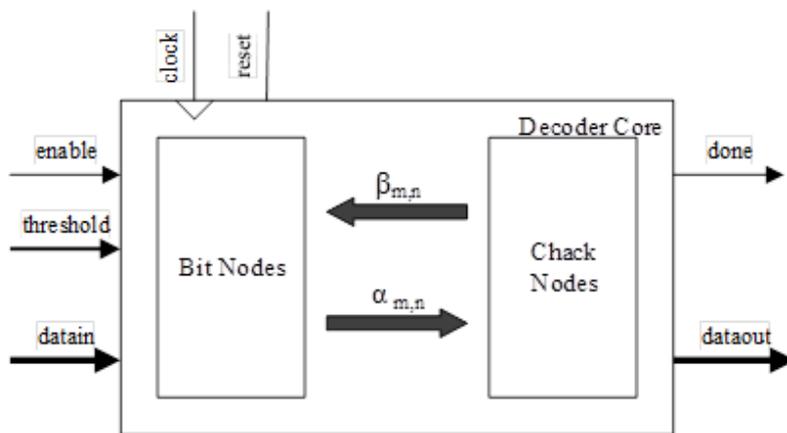


Figure 3-50: Block Diagram Representation of the Gallager-B Decoder

Table 3-23 summarizes the functionality and usage of the ports.

Table 3-23: LDPC Decoder Ports Description

Port	I/O	Size	Description
enable	in	1	Control signal to enable the iteration process when data is ready. High level active
threshold	in	3	The threshold value
datain	in	2× NB_COLUMN	Codeword parallel input with the extended alphabet (leave to “1”)
done	out	1	When active indicates all parity check equations are verified
dataout	out	2× NB_COLUMN	The processed code word with the extended alphabet (0 for correct, 1 for error)

Table 3-24 captures all the important parameters involved in the design of the Gallager-B LDPC decoder.

Table 3-24: LDPC Decoder Design Parameters

Parameter	Description
NB_ROW	Number of rows in the H matrix, also represents the number of check nodes
NB_COLUMN	Number of columns in the H matrix, also represents the number of bit nodes
NB_ONE	Number of “1”s in the H matrix
N	Number of “1”s in a particular column, also represents the number of neighbour check nodes of a particular bit node
M	Number of “1”s in a particular row, also represents the number of neighbour bit nodes of a particular check node
SIZE_SUM	The maximum size of a bit node sum computation could require (size of $s_n$ and $s_{m,n}$ )

### 3.5.6.1. Bit Node Implementation

The block diagram of a bit node is depicted in Figure 3-51, which is one of the NB\_COLUMN bit nodes in the bit nodes block stated above. Bit node calculates the bit-to-check messages  $\alpha_{m,n}$  using the a priori gamma message  $\gamma_n$  and the extrinsic messages  $\beta_{m',n}$  and also calculates the a posteriori

gamma message  $\tilde{\gamma}_n$  using the a priori gamma message  $\gamma_n$  and all messages  $\beta_{m,n}$  from its N neighbour check nodes. Therefore input gamma priori needs 2 bits while the check message in needs  $2 \times N$  bits.

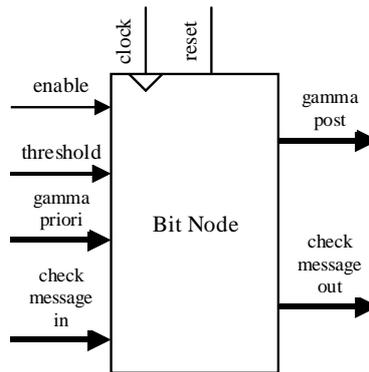


Figure 3-51: Bit Node Block Diagram

A summary of all ports information is shown in Table 3-25.

Table 3-25: Bit Node Port Description

Port	I/O	Size	Description
enable	in	1	Control signal to enable the iteration process when data is ready. High level active
threshold	in	3	The threshold value
gamma_priori	in	2	Codeword parallel input with the extended alphabet (leave to "1")
check_message_in	in	$2 \times N$	Message received from check node m, where N is the number of neighbour check nodes
gamma_post	out	2	Gamma posterior output
check_message_out	out	$2 \times N$	Message sent from bit node to check node, where N is the number of neighbour check nodes

We will now study the implementation of the bit node. The main difficulty was to consider the extended alphabet bit. No parity equations will be validated if they contain information "x". To solve this issue, each computing unit (including addition and subtraction blocks) processes only the message bit with the extended bit being "1". To make the implementation simpler, it first calculates the sum  $s_n$  because it is then possible to calculate with only one additional subtracting the  $\beta_{m,n}$  message to obtain the sum  $s_{m,n}$ . Bit Node implementation details are presented in Figure 3-52.

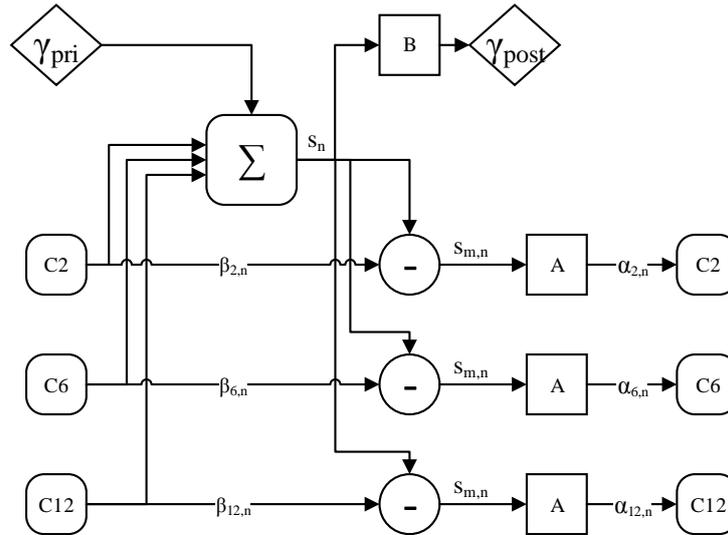


Figure 3-52: Bit Node Architecture

The block A representation is captured in Figure 3-53. A block calculates the  $\alpha_{m,n}$  message to be sent to the check node. First we exclude the problem of bit error by placing a condition on the input, if the value is 0, it returns erroneous output “x”, in which case the sign bit is not important and its value is by default set to 0 (positive). Then we compare the absolute value of  $s_n$  with the threshold value  $t$ , if  $s_n$  is smaller than the threshold, it returns  $\gamma_n$ , otherwise a concatenation of the sign of  $s_{m,n}$  and an error bit set to “1” is returned.

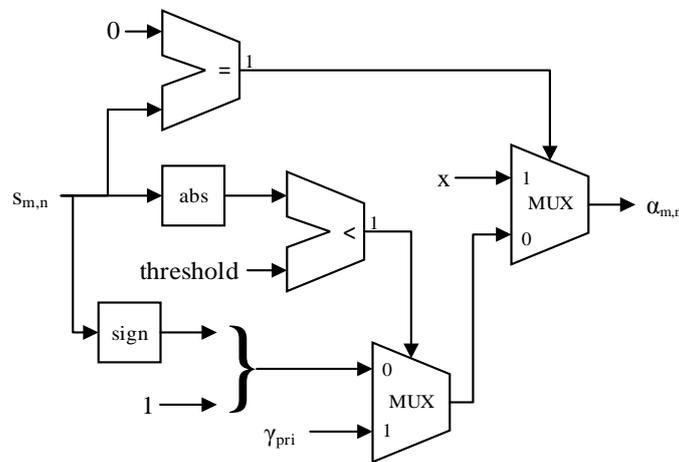


Figure 3-53: Block A Implementation

The architecture of B Block is depicted in Figure 3-54 . For B block only one multiplexer is used. We first compare the  $s_n$  with the zero vector. Then we got the sign of the sum in the same manner as above by converting the sum of the "sign-magnitude" format. If the result is zero it returns  $\gamma_n$ , otherwise it returns a concatenation of the sign and a valid error bit.

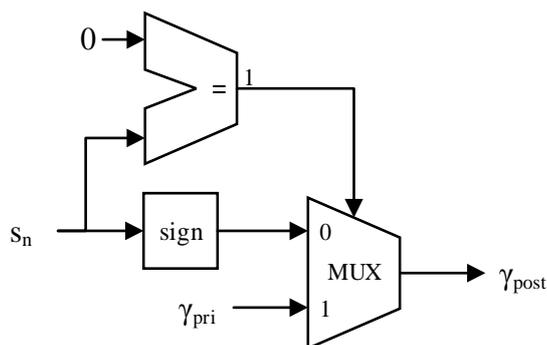


Figure 3-54: Block B Implementation

### 3.5.6.2. Check Node Implementation

The block diagram of one check node is depicted in Figure 3-55 (the total number of check-nodes is equal to NB\_ROW). Each check node calculates the check-to-bit message  $\beta_{m,n}$  from the bit-to-check messages  $\alpha_{m,n}$  received from its M neighbour bit nodes. Therefore both of the bit message in and out are of size  $2 \times M$ .

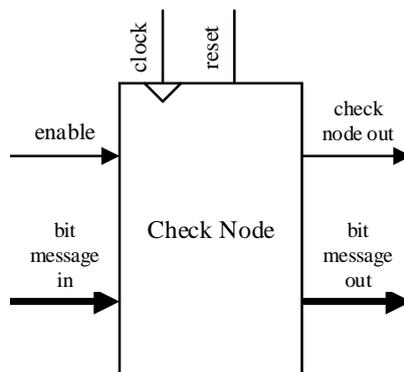


Figure 3-55: Check Node Block Diagram

The port description is captured in .

Table 3-26.

Table 3-26: Check Node Ports Description

Port	I/O	Size	Description
enable	in	1	Control signal to enable the iteration process when data is ready. High level active
bit_message_in	in	$2 \times M$	Message received from bit node n, where M is the number of neighbour bit nodes
check_node_out	out	1	When active indicates all equation are verified, high level active
bit_message_out	out	$2 \times M$	Message sent from check node to bit node, where M is the number of neighbour bit nodes

Check node is a series of multiplications between bit nodes messages. The multiplication is equivalent to an XOR for the chosen alphabet. However, we must take into account the extended alphabet. To do so we add an AND. This way all error bits must be “1” (no errors) to return a “1”, otherwise the error bit of check-to-variable message will be “0”, hence the returned message will be “x”. A further part was added in practice to allow checking if all parity equations were verified, then without unnecessary calculation, if the check\_node\_out returns “0” which means all equations were checked to be satisfied, the iteration process will be terminated and “1” otherwise. The implementation detail of bit node is presented in Figure 3-56.

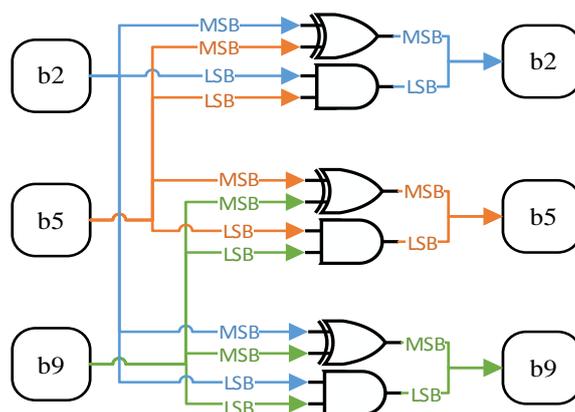


Figure 3-56: Check Node Architecture

### 3.6. Performance Evaluation and Comparison

This section reports on the performance evaluation of the implemented decoders, for both Voltage Scaling and Simulated Fault Injection scenarios.

#### 3.6.1. Voltage Scaling

To evaluate LDPC decoders under voltage scaling, we physically implemented the following 10 decoders on Xilinx Virtex-7 FPGA: XC7VX485TFFG1761-2 inside the Xilinx VC707 board: SD, GB, GDBF, PGDBF, MS, SCMS, FAID, MSnoET, SCMSnoET, and FAIDnoET. Implementation details for the former six decoders have been provided in Section 3.5. MSnoET, SCMSnoET, and FAIDnoET decoders are variants of the MS, SCMS, and FAID decoders, respectively, without the Early Termination (ET) circuit. In [i-RISC/D3.1] it has been demonstrated that an error-free (i.e., not subject to timing errors) ET circuit may greatly improve the decoder’s error correction capability on the error-floor region. Here, we investigate the impact of the ET circuit on the error correction performance, when it is submitted to the same aggression profile than the rest of the decoder. Using similar arguments to those in [i-RISC/D3.1, Section 1.7.2], it is expected that the decoding error probability on the error-floor region will be lower-bounded by the error probability of the ET circuit output.

To get inside on the complexity of the implemented designs we summarize the FPGA resources utilization and maximum clock frequency for all the evaluated decoders in Table 3-27. One can observe in the table that all designs, except GDBF and PGDBF, can operate at a maximum clock

frequency around 200MHz and that MS, SCMS, and FAID, utilize BRAM blocks (embedded RAM FPGA resources), while the others are solely realized by means of Flip-Flops and LUTs.

Table 3-27: FPGA Resources Utilization and Maximum Clock Frequency of Implemented Hardware Decoders

Decoders	FFs	LUTs	BRAMs	Max. Clock Freq. [MHz]
SD	81015	47260	0	194.286
GB	23577	28152	0	185.791
GDBF	7365	14283	0	132.520
PGDBF	8016	16061	0	123.093
MS	16608	14147	20	198.373
MSnoET	15296	13354	20	196.299
SCMS	16948	15855	21	195.442
SCMSnoET	15705	13819	22	196.299
FAID	14814	15176	15	195.525
FAIDnoET	13569	12848	16	195.221

All decoders are evaluated in terms of: Bit Error Rate (BER), Frame Error Rate (FER), average number of decoding iterations, throughput, and energy/bit. The throughput ( $T$ ) and energy/bit ( $E$ ) are computed by:

$$T = \frac{N}{I \cdot n_{cc} \cdot t_{clk}} \quad \text{and} \quad E = \frac{I \cdot n_{cc} \cdot t_{clk} \cdot P}{N},$$

where  $N$  is the code-length,  $I$  is the average number of decoding iterations,  $n_{cc}$  is the number of clock-cycles per iteration,  $t_{clk}$  is the clock period, and  $P$  is the consumed power.

We note that the above throughput and energy formulae are derived by considering all coded bits, independently of whether or not they have been successfully decoded by the decoder. To account for the decoder performance, so that to have a fair comparison between different decoders and different aggression profiles, the throughput and energy/bit are further normalized to either BER or FER, as follows:

$$\begin{aligned} T_{BER} &= T \cdot (1 - \text{BER}), & T_{FER} &= T \cdot (1 - \text{FER}) \\ E_{BER} &= E / (1 - \text{BER}), & E_{FER} &= E / (1 - \text{FER}) \end{aligned}$$

Normalizing to BER amounts to computing the throughput and energy by only considering the successfully decoded bits. However, in case a frame is in error – which can be detected by syndrome computation or by using a Cyclic Redundancy Check (CRC) code – one cannot know which bits are in error and which are correct. In such a case, communication systems implementing retransmission protocols usually discard the entire frame and ask for retransmission. Hence, normalizing to FER amounts to computing the throughput and energy by considering the bits within successfully decoded frames only and it is better tailored to the communication system prospective.

In order to reduce the number of simulations, all the decoders have been evaluated for the Binary Symmetric Channel (BSC) only. It is worth mentioning that GB, GDBF, PGDBF, and FAID are *hard-decision* decoders, i.e., they operate on data that take only on 0 and 1 values. Using them over a

channel other than BSC would require the channel output to be quantized to 1 bit, which would actually turn the channel into a BSC.

SD, MS, and SCMS decoders are *soft-decision* decoders, *i.e.*, their inputs may take on a wider range of values, and thus are quantized on a higher number of bits. The MS and SCMS decoder inputs are 4-bit signed integers, representing quantized Log-Likelihood Ratio (LLR) values. The binary output of the BSC channel is fed to either  $-4$  or  $+4$  (output 0 is fed to  $+4$ , while output 1 is fed to  $-4$ ). The impact of the *channel value* (in this case 4) on the MS decoder performance has been investigated in **[i-RISC/D3.1]** (the SCMS decoder is insensitive to the choice of the channel value). Here, we use a channel value of 4, which yields good performance in both the waterfall and the error-floor region.

The inputs of the SD decoder are 6-bit unsigned integers, representing quantized probability values. The binary output of the BSC channel is fed to either 8 or 55 (output 0 is fed to  $+8$ , while output 1 is fed to  $+55$ ), corresponding to the noise-dependent scaling method introduced in **[Tehrani06]**.

In the sequel we present the result of our voltage scaling experiment experiments in a decoder wise fashion. For each decoder we present it figure of merit in terms of the previously mentioned metrics and comment on the results.

#### 3.6.1.1. Stochastic Decoder

The results of voltage scaling experiments of the stochastic decoder under the BSC channel are presented in Figure 3-57 to Figure 3-63. The maximum number of iterations is set to 1000. One can observe in these figures that:

- The decoder starts experiencing the impact of voltage scaling at 0.82V and it goes to an almost flat FER (*i.e.*, FER = 1) at 0.78V after the decoder tries its best, decoding to its maximum number of iterations. This suggests that it takes 0.18V starting from nominal  $V_{dd}$  until the decoder reaches its critical point when it starts experiencing timing errors, and 0.04V from its critical point until it is completely knocked down.
- The average number of decoding iterations is increasing with decreasing supply voltage, which is consistent with the BER/FER degradation. However, for supply voltage values below 0.79V the number of decoding iterations is getting below the maximum, while the FER is almost flat equal to 1. This indicates that the ET circuit is affected by timing errors, making the decoder stop while it shouldn't.
- Decreasing the supply voltage from 1V to 0.82V enables a reduction of 50% in terms of energy per bit (normalized to BER) with almost the same decoding performance without any degradation of throughput.

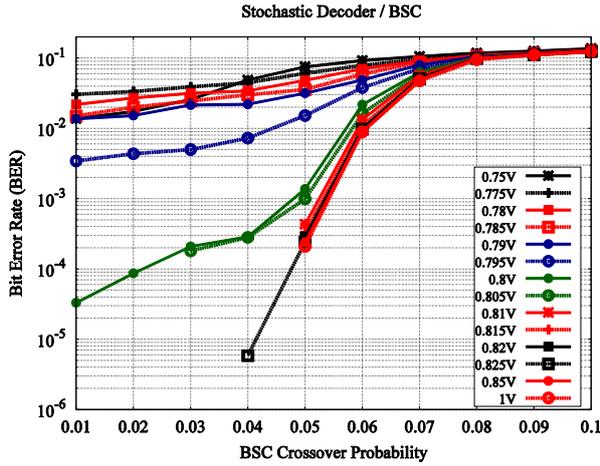


Figure 3-57: BER of Stochastic Decoder under BSC

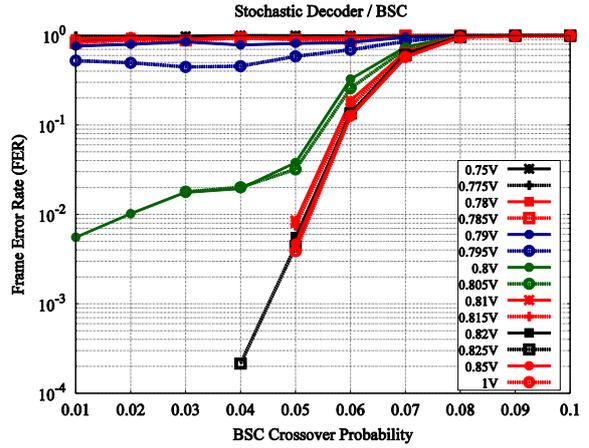


Figure 3-58: FER of Stochastic Decoder under BSC

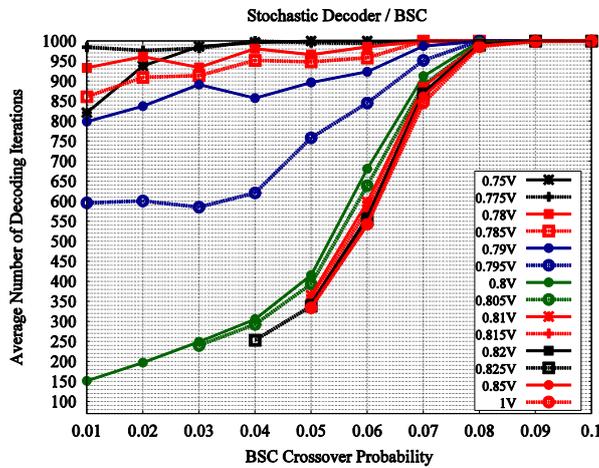


Figure 3-59: Average Number of Iterations of Stochastic Decoder under BSC

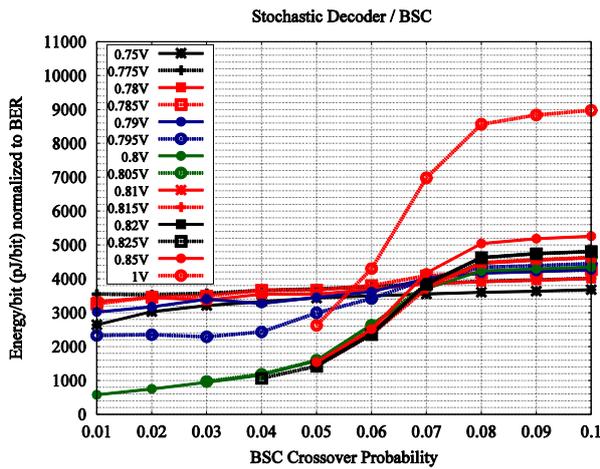


Figure 3-60: Energy/Bit Normalized to BER of Stochastic Decoder under BSC

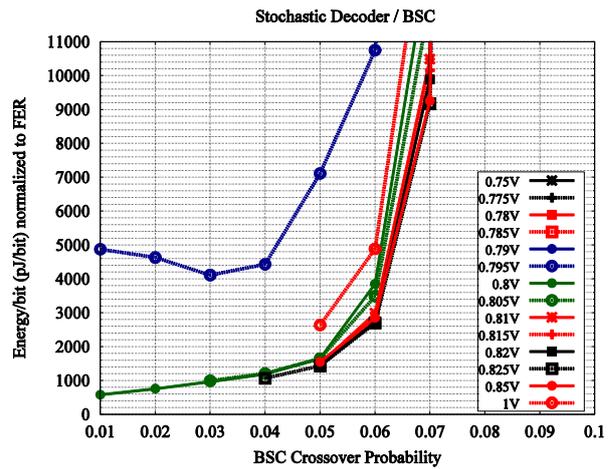


Figure 3-61: Energy/Bit Normalized to FER of Stochastic Decoder under BSC

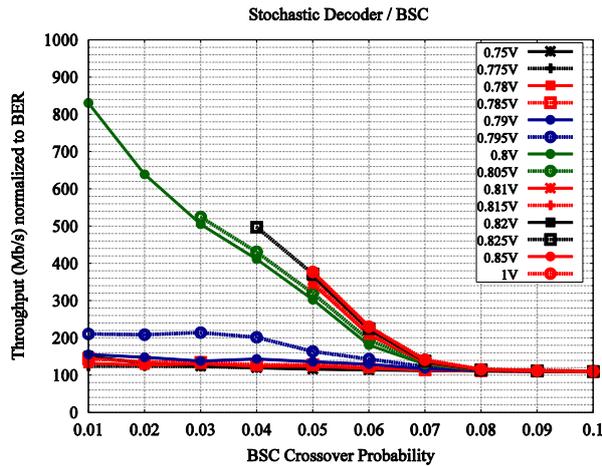


Figure 3-62: Throughput Normalized to BER of Stochastic Decoder under BSC

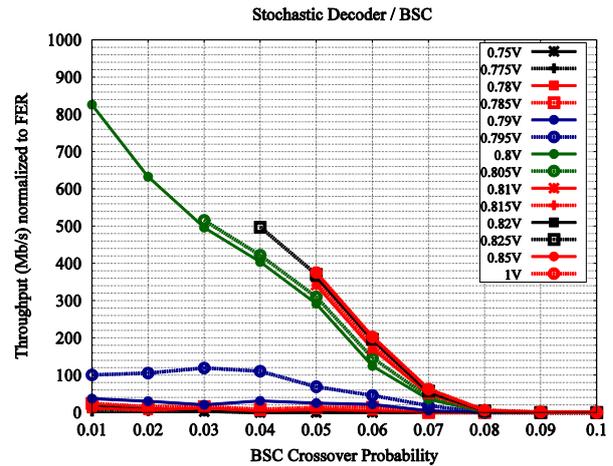


Figure 3-63: Throughput Normalized to FER of Stochastic Decoder under BSC

### 3.6.1.2. Gallager-B Decoder

Figure 3-64 to Figure 3-70 present the results of voltage scaling experiments on the Gallager B decoder under the BSC channel. The maximum number of iterations is set to 100. One can observe in these figures that:

- The impact of voltage scaling to the decoding performance is visible starting from 0.722V and it turns the decoder to its lowest decoding performance at 0.711V shown by its almost flat FER (i.e., FER = 1).
- 67% of the nominal energy (normalized to BER) can be saved when by reducing the supply voltage from 1V to 0.725V while maintaining almost the same decoding performance without any degradation of throughput.
- Surprisingly, the average number of decoding iterations doesn't vary with the supply voltage value. Even for flat FER = 1, the average number of decoding iterations is virtually the same as for nominal voltage and the BER presents a rather low error-floor ( $\approx 1E-3$  or below) when compared to the FER performance. This seems to indicate that the processing units of the decoder (including the ET) are not or only negligible affected by timing errors, but the offloaded data might not be the one on which the ET has been checked. Such a behavior could be explained by a malfunctioning control unit, which seems to be affected by timing errors well before the decoder processing units. A malfunctioning control unit could indeed result in overwriting the hard-decision (output) buffer of the decoder, after the ET condition has been fulfilled.

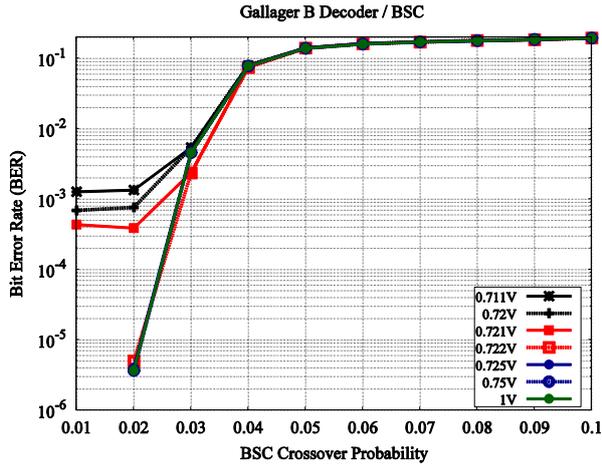


Figure 3-64: BER of Gallager B Decoder under BSC

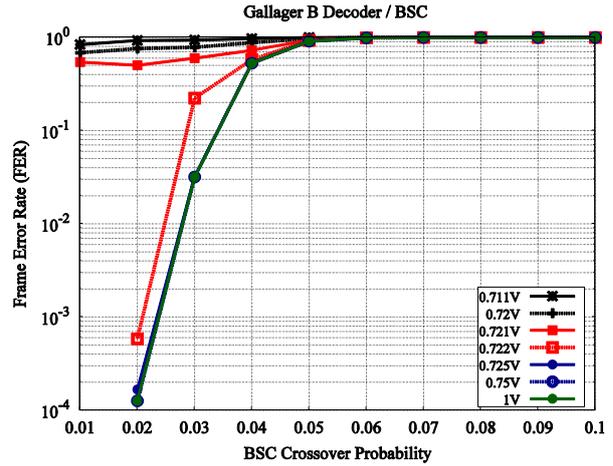


Figure 3-65: FER of Gallager B Decoder under BSC

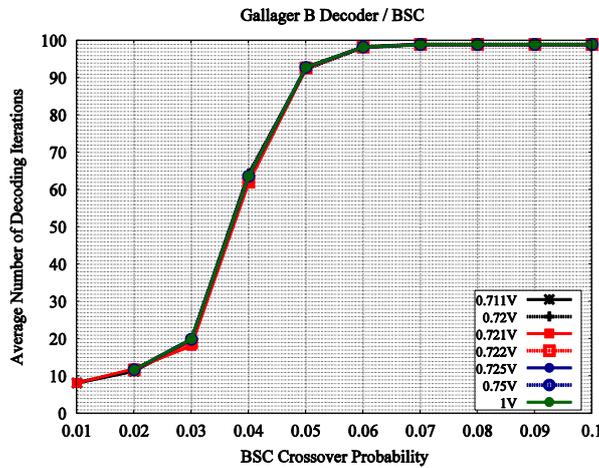


Figure 3-66: Average Number of Iterations of Gallager B Decoder under BSC

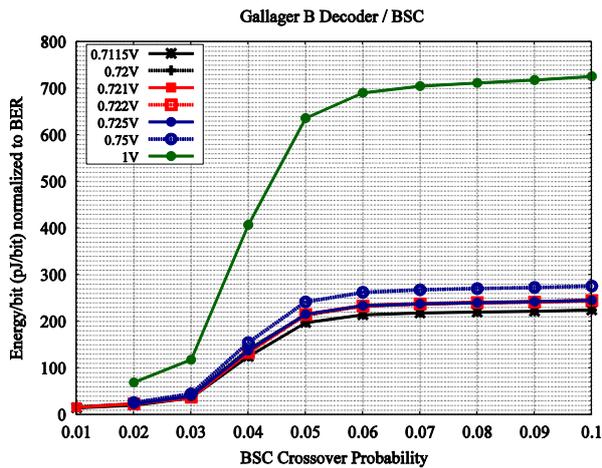


Figure 3-67: Energy/Bit Normalized to BER of Gallager B Decoder under BSC

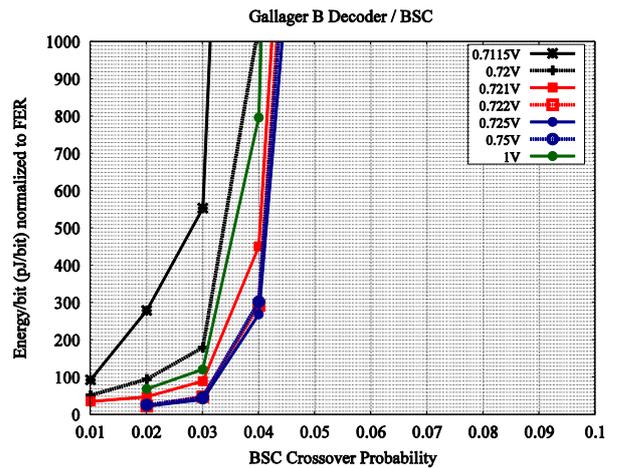


Figure 3-68: Energy/Bit Normalized to FER of Gallager B Decoder under BSC

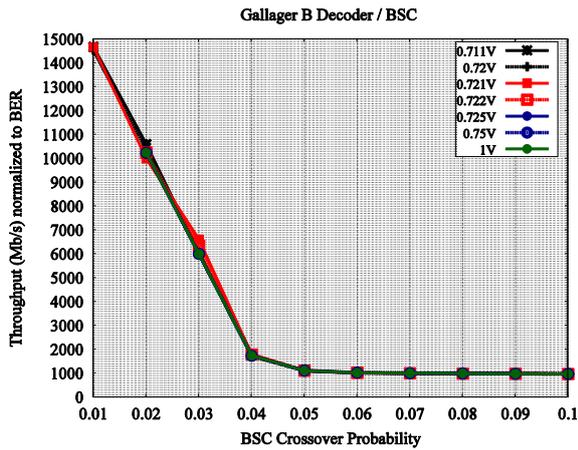


Figure 3-69: Throughput Normalized to BER of Gallager B Decoder under BSC

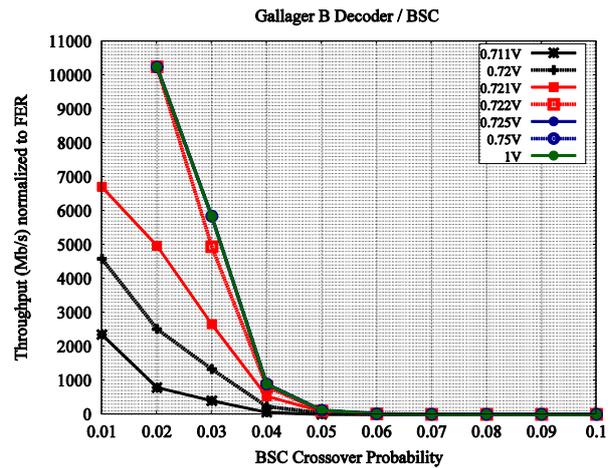


Figure 3-70: Throughput Normalized to FER of Gallager B Decoder under BSC

### 3.6.1.3. GDBF Decoder

The results of voltage scaling experiments of the GDBF decoder under the BSC channel are depicted in Figure 3-71 to Figure 3-77. The maximum number of iterations is set to 100. One can observe in these figures that:

- The decoder start to get affected by voltage scaling from 0.8V downwards and it decodes almost no received frames (*i.e.*, FER = 1) at 0.76V.
- Reducing the supply voltage from 1V to 0.81V improves energy efficiency by 60% while almost the same decoding performance is achieved without any degradation of throughput.
- The average number of decoding iterations is virtually the same as for nominal voltage for all voltage supply values, except for 0.76V when it suddenly increases to the maximum value. This seems to indicate that the control unit gets affected by timing errors at relatively high voltage supply (where the processing units are not or only negligible affected by timing errors), followed then by the processing units that get severely affected by timing errors at voltage supply equal to 0.76V.

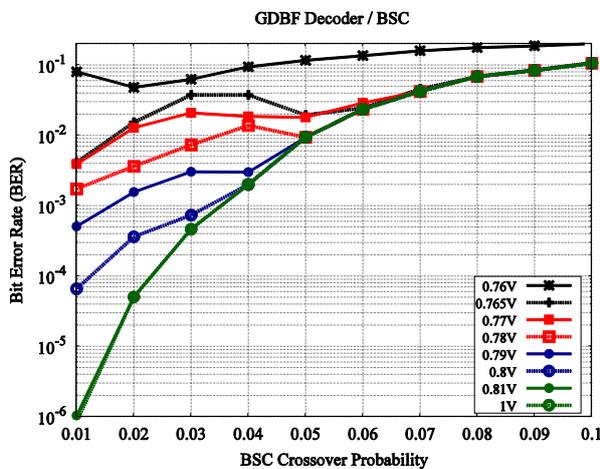


Figure 3-71: BER of GDBF Decoder under BSC

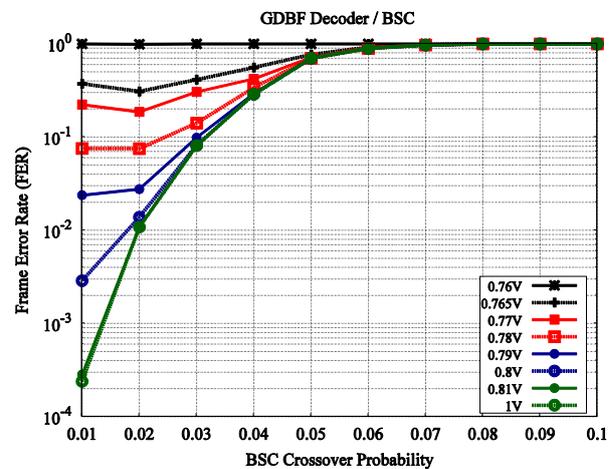


Figure 3-72: FER of GDBF Decoder under BSC

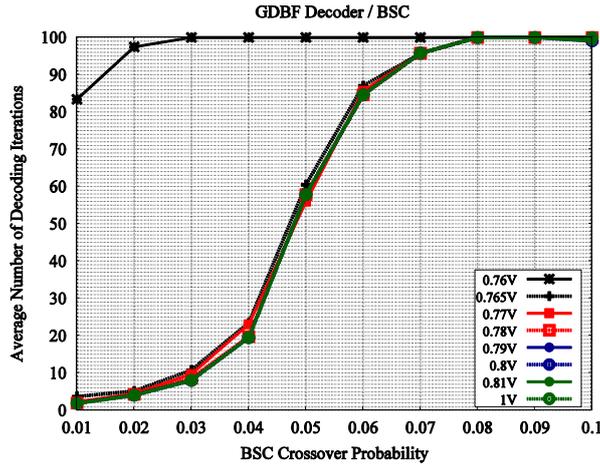


Figure 3-73: Average Number of Iterations of GDBF Decoder under BSC

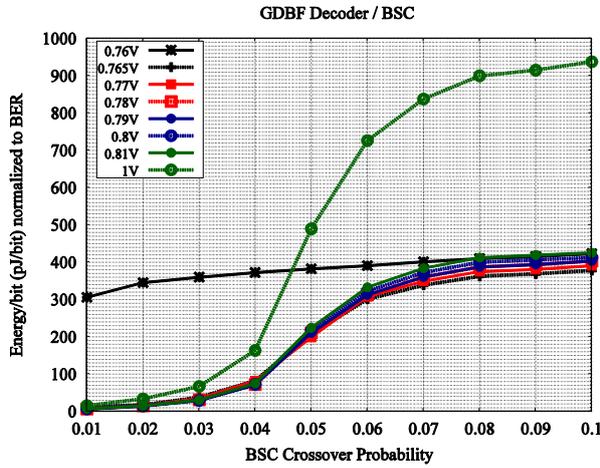


Figure 3-74: Energy/Bit Normalized to BER of GDBF Decoder under BSC

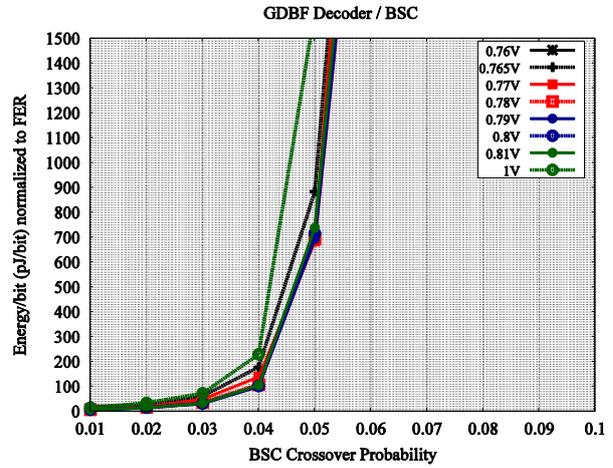


Figure 3-75: Energy/Bit Normalized to FER of GDBF Decoder under BSC

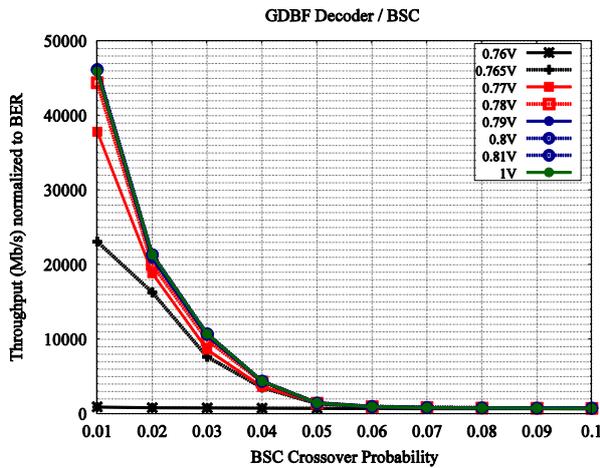


Figure 3-76: Throughput Normalized to BER of GDBF Decoder under BSC

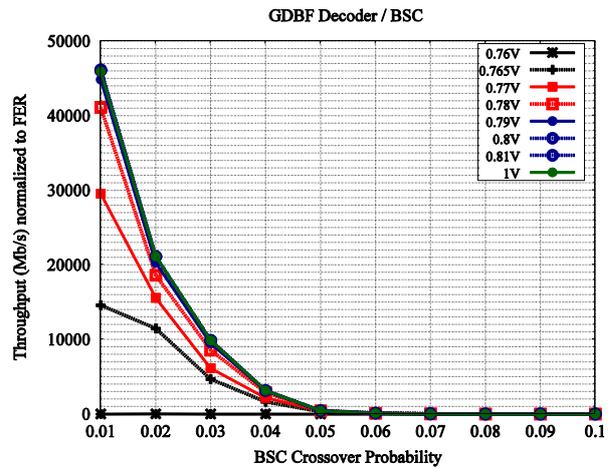


Figure 3-77: Throughput Normalized to FER of GDBF Decoder under BSC

3.6.1.4. PGDBF Decoder

Figure 3-78 to Figure 3-84 depicts the results of voltage scaling experiments of the PGDBF decoder under the BSC channel. The maximum number of iterations is set to 100. One can observe in these figures that:

- The decoding performance starts degrading at 0.85V due to voltage scaling and it goes to its worst condition in which the FER is almost flat (i.e., FER = 1) at 0.76V.
- Energy efficiency is improved by 47% while keeping almost the same decoding performance without any throughput degradation when we turn down the supply voltage from 1V to 0.85V.
- The average number of decoding iterations increases with decreasing supply voltage, which is consistent with the BER/FER degradation. Hence, unlike the GDBF decoder, we deduce that the BER/FER degradation is mainly due to timing errors affecting the processing units, and not to malfunctioning control unit.

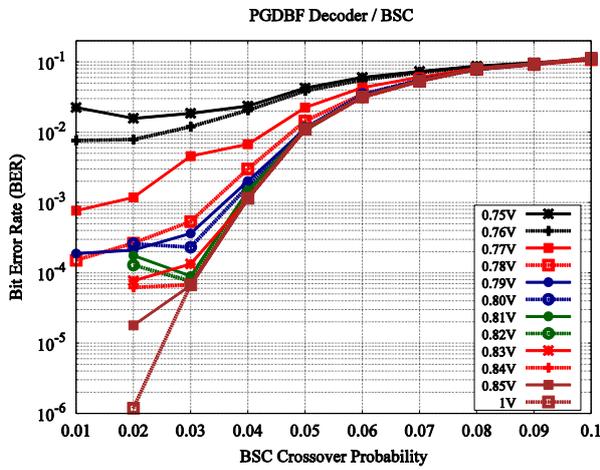


Figure 3-78: BER of PGDBF Decoder under BSC

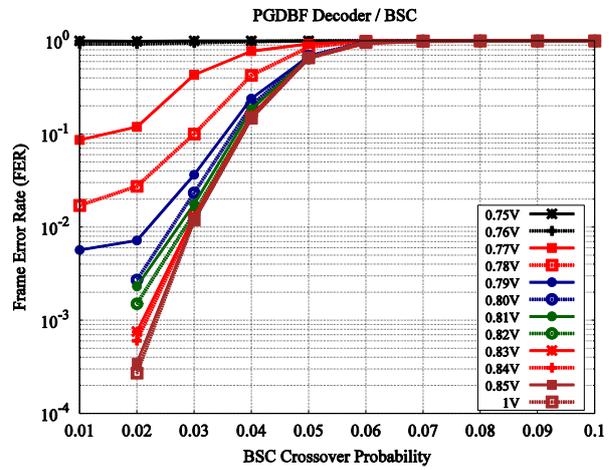


Figure 3-79: FER of PGDBF Decoder under BSC

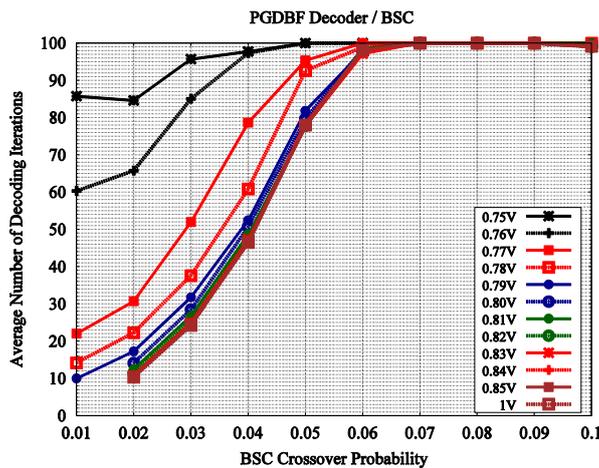


Figure 3-80: Average Number of Iterations of PGDBF Decoder under BSC

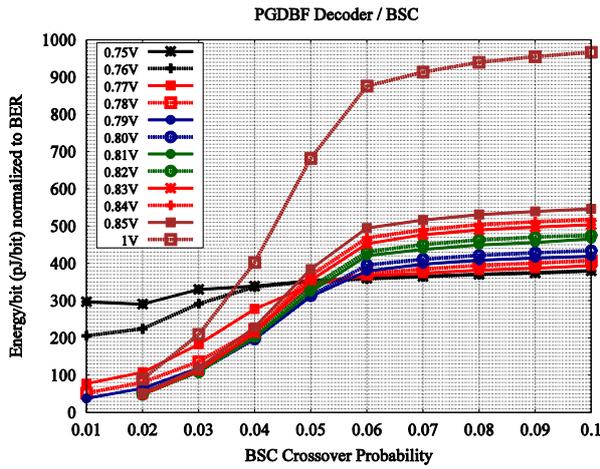


Figure 3-81: Energy/Bit Normalized to BER of PGDBF Decoder under BSC

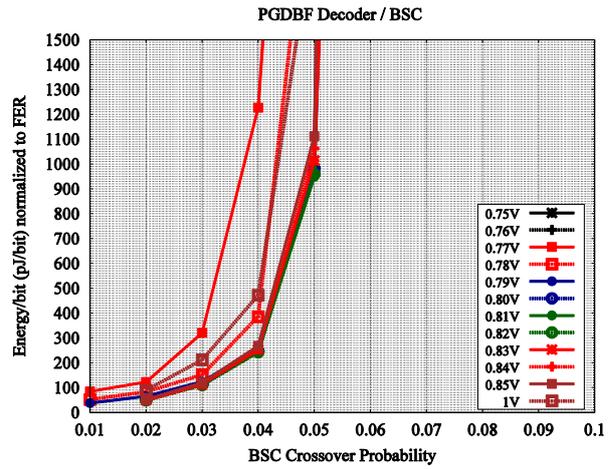


Figure 3-82: Energy/Bit Normalized to FER of PGDBF Decoder under BSC

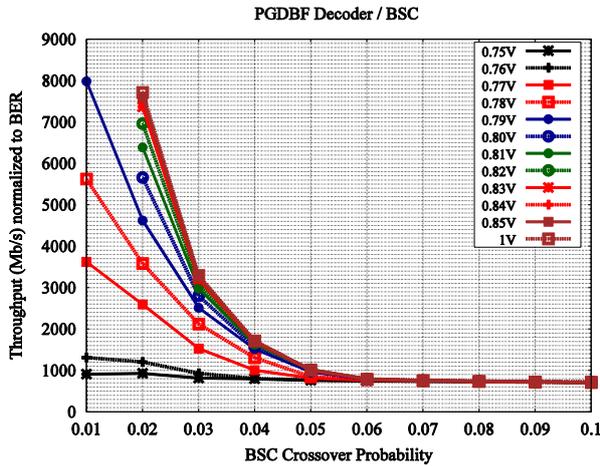


Figure 3-83: Throughput Normalized to BER of PGDBF Decoder under BSC

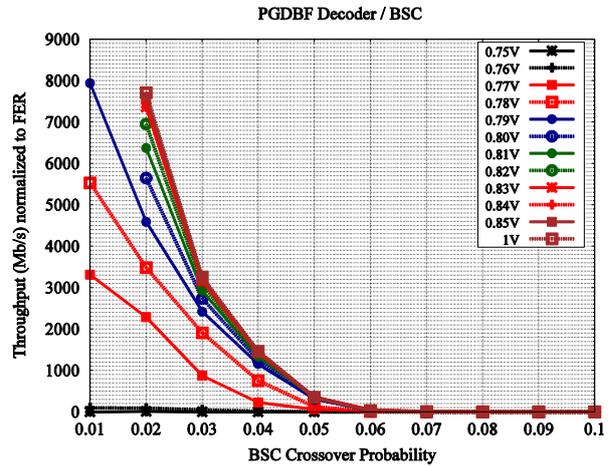


Figure 3-84: Throughput Normalized to FER of PGDBF Decoder under BSC

### 3.6.1.5. MS Decoder

Figure 3-85 to Figure 3-91 and Figure 3-92 to Figure 3-97 present the voltage scaling experiment results for the MS decoder with and without early termination (i.e., the MSnoET decoder), respectively, under the BSC channel. The maximum number of iterations is fixed at 30. One can observe in these figures that:

#### MS decoder

- The voltage scaling degrades the decoding performance starting from 0.77V and it turns the decoder to its lowest decoding performance at 0.75V shown by its almost flat FER = 1.
- The average number of decoding iterations is virtually the same as for any supply voltage value, while both BER and FER exhibit a near-flat error-floor at low BSC crossover probability. As discussed in Section 3.6.1.2, this indicates that the data offloaded by the decoder and the

one on which the ET has been checked are different, which could be explained by a malfunctioning control unit.

- The decoder consumes 53% less energy (normalized to BER) when it is operated at 0.77V, when compared to 1V operation, while having the same decoding performance without any degradation of throughput.

**MSnoET decoder**

- We note that the performance of the MSnoET decoder is virtually the same as in nominal conditions, for any simulated supply voltage down to 0.725V. This indicates that the processing units of the decoder are only slightly affected by timing errors. For voltage supply values less than 0.725V the control unit stops working (no done signal is given to the testbed), and therefore we could not continue the experiment. This confirms our guess that the performance degradation of the MS decoder is due to the control unit, and is further indicating that the control unit might fail to work properly due to the way the ET circuit is integrated to the decoder.
- The decoder consumes about 58% less energy (normalized to BER) when it is operated at 0.725V compared to 1V, while having the same decoding performance without any degradation of throughput.

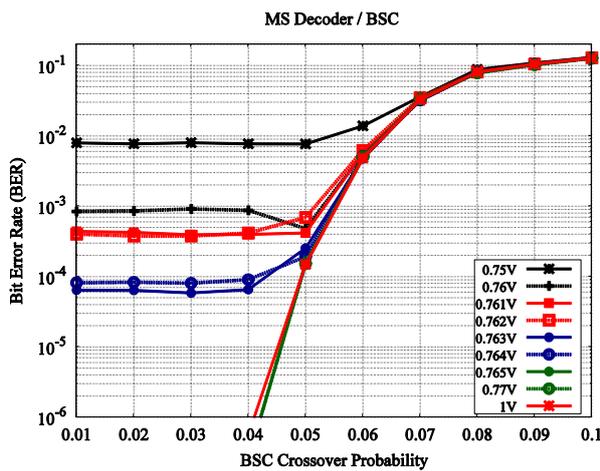


Figure 3-85: BER of MS Decoder under BSC

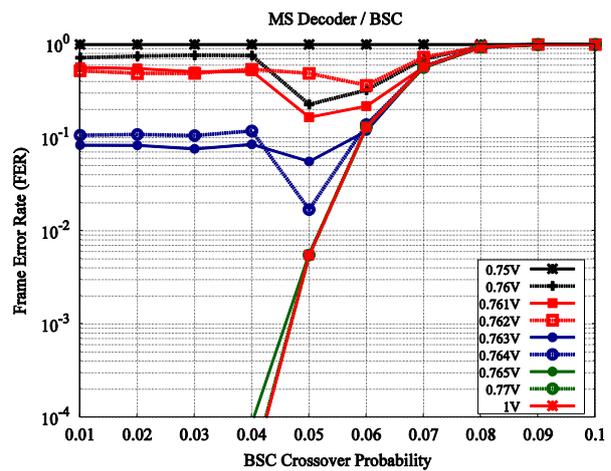


Figure 3-86: FER of MS Decoder under BSC

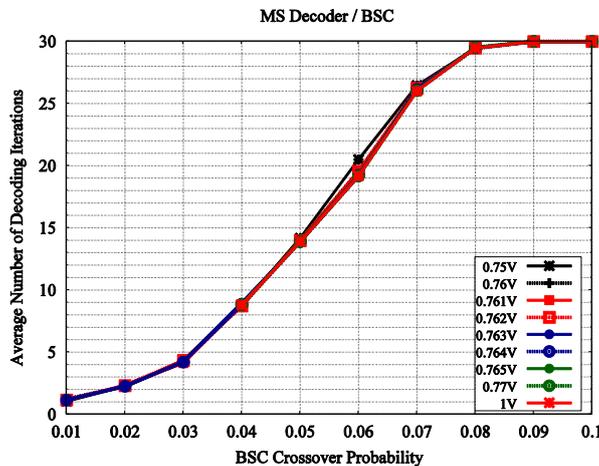


Figure 3-87: Average Number of Iterations of MS Decoder under BSC

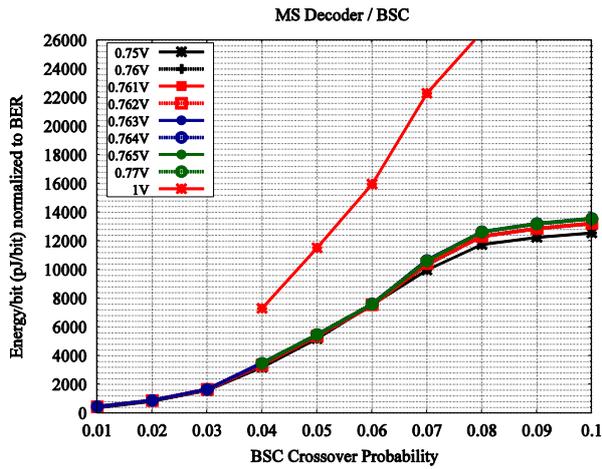


Figure 3-88: Energy/Bit Normalized to BER of MS Decoder under BSC

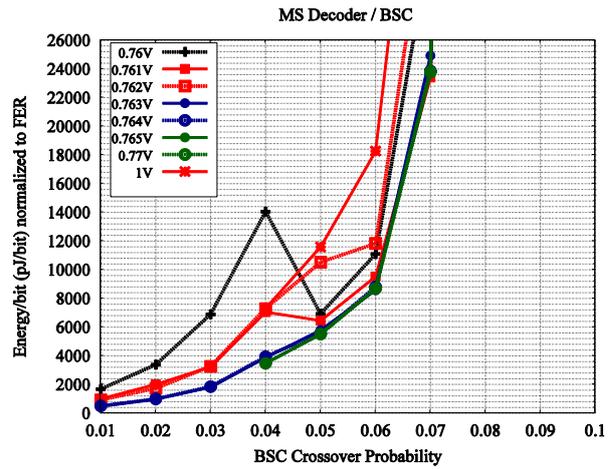


Figure 3-89: Energy/Bit Normalized to FER of MS Decoder under BSC

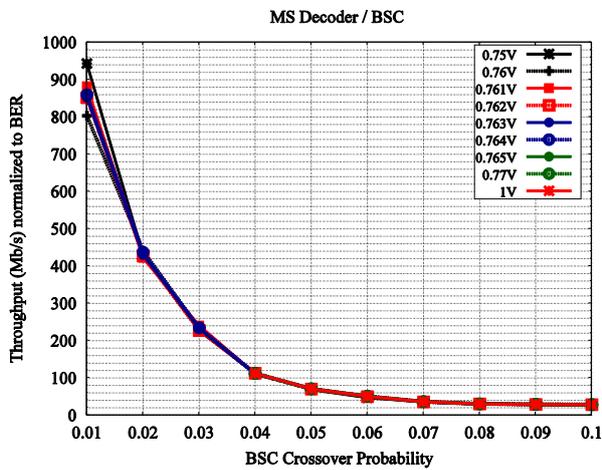


Figure 3-90: Throughput Normalized to BER of MS Decoder under BSC

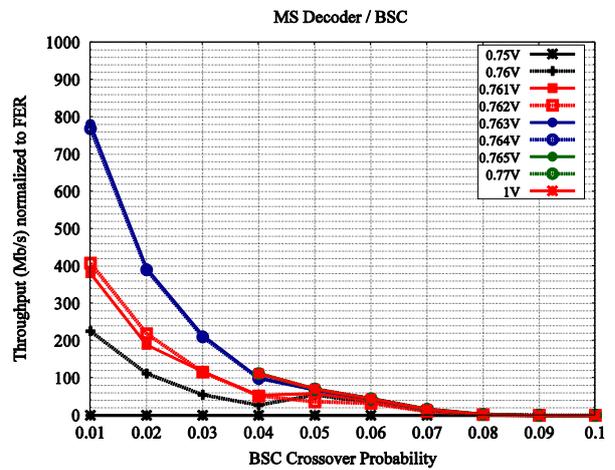


Figure 3-91: Throughput Normalized to FER of MS Decoder under BSC

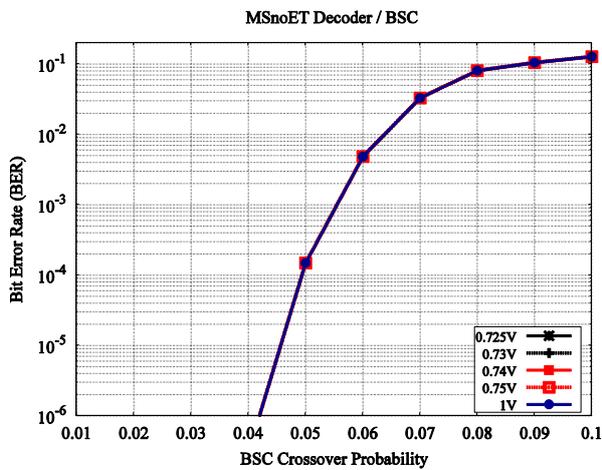


Figure 3-92: BER of MSnoET Decoder under BSC

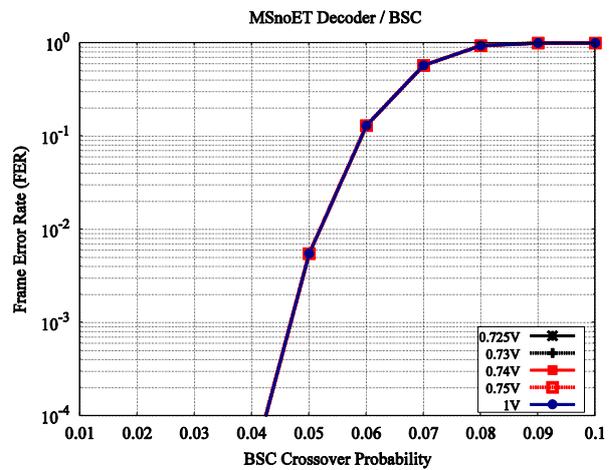


Figure 3-93: FER of MSnoET Decoder under BSC

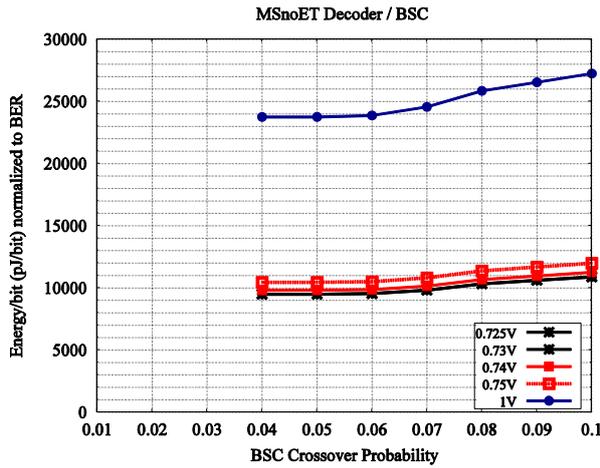


Figure 3-94: Energy/Bit Normalized to BER of MSnoET Decoder under BSC

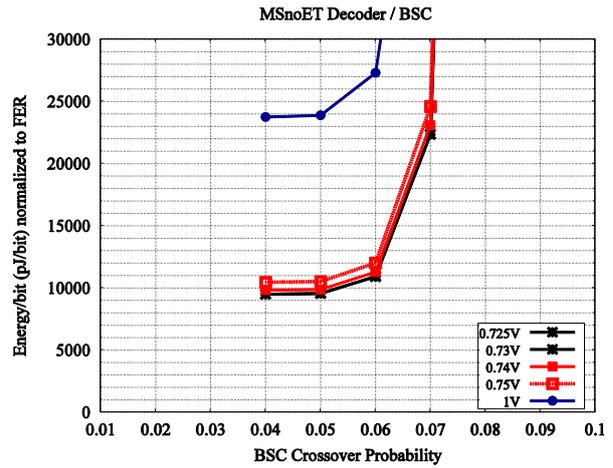


Figure 3-95: Energy/Bit Normalized to FER of MSnoET Decoder under BSC

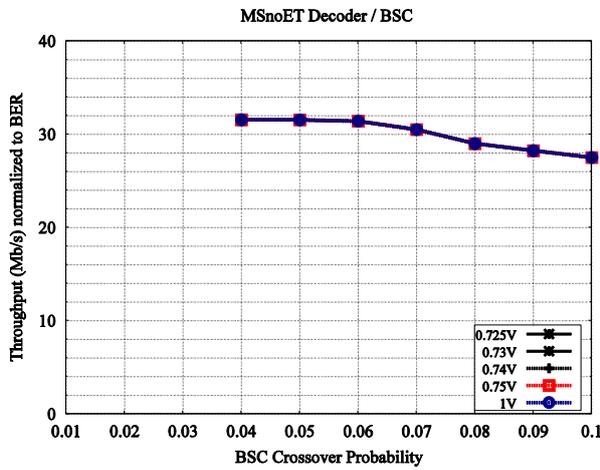


Figure 3-96: Throughput Normalized to BER of MSnoET Decoder under BSC

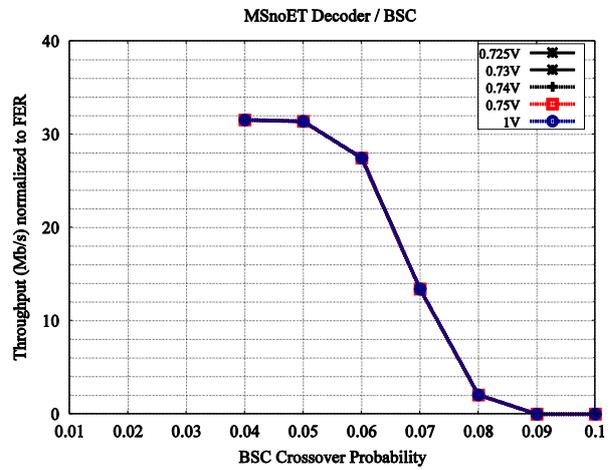


Figure 3-97: Throughput Normalized to FER of MSnoET Decoder under BSC

### 3.6.1.6. SCMS Decoder

Figure 3-98 to Figure 3-104 and Figure 3-105 to Figure 3-110 present the results of voltage scaling experiments performed on the SCMS decoder with and without early termination (i.e., SCMSnoET), respectively, under the BSC channel. The maximum number of iterations is fixed at 30. One can observe in these figures that:

#### SCMS decoder

- The voltage scaling degrades the decoding performance starting from 0.771V and it turns the decoder to its lowest decoding performance at 0.75V shown by its almost flat FER = 1.
- The average number of decoding iterations is virtually the same for any supply voltage value, while both BER and FER exhibit a near-flat error-floor at low BSC crossover probability. Similar to the MS decoder, this indicates that the data offloaded and the one on which the ET is checked are different, which could be explained by a malfunctioning control unit.

- The decoder consumes 54% less energy (normalized to BER) when it is operated at 0.78V when compared to 1V operation, while having the same decoding performance without any degradation of throughput.

**SCMSnoET decoder**

- The MSnoET decoder show virtually the same BER/FER performance at any supply voltage between 1V and 0.79V, then it suddenly reaches flat FER = 1 at 0.78V. This degradation is expected to be due to a malfunctioning control unit, although we do not have a clear indication for this (except possibly the near-flat error floor of the BER curves, similar to the behavior observed when we had strong indications of malfunctioning control unit).
- The decoder consumes about 50% less energy (normalized to BER) when it is operated at 0.79V compared to 1V, while having the same decoding performance without any degradation of throughput.

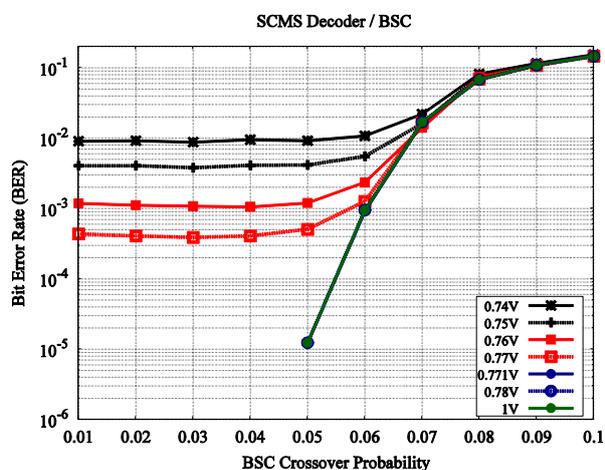


Figure 3-98: BER of SCMS Decoder under BSC

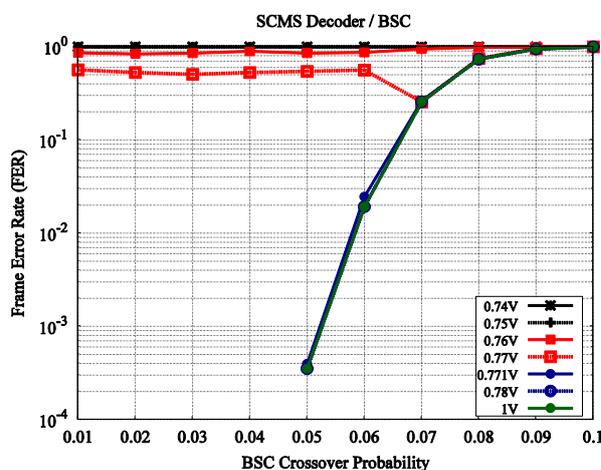


Figure 3-99: FER of SCMS Decoder under BSC

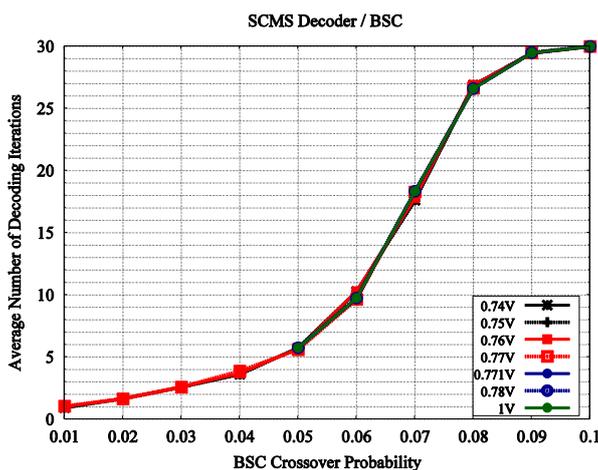


Figure 3-100: Average Number of Iterations of SCMS Decoder under BSC

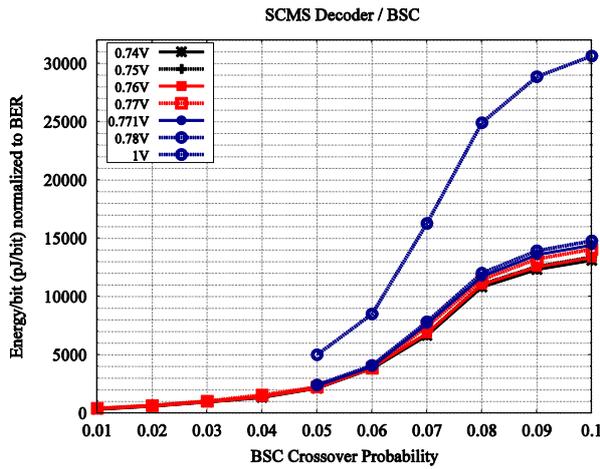


Figure 3-101: Energy/Bit Normalized to BER of SCMS Decoder under BSC

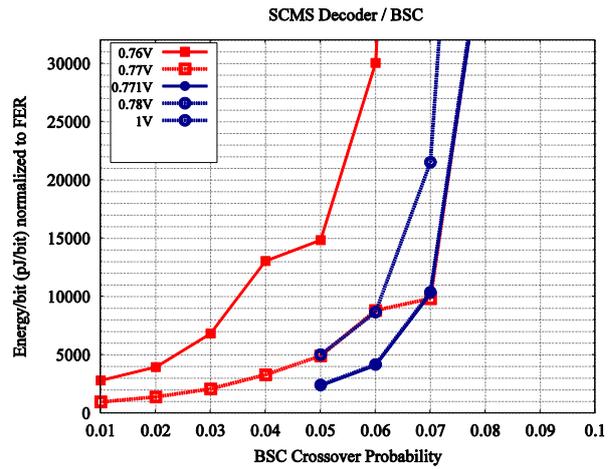


Figure 3-102: Energy/Bit Normalized to FER of SCMS Decoder under BSC

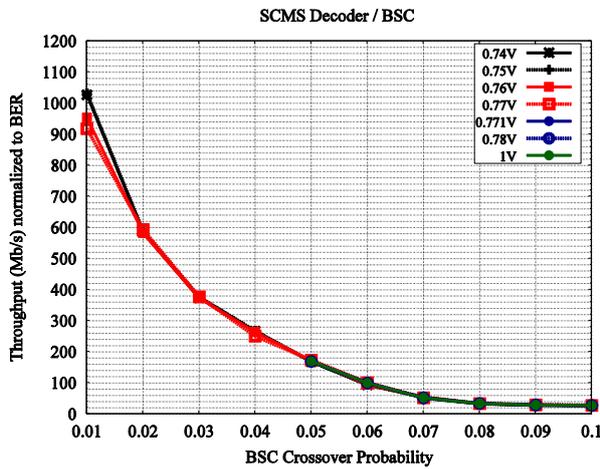


Figure 3-103: Throughput Normalized to BER of SCMS Decoder under BSC

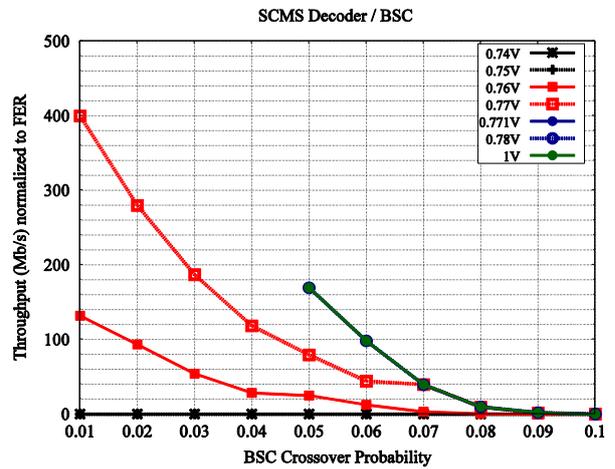


Figure 3-104: Throughput Normalized to FER of SCMS Decoder under BSC

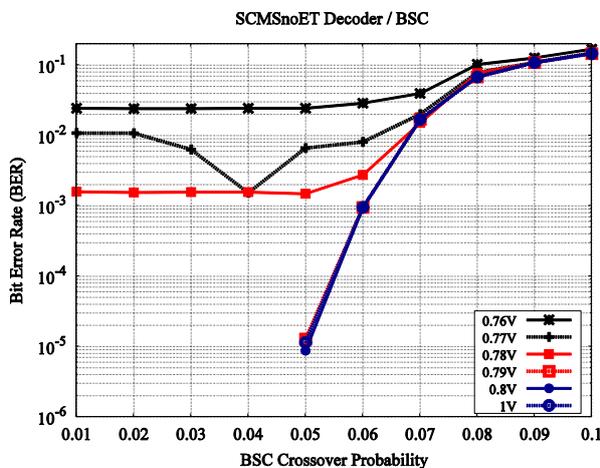


Figure 3-105: BER of SCMSnoET Decoder under BSC

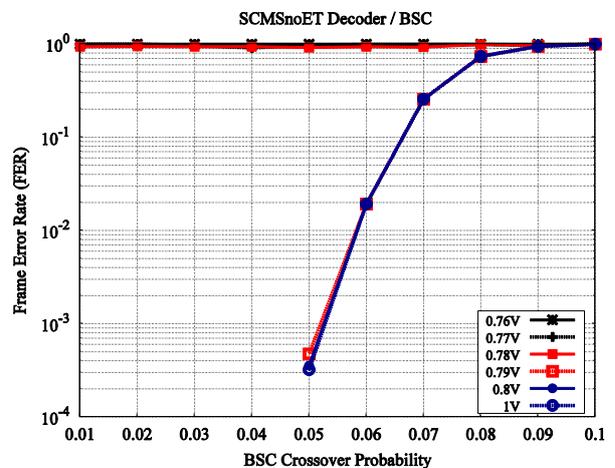


Figure 3-106: FER of SCMSnoET Decoder under BSC

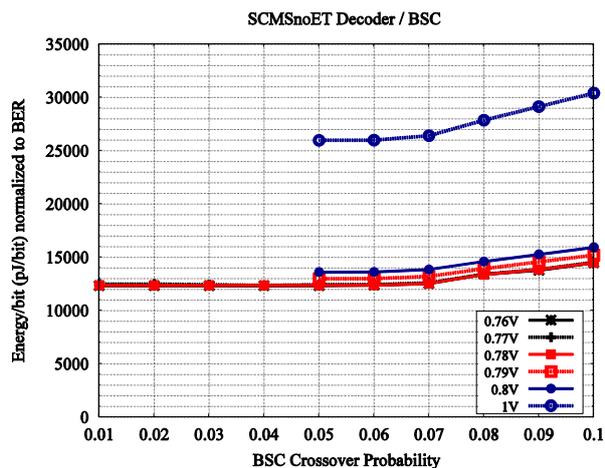


Figure 3-107: Energy/Bit Normalized to BER of SCMSnoET Decoder under BSC

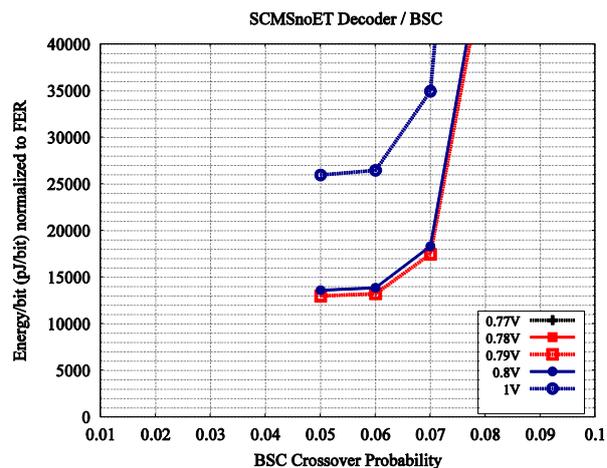


Figure 3-108: Energy/Bit Normalized to FER of SCMSnoET Decoder under BSC

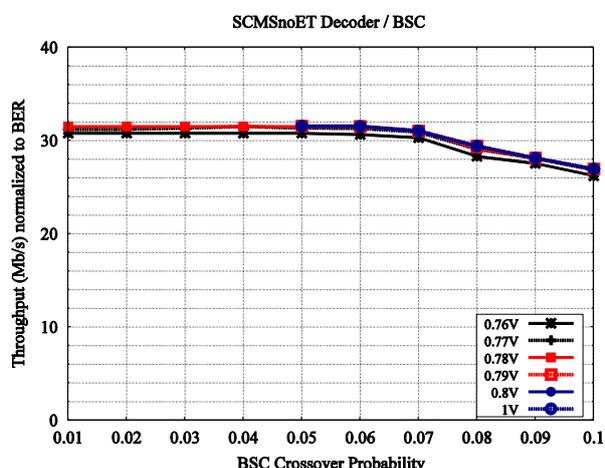


Figure 3-109: Throughput Normalized to BER of SCMSnoET Decoder under BSC

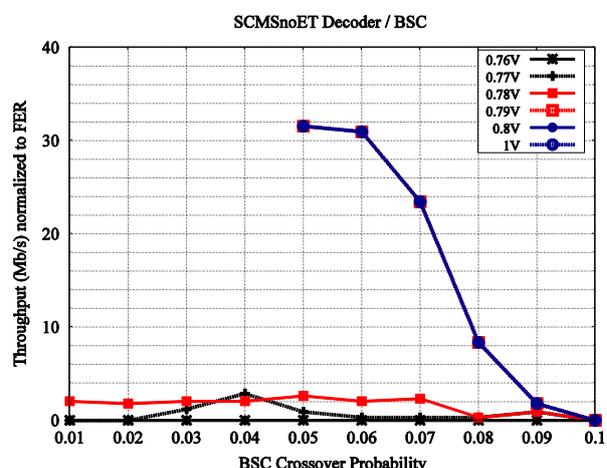


Figure 3-110: Throughput Normalized to FER of SCMSnoET Decoder under BSC

### 3.6.1.7. FAID Decoder

The results of voltage scaling experiments of the FAID decoder with and without (i.e., FAIDnoET) early termination under the BSC channel in which its maximum number of iterations is set at 30 are presented in Figure 3-111 to Figure 3-117 and in Figure 3-118 to Figure 3-123, respectively. One can observe in these figures that:

#### FAID decoder

- The degradation of decoding performance caused by voltage scaling is visible starting from 0.76V and it continues degrading till its FER is flat = 1 at 0.74V.
- The average number of decoding iterations is increasing as supply voltage decreases from 1V to 0.75V, which is consistent with the BEF/FER degradation. For supply voltage below 0.74V, the average number of decoding iterations is way below its maximum value, although the achieved FER is virtually flat equal to 1. This indicates that the ET termination circuit is severely affected by timing errors, hence reporting a correct codeword and making the decoder stop while it shouldn't.

- Although FAID architecture and implementation present many similarities with those of MS and SCMS decoders, we note that for the FAID there is no clear evidence of any kind of control unit failure.
- Turning down the supply voltage from 1V to 0.77V improves energy efficiency (normalized to BER) by 58%, without almost any degradation of the decoding performance or of the throughput.

**FAIDnoET decoder**

- The degradation of decoding performance caused by voltage scaling is visible starting from 0.79V and it continues degrading till its FER is flat = 1 at 0.76V.
- We also note that the decoding performance of the FAIDnoET decoder is degraded with respect to that of the FAID decoder. This confirms the fact that – for properly working control unit – the ET circuit may improve the decoder performance, as discussed in the first paragraph of Section 3.6.1. The intuition behind goes as follows: when the number or timing errors affecting the processing units is moderate, the decoder manages to handle them, so that the bit error rate is maintained at a low level throughout the iterative process. The number of bit errors from one iteration to another may vary (increase or decrease), but if the bit error probability is low enough, the decoder will eventually reach an error free iteration when the absence of errors is eventually detected by the ET circuit and the decoder stops. When the ET circuit is not implemented, the decoder stops when it reaches the maximum number of decoding iterations, while there is no particular reason for that the last iteration to be error free.

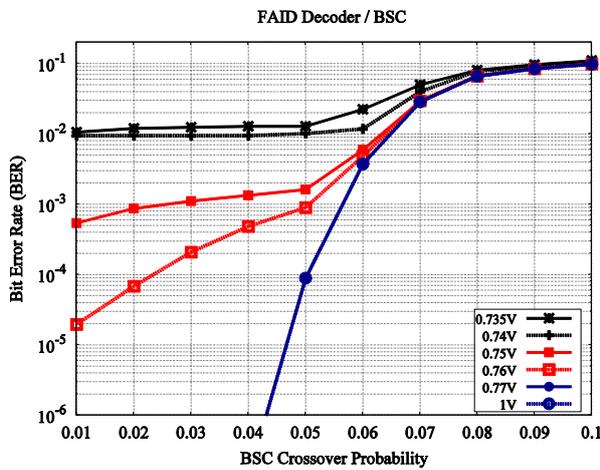


Figure 3-111: BER of FAID Decoder under BSC

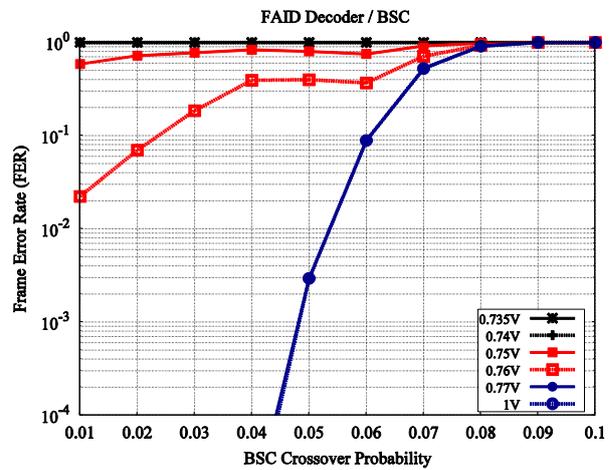


Figure 3-112: FER of FAID Decoder under BSC

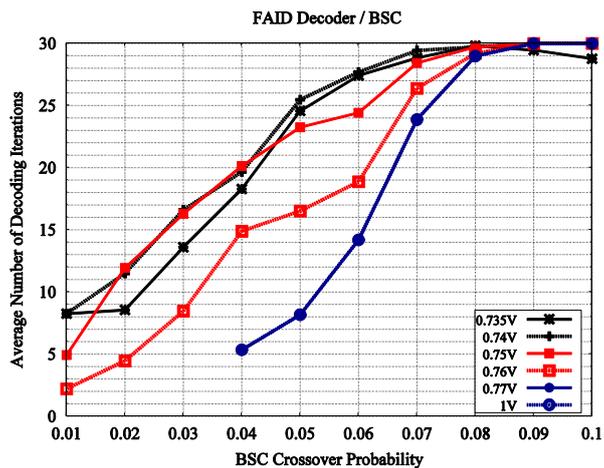


Figure 3-113: Average Number of Iterations of FAID Decoder under BSC

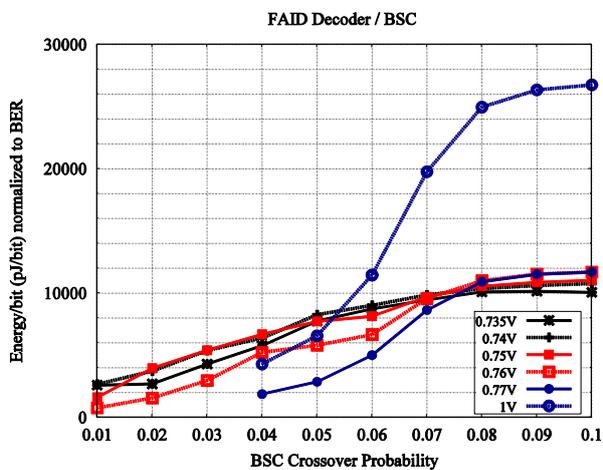


Figure 3-114: Energy/Bit Normalized to BER of FAID Decoder under BSC

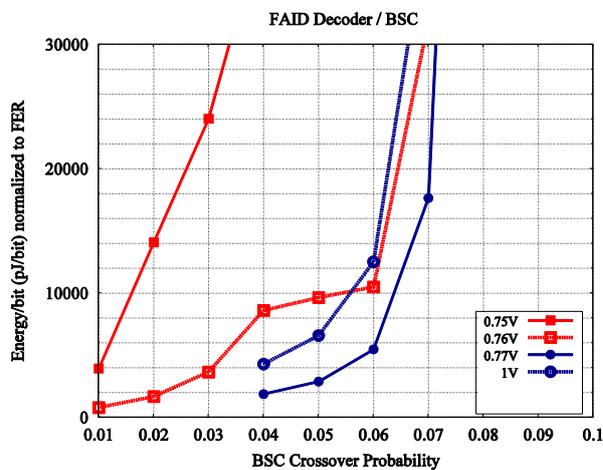


Figure 3-115: Energy/Bit Normalized to FER of FAID Decoder under BSC

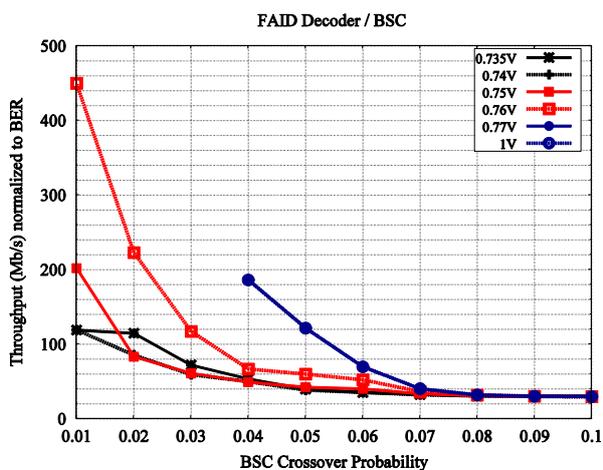


Figure 3-116: Throughput normalized to BER of FAID Decoder under BSC

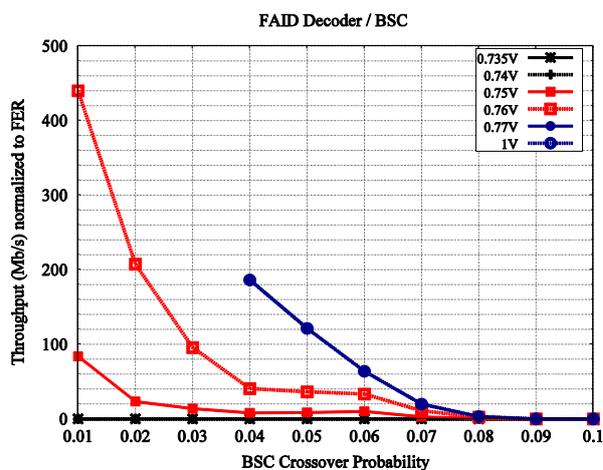


Figure 3-117: Throughput Normalized to FER of FAID Decoder under BSC

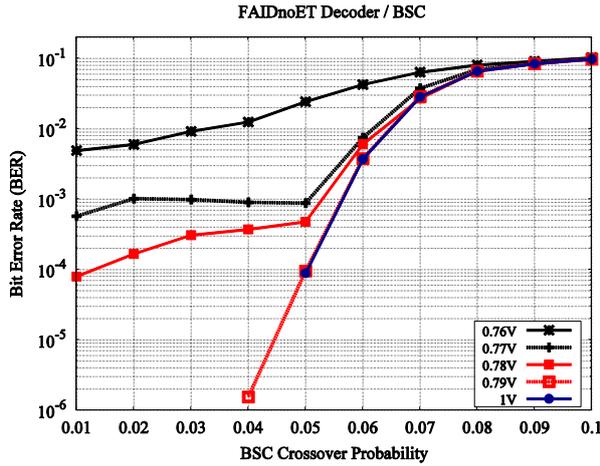


Figure 3-118: BER of FAIDnoET Decoder under BSC

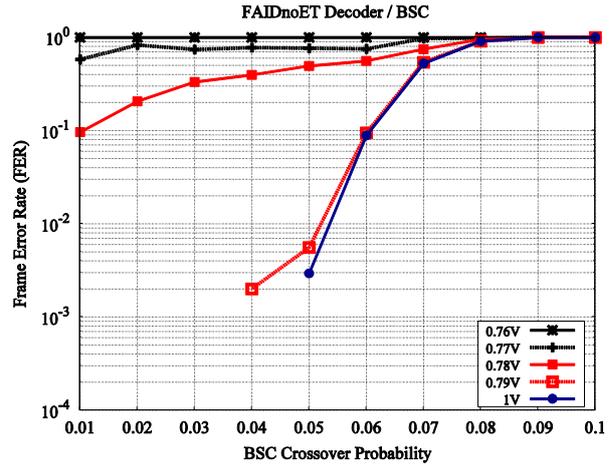


Figure 3-119: FER of FAIDnoET Decoder under BSC

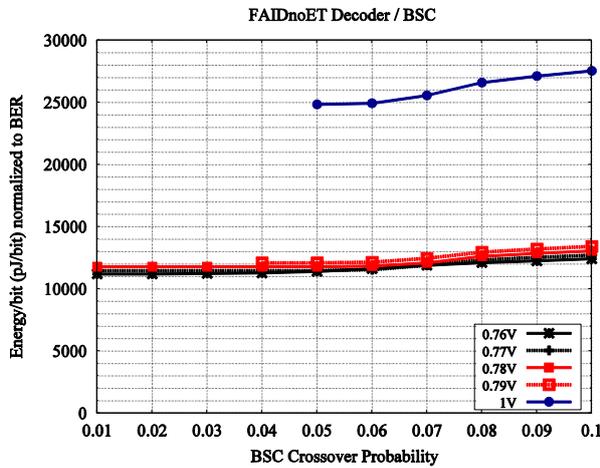


Figure 3-120: Energy/Bit Normalized to BER of FAIDnoET Decoder under BSC

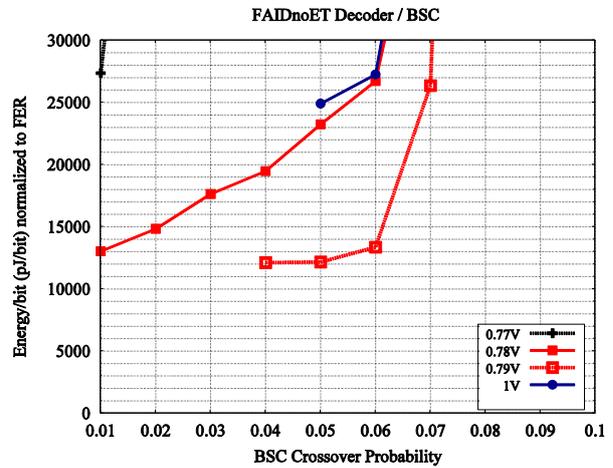


Figure 3-121: Energy/Bit Normalized to FER of FAIDnoET Decoder under BSC

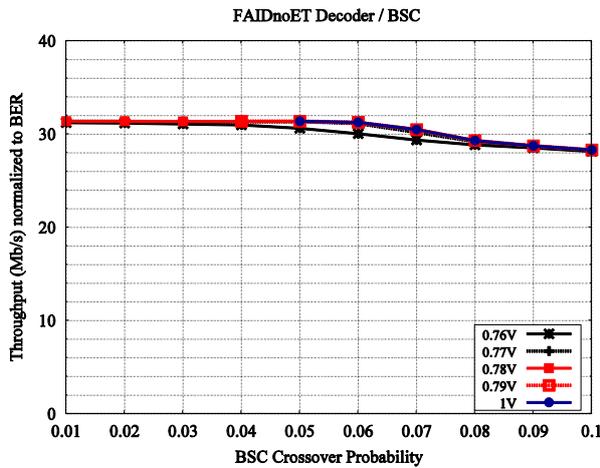


Figure 3-122: Throughput Normalized to BER of FAIDnoET Decoder under BSC

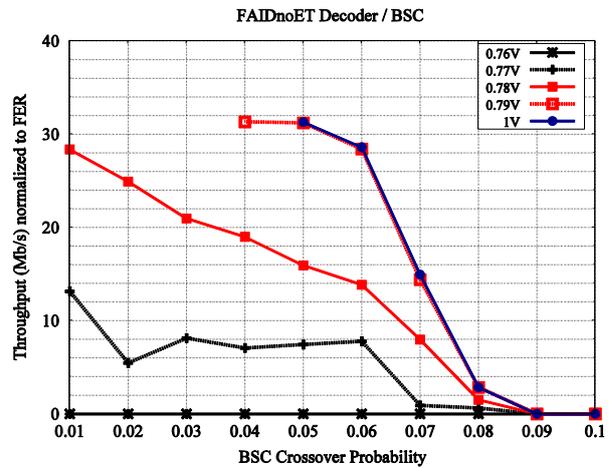


Figure 3-123: Throughput Normalized to FER of FAIDnoET Decoder under BSC

### 3.6.1.8. Voltage Scaling Sensitivity (VSS)

In this section we summarize the evaluation results for the 10 decoders under test, from the VSS metric perspective. In order to capture the way a decoder reacts to the voltage scaling process, the Performance Preservation Region (PPR) and Performance Degradation Region (PDR) have been introduced in Section 3.2. These regions are delimited by the  $V_{pp}$  and  $V_{pd}$  voltage values, which are estimated based on the Monte-Carlo simulation results reported previously:

- $V_{pp}$  = the lowest voltage value for which the FER is the same as for nominal  $V_{dd}$
- $V_{pd}$  = the highest voltage value for which the FER is flat, equal to 1

With the above notation, it follows that:

$$PPR = V_{dd} - V_{pp} \quad \text{and} \quad PDR = V_{pp} - V_{pd}$$

PPR measures the decoder potential to save energy while providing its expected performance while PDR measures how much energy savings one can still get if the channel conditions permit.

Table 3-28 shows the PPR and PDR metrics for the decoders under test. Reported values for  $V_{pp}$  and  $V_{pd}$  have been estimated by investigating the FER performance of each decoder for various supply voltage values. The accuracy of the estimation depends on the increment/decrement between supply voltage values for which the decoder has been evaluated. The uncertainty intervals and the error margins for the  $V_{pp}$  and  $V_{pd}$  estimates are shown in Table 3-29.

The uncertainty intervals shown in Table 3-29 should be understood as follows (we include here the explanation for the SD only, the same holds true for all the other decoders):

- Uncertainty interval for  $V_{pp}$  (SD decoder):
  - The achieved FER is (virtually) the same, for any supply voltage value starting from 1V down to 0.82V. The next simulated supply voltage value is 0.815V – however, for this supply voltage the FER presents a visible degradation
  - Therefore, the  $V_{pp}$  value reported in Table 3-28 is 0.82V (smallest value without degradation), while the actual value could be within the interval [0.815V – 0.82V]. The error margin of the estimation is thus equal to 0.005V.
- Uncertainty interval for  $V_{pd}$  (SD decoder):
  - The achieved FER is (virtually) flat equal to 1, for all supply voltage values up to 0.78V. The next simulated supply voltage value is 0.785V – however, for this voltage value the FER is no longer flat
  - Therefore, the  $V_{pd}$  value reported in Table 3-28 is 0.78V (highest value with flat FER = 1), but the actual value could be within the interval [0.78V – 0.785V]. The error margin of the estimation is thus equal to 0.005V.

Note that values highlighted in red in Table 3-29 are  $V_{pp}$  and  $V_{pd}$  reported in Table 3-28.

Table 3-28: PPR and PDR Metrics for the Decoders under Test (Voltage Scaling Scenario)

	SD	GB	GDBF	PGDBF	MS	MSnoET	SCMS	SCMSnoET	FAID	FAIDnoET
Vdd	1	1	1	1	1	1	1	1	1	1
Vpp	0,82	0,725	0,81	0,85	0,765	0,725	0,771	0,79	0,77	0,79
Vpd	0,78	0,711	0,76	0,76	0,75	N/A	0,75	0,78	0,74	0,76
PPR	0,18	0,275	0,19	0,15	0,235	0,275	0,229	0,21	0,23	0,21
PDR	0,04	0,014	0,05	0,09	0,015	N/A	0,021	0,01	0,03	0,03
Comments	(1)	(2)	(2)	(1)	(2)	(3)	(2)	(1)	(1)	(1)

- (1) Errors in the  $V_{pd} - V_{pp}$  interval are most likely due to timing errors affecting processing units;
- (2) Errors in the  $V_{pp} - V_{dd}$  interval are most likely due to malfunctioning controller, while processing units (including ET) seem to work properly;
- (3) Same performance as for nominal  $V_{dd}$  for any supply voltage value - in particular, the actual  $V_{pp}$  value is lower than the displayed one.

Table 3-29: Uncertainty Intervals for  $V_{pp}$  and  $V_{pd}$  Estimates

	SD	GB	GDBF	PGDBF	MS	MSnoET	SCMS	SCMSnoET	FAID	FAIDnoET
Vpp	[0,815-0,82]	[0,722-0,725]	[0,8-0,81]	[0,84-0,85]	[0,764-0,765]	$\leq 0,725^{(*)}$	[0,77-0,771]	[0,78-0,79]	[0,76-0,77]	[0,78-0,79]
Vpd	[0,78-0,785]	[0,711-0,72]	[0,76-0,765]	[0,76-0,77]	[0,75-0,76]	N/A	[0,75-0,76]	[0,78-0,79]	[0,74-0,75]	[0,76-0,77]
error margins	0,005	0,003	0,01	0,01	0,001	N/A	0,001	0,01	0,01	0,01
	0,005	0,009	0,005	0,01	0,01	N/A	0,01	0,01	0,01	0,01

(\*) Lower supply voltages have not been simulated because the control unit stops working

Table 3-30 depicts the energy savings of the evaluated decoders when we operate them at  $V_{pp}$  instead of nominal voltage. Energy savings are computed in terms of energy/bit normalized to BER, and in order to have a fair comparison, they are evaluated for all decoders at the same BER =  $10^{-4}$ . The BSC crossover probability for which the decoders achieve the target BER =  $10^{-4}$  is also reported in the table. It can be seen that the energy savings vary between 45% and 67%. The GB decoder presents the highest potential for energy savings, but it also exhibits the second worst error correction performance. The best error correction performance is provided by the SCMS and SCMSnoET decoders, while they allow for energy savings of about 50%.

Table 3-30: Energy Savings of Evaluated Decoders Operating at  $V_{pp}$  (@BER =  $10^{-4}$ )

	SD	GB	GDBF	PGDBF	MS	MSnoET	SCMS	SCMSnoET	FAID	FAIDnoET
Energy savings	50 %	67 %	52 %	45 %	53 %	60 %	52 %	50 %	56 %	54 %
BSC crossover probability	0,048	0,025	0,023	0,031	0,049	0,049	0,055	0,055	0,05	0,05

### 3.6.2. Fault Injection

Simulated fault injection has been applied to the MS, SCMS, and FAID architectures described in Sections 3.5.1, 3.5.2, and 3.5.3. The fault injection methodology, the error profile characterization and the fault map generation are described in Section 3.3.2 and Section 3.3.3, respectively. In this section, we are presenting the simulated fault injection results.

Decoders are evaluated in terms of BER, FER, and average number of decoding iterations. MS and SCMS decoders are evaluated for both Binary Input Additive White Noise Gaussian (BI-AWGN) and BSC channel models. As explained in Section 3.6.1, the FAID decoder operates on binary input data, thus it is only evaluated for the BSC model.

#### 3.6.2.1. Analysis of LDPC Decoders under BI-AWGN

The inputs of the MS and SCMS decoder are 4-bit signed integers, representing quantized Log-Likelihood Ratio (LLR) values. A gain factor – referred to as channel scale factor in [i-RISC/D3.1] – is first applied on the channel output, which is then quantized and fed to the decoder input. The optimization of the channel scale factor has been addressed in [i-RISC/D3.1]. Here, we use a channel scale factor equal to 3.5, which yields good performance in both the waterfall and the error-floor region.

Figure 3-124, Figure 3-125, and Figure 3-126 depict the average number of iterations, the BER, and the FER for faulty MS architecture for BI-AWGN. The figures indicate that the MS decoder has the same decoding performance for a clock period of 3.1ns as a fault free decoder. The average error rate for a decoder with a clock period of 3.1ns is of order  $10^{-5}$ , with a maximum of order  $10^{-4}$ . A slight decoding performance degradation (of less than 0.1dB for a  $10^{-5}$  BER) is observed when clock frequency is increased to 400MHz (clock period of 2.5ns). The average error rate in this case is of order  $10^{-4}$ , while the maximum error rate is of order  $10^{-3}$ . Therefore, average error rates of up to  $10^{-4}$  in the decoder do not or only slightly affect the error correction capability. Significant decoding performance degradation is observed for a clock period of 2.2ns, which corresponds to an average error rate of  $10^{-3}$ . For a clock period of 1.9ns, the MS decoder cannot decode.

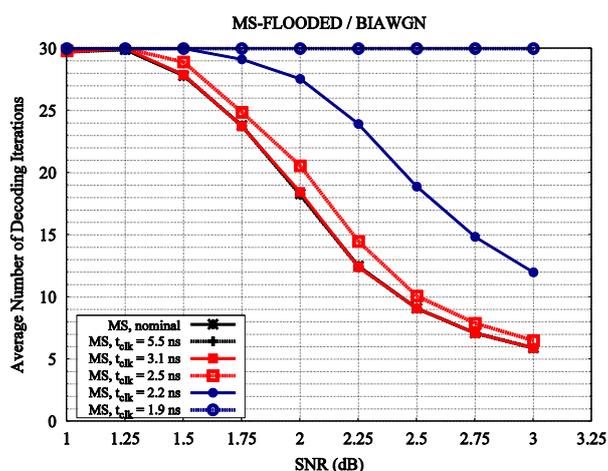


Figure 3-124: Average Number of Iterations for Faulty MS under BI-AWGN

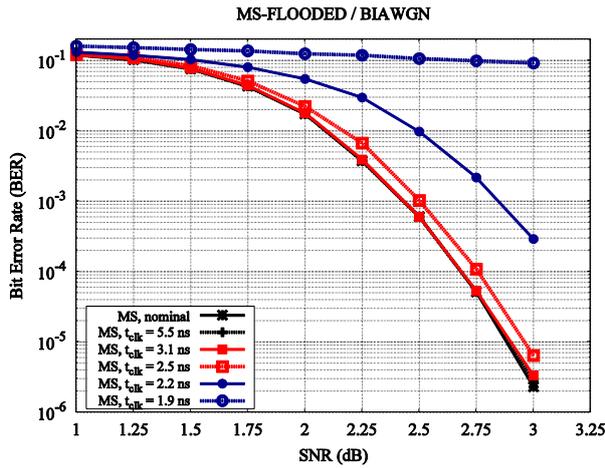


Figure 3-125: BER for Faulty MS under BI-AWGN

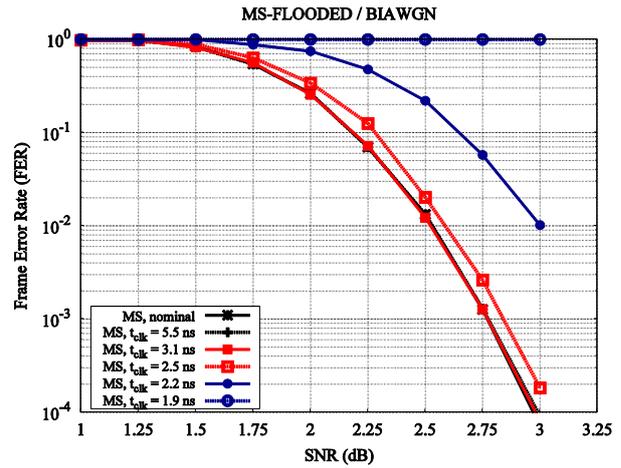


Figure 3-126: FER for Faulty MS under BI-AWGN

Figure 3-127, Figure 3-128, and Figure 3-129 depict the average number of iterations, the BER, and the FER for faulty SCMS architecture for BI-AWGN, when errors are injected in all memories (including the memory for previous  $\alpha$  message signs and the memory for erasure bits). The figures indicate that the SCMS decoder has the same decoding performance for a clock period of 2.5ns as a fault free decoder. For a clock period of 2.2ns, the SCMS decoder exhibits an error floor starting at  $\text{SNR} \approx 2.5\text{dB}$  ( $\text{FER} \approx 4 \times 10^{-3}$ ,  $\text{BER} \approx 6 \times 10^{-5}$ ). As for the MS decoder, for a clock period of 1.9ns, the circuit has no error correction capability.

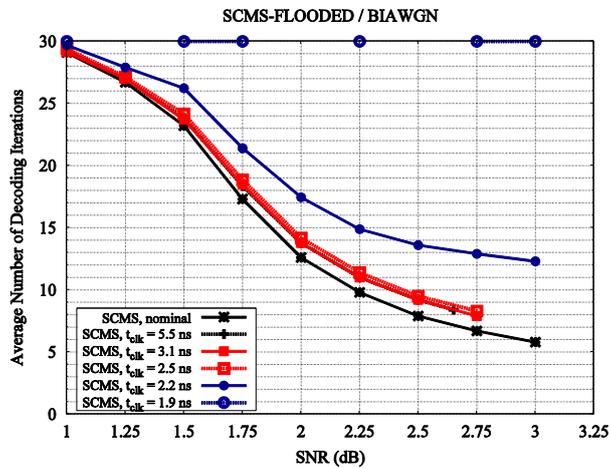


Figure 3-127: Average Number of Iterations for Faulty SCMS under BI-AWGN with Errors Injected in the Two Additional Memories

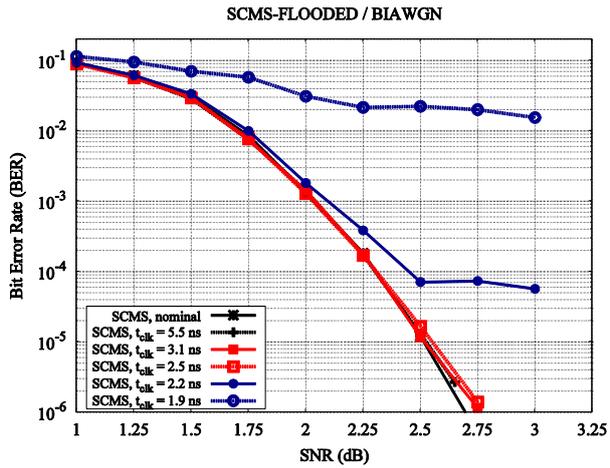


Figure 3-128: BER for Faulty SCMS under BI-AWGN with Errors Injected in the Two Additional Memories

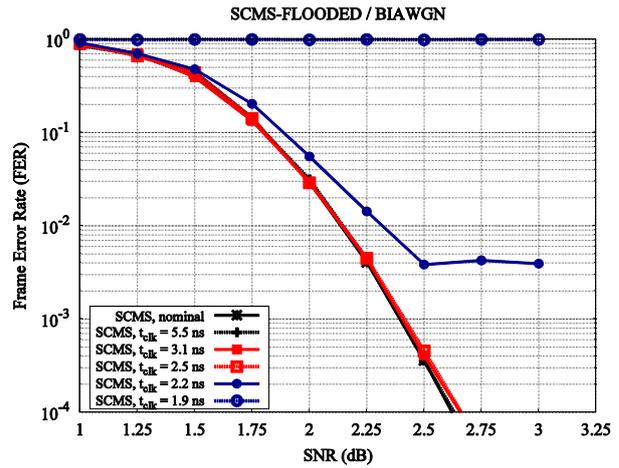


Figure 3-129: FER for Faulty SCMS under BI-AWGN with Errors Injected in the Two Additional Memories

Figure 3-130, Figure 3-131, and Figure 3-132 depict the average number of iterations, the BER, and the FER for faulty SCMS architecture for BI-AWGN, when errors are not injected in the two additional memories (the memory for previous  $\alpha$  message signs and the memory for erasure bits). When errors do not affect the two memories, a small decoding performance decrease (of less than 0.1 dB) is obtained for a clock period of 2.2 ns. With respect to the situation when these two memories are injected with faults, the decoder does not longer exhibit the error floor phenomena at  $\text{SNR} \approx 2.5\text{dB}$ .

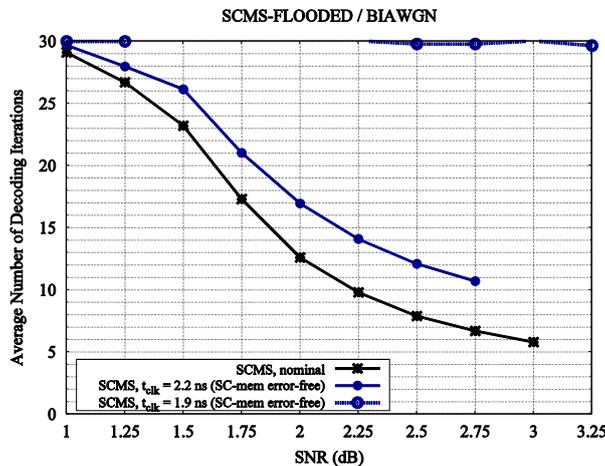


Figure 3-130: Average Number of Iterations for Faulty SCMS under BI-AWGN with No Errors Injected in the Two Additional Memories

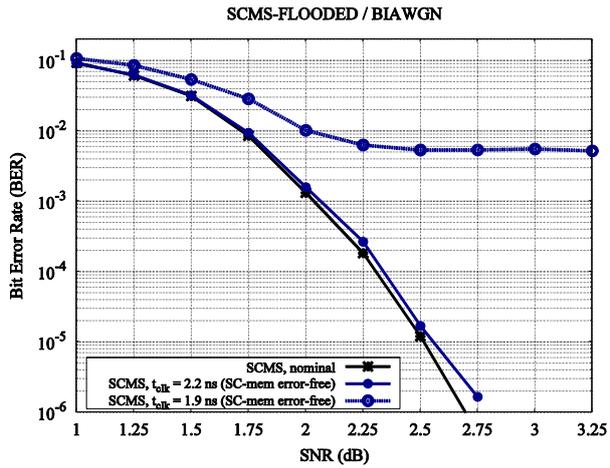


Figure 3-131: BER for Faulty SCMS under BI-AWGN with No Errors Injected in the Two Additional Memories

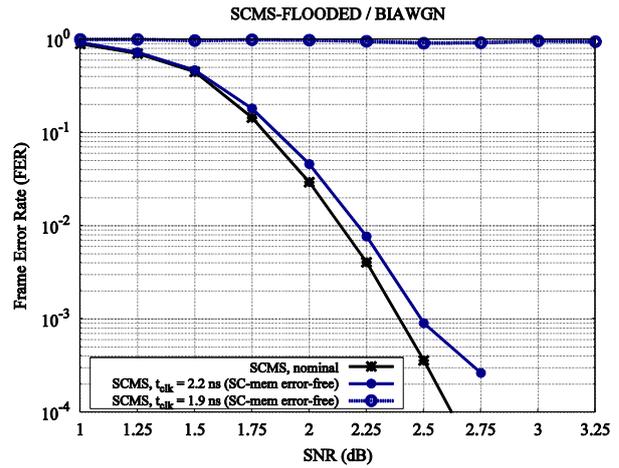


Figure 3-132: FER for Faulty SCMS under BI-AWGN with No Errors Injected in the Two Additional Memories

### 3.6.2.2. Analysis of MS, SCMS, and FAID Decoders under BSC

For MS and SCMS decoders under BSC, the channel scale factor is simply referred to as *channel value* (see discussion from Section 3.6.1). The impact of the channel value on the robustness of the MS decoder has been demonstrated analytically in [i-RISC/D3.1] for theoretical error models. Here, we further highlight this phenomenon for realistic error models, by simulation the MS decoder with channel values 3 and 4. The SCMS decoder is insensitive to the channel value choice hence it is enough to simulate the SCMS decoder with a channel value equal of 3. Note also that the channel value does not apply to the FAID, which operate on binary input data.

Figure 3-133, Figure 3-134, and Figure 3-135 depict the average number of iterations, the BER, and the FER for faulty MS architecture. The results indicate a strong influence of the channel value for the MS decoder. On one hand, a channel value of 4 will lead to a better error correction capability of the decoder with respect to a channel value of 3. On the other hand, for a clock period of 2.2ns, the decoding performance when applying a channel value of 4 is almost the same with the one of an error-free decoder. Applying a channel value of 3, decoding performance degradation can be observed for a clock period of 2.2ns with respect to the error-free decoder. For both values of the channel value, for a clock period of 2.5ns or higher there is no performance loss in decoding, while for a clock period of 1.9ns, the decoders do not decode.

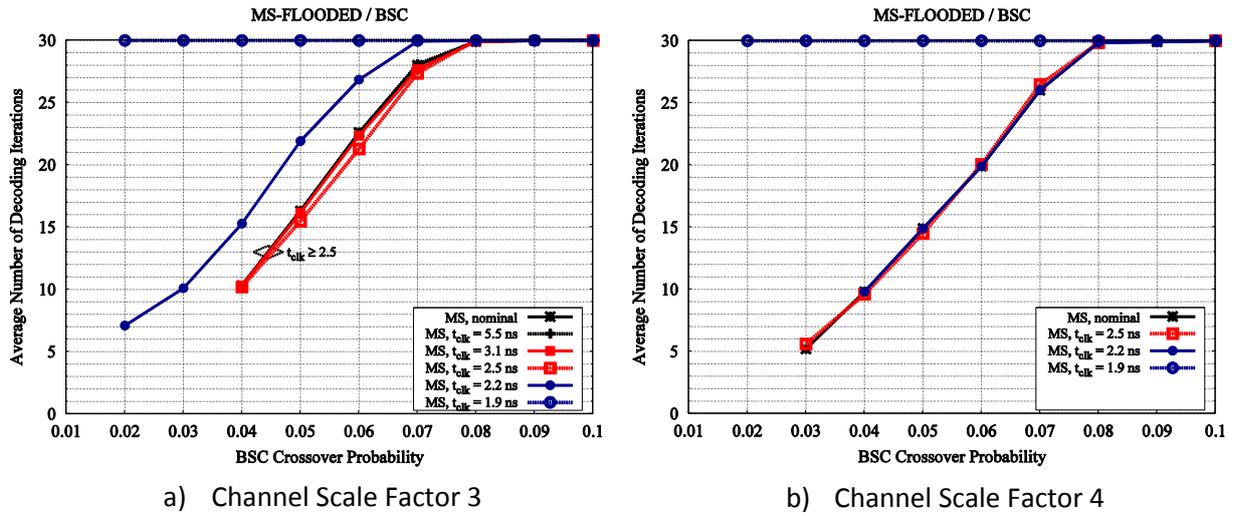


Figure 3-133: Average Number of Iterations for Faulty MS under BSC

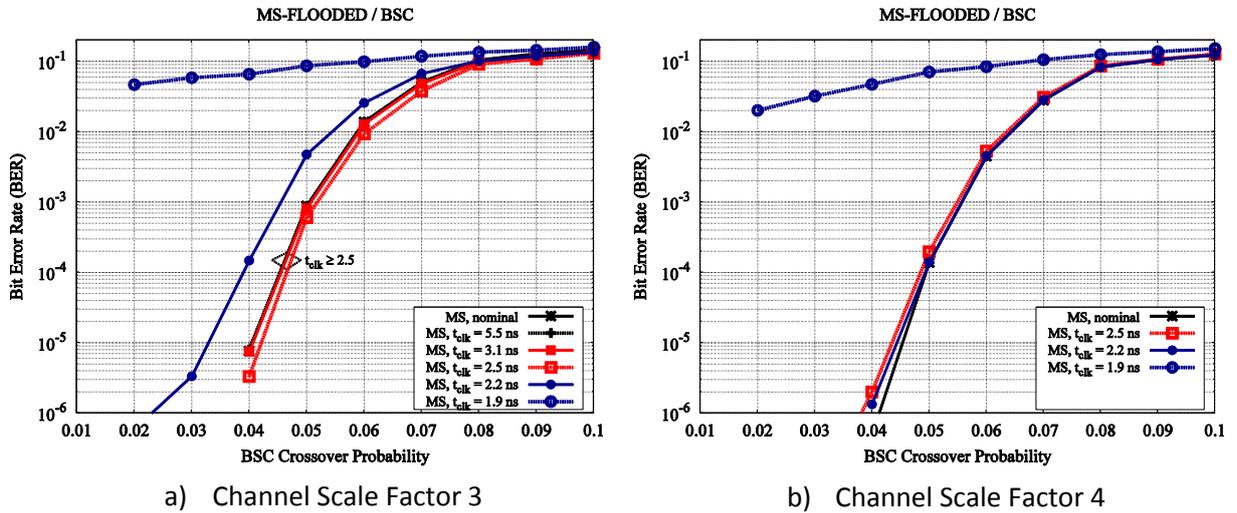


Figure 3-134: BER for Faulty MS under BSC

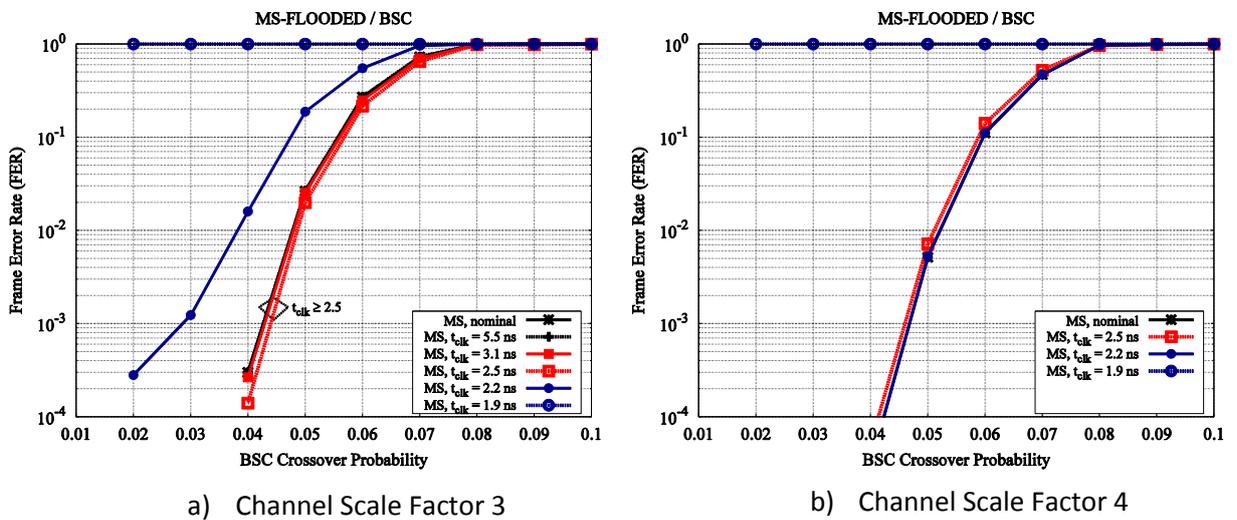


Figure 3-135: FER for Faulty MS under BSC

Figure 3-136, Figure 3-137, Figure 3-138, Figure 3-139, Figure 3-140, and Figure 3-141 depict the average number of iterations, the BER, and the FER for faulty SCMS architecture under BSC with errors injected in the two additional memories (the former 3) and no error injected in the additional memories (the latter 3). A value of 3 has been considered for the channel value. The obtained results are similar to the ones obtained for BI-AWGN. For the SCMS decoder with faulty previous  $\alpha$  sign memory and faulty erasure memory, an error-floor type of behavior can be observed for a clock period of 2.2ns. For the SCMS decoder with error free additional memories, the decoding performance for the clock period of 2.2ns is similar to an error free decoder.

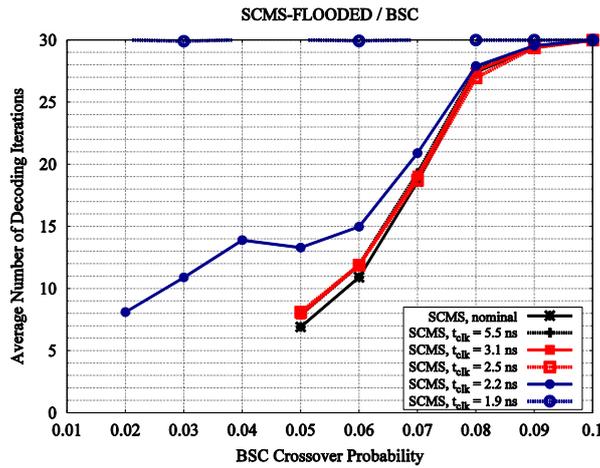


Figure 3-136: Average Number of Iterations for Faulty SCMS under BSC with Errors Injected in All Memories

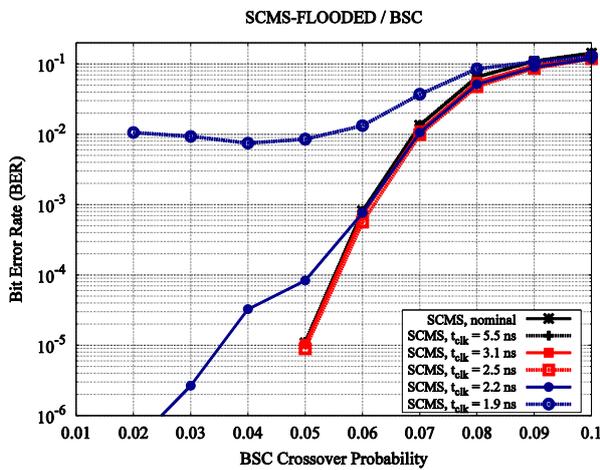


Figure 3-137: BER for Faulty SCMS under BSC with Errors Injected in All Memories

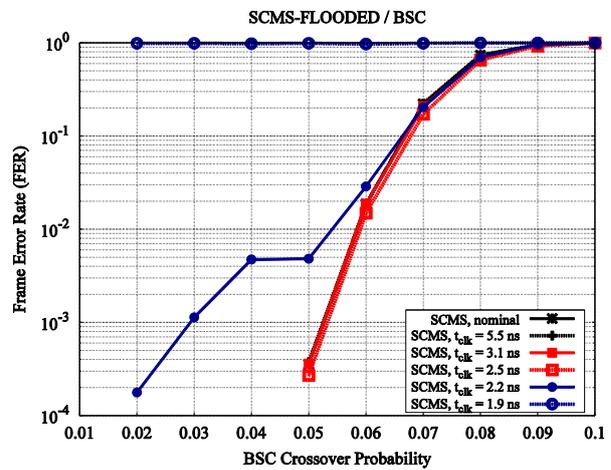


Figure 3-138: FER for Faulty SCMS under BSC with Errors Injected in All Memories

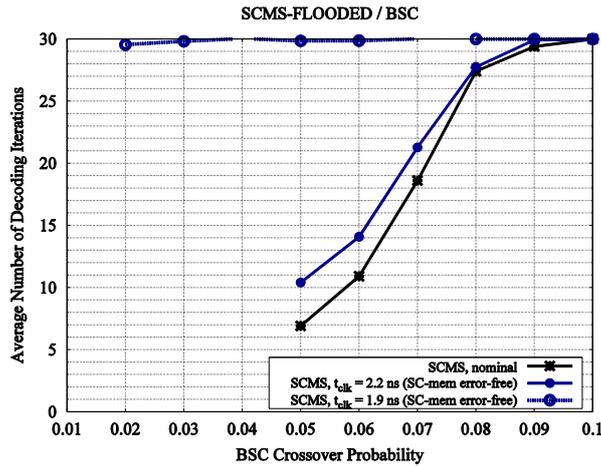


Figure 3-139: Average Number of Iterations for Faulty SCMS under BSC with No Errors Injected in the Two Additional Memories

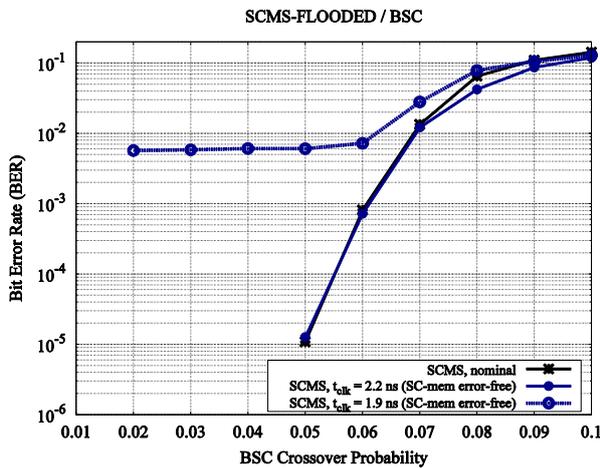


Figure 3-140: BER for Faulty SCMS under BSC with No Errors Injected in the Two Additional Memories

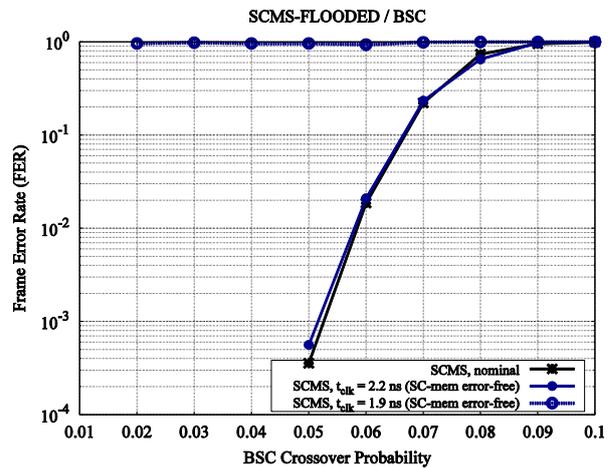


Figure 3-141: FER for Faulty SCMS under BSC with No Errors Injected in the Two Additional Memories

Figure 3-142, Figure 3-143, and Figure 3-144 depict the simulation results for the FAID decoders under BSC channels. The FAID decoder presents no decoding performance degradation for clock periods of 2.2ns or higher with respect to the error free decoder. Slight performance degradation is observed for a clock period of 1.9ns. The FAID decoder cannot decode for clock periods of 1.7ns or lower. With respect to the MS decoders, the FAID decoder can decode for the clock period of 1.9ns. It must be noted that, as indicated in Figure 3-126, the expected number of activated faults in a FAID decoder for the 1.9ns clock period is half compared to the MS and SCMS decoders.

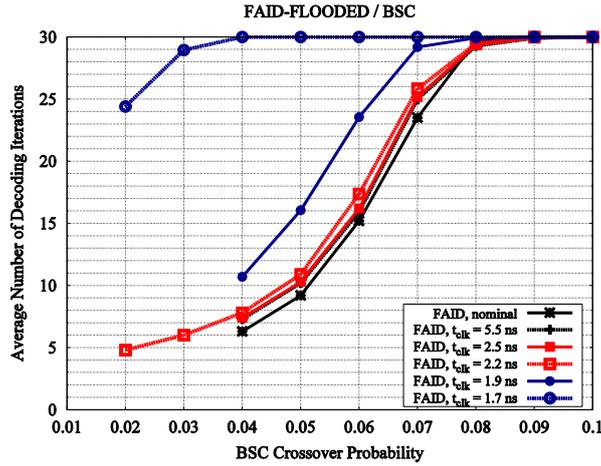


Figure 3-142: Average Number of Iterations for Faulty FAID under BSC

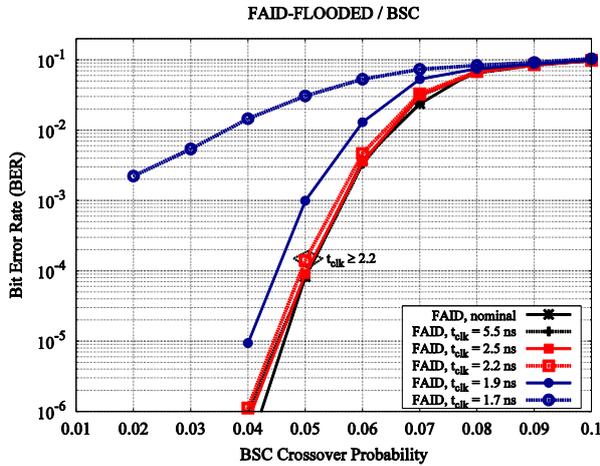


Figure 3-143: BER for Faulty FAID under BSC

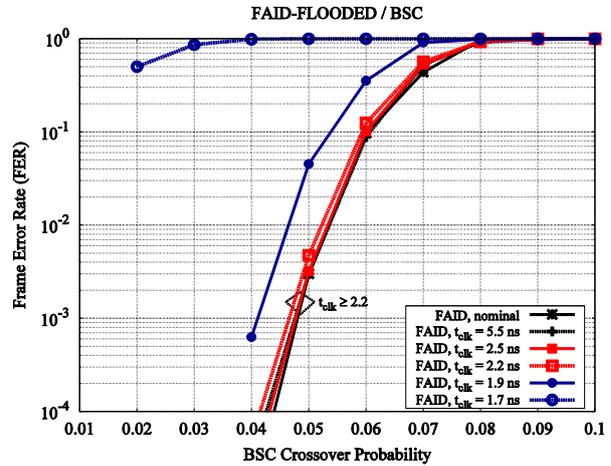


Figure 3-144: FER for Faulty FAID under BSC

### 3.6.2.3. Frequency Scaling Sensitivity

In this section we summarize the evaluation results for the decoders under test, from the FSS metric perspective. In order to capture the way a decoder reacts to the voltage scaling process, the Performance Preservation Region (PPR) and Performance Degradation Region (PDR) have been introduced in Section 3.3. These regions are delimited by the  $T_{pp}$  and  $T_{pd}$  clock period values, which are estimated based on the Monte-Carlo simulation results reported previously:

- $T_{pp}$  = the lowest clock period for which the FER is the same as for nominal  $T_{clk}$
- $T_{pd}$  = the highest clock period for which the FER is flat, equal to 1

With the above notation, it follows that:

$$PPR = T_{dd} - T_{pp} \text{ and } PDR = T_{pp} - T_{pd}$$

PPR measures the decoder potential to increase throughput by means of overclocking, while providing its expected performance. PDR measures about how much overclocking one can still resort to if channel conditions permit. While FSS analysis is very similar to the VSS analysis presented in

Section 3.6.1.8, we note that  $T_{pp}$  and  $T_{pd}$  values are determined based on simulated fault injection and not by actual overclocking of the design, and therefore we have to treat them as such.

Table 3-31 shows the PPR and PDR metrics for the decoders under test. Reported values for  $T_{pp}$  and  $T_{pd}$  have been estimated by investigating the FER performance of each decoder for various clock period values (similar to the methodology described in Section 3.6.1.8). For MS and SCMS decoders, we may observe that the obtained  $T_{pp}$  values are slightly smaller for the BSC than the BI-AWGN, which may indicate an increased robustness under BSC. The smaller  $T_{pp}$  value is obtained for SCMS(\*) and FAID decoders under BSC, and indicates a frequency (hence, throughput) increase by a factor of 2.5, without any degradation of the error correction performance. Note that reducing the clock period from nominal  $T_{clk}$  to  $T_{pp}$  corresponds to a throughput increase between 77% (for  $T_{pp} = 3.1ns$ ) and 150% (for  $T_{pp} = 2.2ns$ ).

Table 3-31: PPR and PDR Metrics for the Decoders under Test (SFI Scenario)

	BI-AWGN		BSC				
	MS(c=3.5)	SCMS(c=3.5)	MS(c=3)	MS(c=4)	SCMS(c=3)	SCMS(*)	FAID
Tclk	5,5	5,5	5,5	5,5	5,5	5,5	5,5
Tpp	3,1	2,5	2,5	2,2	2,5	2,2	2,2
Tpd	1,9	1,9	1,9	1,9	1,9	1,9	1,7
PPR	2,4	3	3	3,3	3	3,3	3,3
PDR	1,2	0,6	0,6	0,3	0,6	0,3	0,5

(c=x) Channel scale factor or channel value used for BI-AWGN and BSC, respectively

(\*) No errors injected in the two additional memories (SCMS specific)

### 3.7. Conclusion

A number of seven LDPC decoders have been implemented and evaluated under either voltage scaling or fault injection scenario. Among them, four are hard-decision decoders (GB, GDBF, PGDBF, and FAID) and only apply to the BSC channel model. The remaining three decoders (SD, MS, and SCMS) are soft-decision decoders, and apply to both Bi-AWGN and BSC channel models. Furthermore, three other MS, SCMS and FAID variants without Early Termination (ET) circuit have been implemented, thus bringing the total number of evaluated decoders to ten.

Under voltage scaling scenario, the ten decoders have been evaluated for the BSC model. Evaluation results have been reported in terms of error correction performance (BER and FER), average number of decoding iterations, throughput, and energy/bit. It has been demonstrated that all decoders are able to preserve their nominal error correction performance when supply voltage is scaled down to some critical point, from which the error correction performance starts degrading. Informed by the evaluation results in terms of BER, FER or average number of decoding iterations, we revealed two main phenomena responsible for the error correction performance degradation. For SD, PGDBF, FAID, and FAIDnoET decoders, the degradation of the error correction performance is most likely due to timing errors affecting the decoder data processing units. For GB, GDBF, MS, SCMS, and SCMSnoET decoders the error correction performance lost is primarily due to control unit, which fails to work properly at low voltage supply. A particular case is represented by the MSnoET decoder, which

preserves its nominal error correction performance until the control unit completely stops functioning.

Concerning the ET circuit, we showed that it helps improving the error correction performance, in case that the control unit works properly. This phenomenon has been highlighted for the FAID decoder. However, the way the ET circuit is integrated to the MS and SCMS decoders seems to be responsible for their control units' failures, which in turn results in a decreased robustness of the MS and SCMS decoders with respect to their no-ET counterparts.

In order to get an overall view of how different decoders react to the voltage scaling process, we further introduced the Voltage Scale Sensitivity (VSS) metric and summarized the evaluation results from its perspective. As a result, we measured the decoders potential to save energy while providing their expected performance. We showed that voltage scaling may results in energy savings between 45% and 67%, while preserving the nominal throughput and error correction performance.

Given that voltage scaling creates timing faults all over the circuit it only partially reflect the real life decoder exposure to environmental aggression factors. In such situations fault occurrence location and rate (fault occurrence map) also relate to decoder architecture and implementation details. To this end we introduced a fault map creation methodology and performed fault map guided Simulation Fault Injection. In this way we selectively exposed decoder data path parts to errors and observed MS, SCMS, and FAID behavior in this conditions. A fault free control has been considered for the experiment. The three decoders have been evaluated in terms of BER, FER, and average number of decoding iterations, under either the BSC (all decoders) or the BI-AWGN (MS and SCMS, only) channel model. The primary conclusion of the SFI experiments is that all the decoders have the ability to also correct errors in their data-path, not only those which appear in the transmission channel. For all three decoders, increasing the clock frequency by a factor of 2 with respect to the maximum supported by the error free decoder will lead to no error correction capability degradation.

For BSC channel, we showed that the channel value has a strong influence on the MS decoder performance: a channel value of 4 leads to better decoding performance and better fault tolerance with respect to channel value of 3.

For both BSC and BI-AWGN channels, the errors in the two SCMS-specific memories (previous  $\alpha$  sign and erasure bit memories) have a strong influence on the SCMS performance. For a clock period of 2.2ns (corresponding to a  $10^{-4}$  error rate in memories), the SCMS decoder presents a high error floor when the two memories are affected by faults. When the two SCMS-specific memories are faulty free, similar performance as the error free decoder has been observed.

The simulations also indicate that the FAID decoder can support lower clock periods than the MS/SCMS decoders. However, it must be taken into consideration that the FAID decoder has a lower number of expected activated fault locations with respect to the MS based decoders.

Finally, we introduced the Frequency Scale Sensitivity (FSS) metric and summarized the evaluation results from its perspective. As a result, based on simulated fault injection results, the decoder potential to increase throughput by means of overclocking has been estimated to be between 77% and 150%, while preserving the nominal error correction performance.

## 4. General Conclusions and Next Steps

A novel integrated CAD flow targeting reliability heavy-multi objective logic synthesis has been proposed in the first part of this Deliverable. We showed that a heterogeneous flow (using both academic and industry tools) was necessary, including custom scripts and wrappers developed on top of existing tools in order to facilitate the reliability analysis and optimization flow. To start with, a detailed study of all the available set of data structures, their benefits and limitations, and the associated open source EDA tools was done. We have developed multiple set of reliability computation engines that would estimate the probable output error with varying degrees of accuracies and speed. Multiple set of logic optimization techniques have been developed which predominantly work at gate level. Further, we have developed logic augmentation techniques to improve the circuit fault tolerance. Going forward, this flow will be used to (i) improve upon the initial work of the reliability computation techniques and the graph optimization algorithms, (ii) optimize the circuits for improving reliability within a multi-objective optimization framework, and (iii) validate and to characterize the circuits proposed as part of proof of concept within i-RISC.

In the second part of this Deliverable, reliable systems in the context of LDPC decoders built out of unreliable components have been evaluated. A number of seven LDPC decoders (Stochastic, Gallager-B, GDBF, PGDBF, MS, SCMS, and FAID) have been implemented in VHDL/Verilog and exposed to external aggression via voltage scaling or simulated fault injection. To have the most realistic results, a real experimental hardware platform has been developed to evaluate the LDPC decoders implemented on a Xilinx Virtex-7 FPGA, providing voltage scaling support and enabling power/energy measurement. For the fault injection simulation scheme, the error profile of the basic building blocks of the LDPC decoders under investigation has first been characterized, and then used as guidance mean for judiciously performing the fault injection.

The implemented decoders have been evaluated for different voltage scaling of fault injection aggression profiles in terms of (i) error correction performance, specifically FER and BER, (ii) average number of iterations, (iii) throughput (Mb/s), and (iv) energy/bit (pJ/bit). We further introduced the Voltage Scaling Sensitivity and the Frequency Scaling Sensitivity metrics, so that to capture their reaction to voltage and frequency scaling, and to determine their potential to save energy or to increase throughput, while delivering their expected error correction performance. As a main contribution of this Deliverable, the conducted experiments substantiated the fault-tolerance capabilities of the LDPC decoders under test and the resulting benefits in terms of energy consumption and throughput.

Having provided the implementation and the assessment of the fault-tolerance capabilities of a large number of LDPC decoders, the first WP6 milestone (MS7 – Proof of concept: implementation and validation of the LDPC decoder) has been reached. Going forward, the results obtained in this Deliverable will be used to (i) assess the fault-tolerance capabilities and (ii) evaluate the energy consumption and throughput figures of merit of the error-correction driven logic augmentation techniques proposed in Work-Package 5 of i-RISC.

## References

- [Aymerich12] N. Aymerich, S.D. Cotofana, and A. Rubio, "Degradation Stochastic Resonance (DSR) in Adaptive-Averaging (AD-AVG) Architectures," *12th IEEE Conference on Nanotechnology (IEEE-NANO)*, pp. 1–4, August 2012.
- [Borkar05] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.
- [Brayton10] R. Brayton and A. Mishchenko, "Abc: An Academic Industrial-Strength Verification Tool," *22nd International Conference on Computer Aided Verification*, pp. 24–40, 2010.
- [Cattell95] K. Cattell, S. Zhang, "Minimal Cost One-Dimensional Linear Hybrid Cellular Automata of Degree Through 500", *Journal of Electronic Testing: Theory and Applications 6*, pp. 255-258, 1995.
- [Chandrasetty12] V.A. Chandrasetty, S.M. Aziz, "An Area Efficient LDPC Decoder Using a Reduced Complexity Min-Sum Algorithm", *VLSI Journal on Integration*, vol. 45, no. 2, pp. 141-148, 2012.
- [Chen11] J. Chen, J. Kang, S. Lin, V. Akella, "Memory System Optimization for FPGA Based Implementation of Quasi-Cyclic LDPC Codes Decoders", *IEEE Transactions on Circuits and Systems I*, vol. 59, no. 1, pp. 98-111, 2011.
- [Chen14] J. Chen, C. Spagnol, S. Grandhi, E. Popovici, S.D. Cotofana, and A. Amaricai, "Linear Compositional Delay Model for the Timing Analysis of Sub-Powered Combinational Circuits," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 380-385, 9-11 July 2014.
- [Choudhury09] M.R. Choudhury, and K. Mohanram, "Reliability Analysis of Logic Circuits." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 3, pp. 392–405, 2009.
- [Constantinescu03] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.
- [Darabiha08] A. Darabiha, A.C. Carusone, F.R. Kschischang, "Block-Interlaced LDPC Decoders With Reduced Interconnect Complexity," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 1, pp. 74–78, January 2008.
- [Dodd03] P. E. Dodd and L. W. Massengill, "Basic Mechanisms and Modelling of Single-Event Upset in Digital Microelectronics," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 583–602, June 2003.
- [Gaines69] B. Gaines, "Advances in Information Systems Science", New York: Plenum, 1969.
- [Gallager62] R. G. Gallager, "Low-Density Parity-Check Codes", *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, 1962.
- [Gammaitoni98] L. Gammaitoni, P. Hanggi, P. Jung, and F. Marchesoni, "Stochastic Resonance" *Reviews of Modern Physics*, vol. 70, pp. 223–287, January 1998.
- [Hu05] X.-Y. Hu, E. Eleftheriou and D.M. Arnold, "Regular and Irregular Progressive Edge-Growth Tanner Graphs", *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386-398, January 2005.
- [i-RISC/D2.1] FP7-ICT/FET-OPEN/ i-RISC project, Deliverable 2.1 "Circuit Level Fault Models for Sub-Powered CMOS Circuits for Uncorrelated and Correlated Errors"  
Online: [http://www.i-risc.eu/home/liblocal/docs/iRISC\\_Deliverables/i-RISC\\_D2.1.pdf](http://www.i-risc.eu/home/liblocal/docs/iRISC_Deliverables/i-RISC_D2.1.pdf)
- [i-RISC/D2.2] FP7-ICT/FET-OPEN/ i-RISC project, Deliverable 2.2 "Higher Abstraction Fault Models and Their Simulation Methodology"  
Online: [http://www.i-risc.eu/home/liblocal/docs/iRISC\\_Deliverables/i-RISC\\_D2.2.pdf](http://www.i-risc.eu/home/liblocal/docs/iRISC_Deliverables/i-RISC_D2.2.pdf)
- [i-RISC/D3.1] FP7-ICT/FET-OPEN/ i-RISC project, Deliverable 3.1 "Fault Tolerant LDPC Encoding and Decoding"  
Online: [http://www.i-risc.eu/home/liblocal/docs/iRISC\\_Deliverables/i-RISC\\_D3.1.pdf](http://www.i-risc.eu/home/liblocal/docs/iRISC_Deliverables/i-RISC_D3.1.pdf)
- [i-RISC/D3.2] FP7-ICT/FET-OPEN/ i-RISC project, Deliverable 3.1 "Fault Tolerant LDPC Encoding and Decoding"

Online: [http://www.i-risc.eu/home/liblocal/docs/iRISC\\_Deliverables/i-RISC\\_D3.2.pdf](http://www.i-risc.eu/home/liblocal/docs/iRISC_Deliverables/i-RISC_D3.2.pdf)

**[i-RISC/D5.1]** FP7-ICT/FET-OPEN/ i-RISC project, Deliverable 5.1, “Data Structures and Design Flow for Fault Tolerant Circuit Synthesis”, January 2014.

Online: [http://www.i-risc.eu/home/liblocal/docs/iRISC\\_Deliverables/i-RISC\\_D5.1.pdf](http://www.i-risc.eu/home/liblocal/docs/iRISC_Deliverables/i-RISC_D5.1.pdf)

**[i-RISC/D5.2]** FP7-ICT/FET-OPEN/ i-RISC project, Deliverable 5.2, “Fault Tolerant Synthesis through Graph Augmentation, Multi Objective Optimization and Boole Shannon limit”, January 2015.

Online: [http://www.i-risc.eu/home/liblocal/docs/iRISC\\_Deliverables/i-RISC\\_D5.2.pdf](http://www.i-risc.eu/home/liblocal/docs/iRISC_Deliverables/i-RISC_D5.2.pdf)

**[Kaeslin08]** H. Kaeslin, “Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication”, 1st ed. New York, NY, USA: Cambridge University Press, 2008.

**[Le15]** K. Le, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis, B. Vasic, “Efficient Realization of Probabilistic Gradient Descent Bit Flipping Decoders”, *IEEE International Symposium on Circuits and Systems (ISCAS)*, 24-27 May 2015.

**[Liva06]** G. Liva, S. Song, L. Lan, Y. Zhang, S. Lin, W. Ryan, “Design of LDPC Codes: A Survey and New Results,” *Journal of Communications Software and Systems*, 2006.

**[Marconi14]** T. Marconi, C. Spagnol, E. Popovici and S.D. Cotofana, “Towards Energy Effective LDPC Decoding by Exploiting Channel Noise Variability”, *22nd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 2014)*, pp. 1-6, October 2014.

**[Marconi15]** T. Marconi and S.D. Cotofana, “Dynamic Bitstream Length Scaling Energy Effective Stochastic LDPC Decoding”, *25th edition of ACM's Great Lakes VLSI Symposium (GLSVLSI)*, Pittsburgh, Pennsylvania, USA, May 2015.

**[Mehrotra11]** R. Mehrotra, T. English, M. Schellekens, S. Hollands, and E. Popovici, “Timing-Driven Power Optimisation and Power-Driven Timing Optimisation of Combinational Circuits,” *Journal of Low Power Electronics*, vol. 7, no. 3, pp. 364–380, 2011.

**[Park14]** Y.S. Park, D. Blaauw, D. Sylvester, and Z. Zhang, “Low-Power High-Throughput LDPC Decoder Using Non-Refresh Embedded DRAM” *IEEE Journal of Solid State Circuits*, vol. 49, no. 3, pp. 783-794, 2014

**[Pedram96]** S. Iman and M. Pedram, “Pose: Power Optimization and Synthesis Environment,” *33rd Design Automation Conference*, pp. 21–26, 1996.

**[Planjery10]** S. Planjery, D. Declercq, S. Chilappagari, B. Vasic, “Multilevel Decoders Surpassing Belief Propagation on the Binary Symmetric Channel”, *IEEE International Symposium On Information Theory (ISIT)*, pp. 769-773, 2010.

**[Poppelbaum67]** W.J. Poppelbaum, C. Afuso, and J.W. Esch., “Stochastic Computing Elements and Systems”, *ACM Fall Joint Computer Conference, AFIPS '67 (Fall)*, pages 635–644, New York, NY, USA, November 14-16, 1967.

**[Rasheed14]** O.A. Rasheed, P. Ivanis and B. Vasic, “Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder”, *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, September 2014.

**[Ribeiro67]** S.T. Ribeiro, “Random-Pulse Machines. Electronic Computers”, *IEEE Transactions on Electronic Computers*, vol. 16, no.3, pp. 261–276, June 1967.

**[Savin08]** V. Savin, “Self-Corrected Min-Sum Decoding of LDPC Codes”, *IEEE International Symposium On Information Theory (ISIT)*, pp. 146-150, 2008.

**[Savin14]** V. Savin, “LDPC decoders”, in Channel coding: Theory, algorithms, and applications, D. Declercq, M. Fossorier, and E. Biglieri editors, Academic Press Library in Mobile and Wireless Communications, Elsevier, June 2014.

**[SDF04]** IEEE. Delay and power calculation standards - Part 3: Standard Delay Format (SDF) for the electronic design process (IEEE Std 1497-2004). IEEE, New York, 2004.

**[Sentovich92]** E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis", Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

**[Stimming12]** A.B. Stimming, A. Dollas, "FPGA-Based Design and Implementation of a Multi - GBPS LDPC Decoder", *22nd Field Programmable Logic and Applications (FPL)*, pp. 262-269, 2012.

**[SYNSAIF]** Synopsys SAIF. Switching Activity Interchange Format.

Online: <http://www.synopsys.com/partners/tapin/saif.html>.

**[SYNTOOL]** Synopsys. Synopsys design platforms. <http://www.synopsys.com/products/products.html>.

**[Tanner01]** M. Tanner, D. Srkdhara, and T. Fuja, "A Class of Group-Structured LDPC Codes," 2001 Online: [citeseer.ist.psu.edu/tanner01class.html](http://citeseer.ist.psu.edu/tanner01class.html).

**[Tehrani06]** S.S. Tehrani, W.J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes", *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.

**[Thorpe03]** J. Thorpe, "Low-Density Parity-Check (LDPC) Codes Constructed from Protographs," *Jet Propulsion Laboratory, InterPlanetary Network*, Technical Report, August 2003.

**[Venkiah08]** A. Venkiah, D. Declercq and C. Poulliat, "Design of Cages with a Randomized Progressive Edge Growth Algorithm", *IEEE Communications Letters*, vol. 12(4), pp. 301-303, April 2008.

**[Vittoz14]** E. A. Vittoz, "Low-Power CMOS Circuits", CRC Press, Nov 2005, chapter "Weak Inversion for Ultimate Low-Power Logic", pp. 1–18, 2005.

**[Wadayama10]** T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami and I. Takumi, "Gradient Descent Bit Flipping Algorithms for Decoding LDPC Codes", *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1610–1614, June 2010.

**[Wu05]** D. Wu and J. Zhu, "FBDD: A Folded Logic Synthesis System", *International Conference on ASIC (ASICON)*, pp. 746–751, October 2005.

**[XilinxTRNG]** C. Baetoni "High Speed True Random Number Generators in Xilinx FPGAs"

Online:

<http://forums.xilinx.com/xlnx/attachments/xlnx/EDK/27322/1/HighSpeedTrueRandomNumberGeneratorsinXilinxFPGAs.pdf>

**[Yanushkevich05]** S. N. Yanushkevich, D. M. Miller, V. P. Shmerko and R. S. Stankovic, "Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook", CRS Press, pp. 429–445, 2006.

## Appendix 1: Quasi-Cyclic Description of the LDPC Codes

In this appendix, we describe the organization of the parity-check matrices for the LDPC codes utilized in the project. For each non-zero entry in the protograph, the integer value indicated below corresponds to the index of the first “1” in the circulant matrix. All the other “1”s of the circulant block are deduced by circular shift, modulo L. A “-1” integer value indicates the zero blocks of the protograph. Only the regular LDPC matrices are shown in this appendix.

### LDPC Code : R=1/2, d<sub>v</sub>=3, N=1296, L=54

```

49 -1 -1 -1 -1 43 -1 -1 -1 -1 50 -1 -1 -1 -1 2 -1 27 -1 -1 -1 -1 49
-1 -1 -1 10 41 -1 -1 -1 -1 52 -1 -1 32 -1 -1 -1 -1 50 -1 50 -1 -1 -1
-1 -1 20 -1 -1 -1 -1 20 -1 -1 -1 51 -1 10 -1 -1 47 -1 -1 -1 -1 33 -1
-1 24 -1 -1 -1 -1 22 -1 53 -1 -1 -1 -1 31 -1 -1 -1 -1 18 -1 47 -1 -1
10 -1 -1 -1 15 -1 -1 -1 -1 2 -1 -1 -1 -1 50 -1 13 -1 -1 -1 -1 53
-1 -1 44 -1 -1 6 -1 -1 -1 -1 -1 29 -1 40 -1 -1 16 -1 -1 13 -1 -1 -1
-1 2 -1 -1 -1 -1 -1 13 41 -1 -1 -1 -1 42 -1 -1 -1 -1 48 -1 49 -1 -1
-1 -1 -1 36 -1 -1 24 -1 -1 50 -1 -1 12 -1 -1 -1 -1 10 -1 -1 -1 48 -1
-1 -1 47 -1 50 -1 -1 -1 -1 0 -1 -1 -1 -1 9 -1 7 -1 -1 -1 -1 28
-1 24 -1 -1 -1 -1 -1 51 -1 38 -1 -1 -1 -1 6 -1 -1 -1 -1 23 -1 16 -1 -1
6 -1 -1 -1 -1 -1 5 -1 -1 -1 -1 13 -1 3 -1 -1 29 -1 -1 -1 16 -1 -1 -1
-1 -1 -1 35 -1 16 -1 -1 37 -1 -1 -1 4 -1 -1 -1 -1 -1 24 -1 -1 -1 29 -1
    
```

### LDPC Code : R=1/2, d<sub>v</sub>=4, N=1296, L=54

```

11 -1 -1 -1 27 -1 -1 -1 33 16 -1 -1 -1 44 -1 -1 44 -1 8 -1 -1 -1 -1 0
-1 25 -1 -1 -1 31 29 -1 -1 -1 29 -1 -1 -1 36 -1 -1 34 -1 15 -1 -1 17 -1
-1 -1 44 4 -1 -1 -1 11 -1 -1 -1 2 50 -1 -1 52 -1 -1 -1 -1 30 33 -1 -1
27 -1 -1 -1 34 -1 20 -1 -1 20 -1 -1 -1 13 -1 -1 27 -1 4 -1 -1 -1 -1 27
-1 42 -1 22 -1 -1 -1 11 -1 -1 -1 44 -1 -1 4 14 -1 -1 -1 45 17 -1 -1
-1 -1 24 -1 -1 10 -1 -1 10 -1 18 -1 2 -1 -1 -1 19 -1 38 -1 -1 31 -1
-1 -1 40 -1 -1 35 -1 -1 31 19 -1 -1 3 -1 -1 42 -1 -1 -1 42 -1 -1 39 -1
-1 29 -1 0 -1 -1 -1 29 -1 -1 5 -1 -1 -1 47 -1 -1 28 -1 -1 28 41 -1 -1
9 -1 -1 -1 7 -1 20 -1 -1 -1 -1 1 -1 19 -1 -1 5 -1 25 -1 -1 -1 -1 41
-1 -1 53 -1 -1 3 -1 -1 26 -1 3 -1 -1 -1 30 -1 -1 5 -1 35 -1 -1 44 -1
-1 4 -1 -1 4 -1 -1 5 -1 -1 -1 13 42 -1 -1 50 -1 -1 -1 -1 36 38 -1 -1
39 -1 -1 17 -1 -1 36 -1 -1 34 -1 -1 -1 46 -1 -1 12 -1 8 -1 -1 -1 -1 15
    
```

### LDPC Code : R=3/4, d<sub>v</sub>=3, N=1296, L=27

```

-1 -1 -1 9 9 -1 -1 -1 -1 6 -1 -1 -1 -1 8 15 -1 -1 -1 -1 18 -1 -1 20 -1 -1 -1 -1 18 -1 -1 12 -1 -1 18 -1 -1 -1 -1 -1 -1 1 19 -1 -1 -1
7 -1 -1 -1 -1 -1 -1 -1 2 -1 -1 9 -1 -1 -1 -1 -1 17 -1 -1 19 -1 -1 -1 26 -1 13 -1 -1 -1 1 -1 -1 -1 -1 17 -1 -1 7 -1 -1 -1 17 -1 -1
-1 10 -1 -1 -1 -1 -1 25 -1 -1 -1 4 -1 17 -1 -1 -1 16 -1 -1 14 -1 -1 -1 15 -1 -1 -1 -1 -1 22 -1 -1 -1 1 -1 21 -1 -1 -1 11 -1 -1 -1 8
-1 -1 22 -1 -1 -1 10 -1 24 -1 -1 -1 -1 22 -1 -1 -1 5 -1 -1 -1 -1 15 -1 -1 -1 7 -1 14 -1 -1 -1 -1 24 -1 -1 -1 9 -1 -1 -1 -1 -1 5 -1
19 -1 -1 -1 -1 -1 25 -1 -1 9 -1 17 -1 -1 -1 -1 -1 8 17 -1 -1 -1 24 -1 -1 18 -1 -1 -1 -1 -1 9 -1 -1 11 -1 -1 8 -1 -1 -1 -1 -1 2
-1 17 -1 -1 9 -1 -1 -1 2 -1 -1 -1 -1 25 12 -1 -1 -1 -1 25 -1 -1 -1 12 -1 -1 -1 -1 15 -1 18 -1 -1 23 -1 -1 -1 -1 -1 1 -1 -1 22 -1 -1
-1 -1 10 -1 21 -1 -1 19 -1 -1 -1 -1 2 -1 -1 -1 3 -1 -1 -1 18 -1 -1 5 -1 -1 -1 -1 26 -1 -1 -1 15 -1 -1 0 -1 -1 -1 -1 -1 24 7 -1 -1 -1
-1 -1 11 -1 -1 -1 20 -1 -1 -1 -1 24 -1 -1 23 -1 -1 -1 22 -1 -1 -1 0 -1 -1 -1 22 -1 -1 0 -1 9 -1 -1 -1 -1 -1 5 20 -1 -1 -1 -1 26 -1
10 -1 -1 -1 20 -1 -1 -1 -1 9 -1 -1 -1 -1 6 -1 -1 -1 19 -1 -1 7 -1 -1 -1 3 -1 6 -1 -1 -1 -1 -1 23 -1 -1 15 -1 -1 24 -1 -1 -1 17 -1 -1
-1 5 -1 -1 -1 -1 15 -1 22 -1 -1 21 -1 -1 -1 6 -1 -1 -1 23 -1 -1 -1 7 -1 -1 -1 -1 -1 19 -1 18 -1 -1 26 -1 -1 -1 5 -1 -1 -1 -1 -1 4
-1 -1 4 -1 -1 -1 16 -1 24 -1 -1 -1 -1 24 -1 -1 -1 4 -1 -1 -1 22 -1 -1 -1 4 -1 11 -1 -1 -1 -1 -1 13 -1 -1 1 -1 -1 -1 6 -1
-1 -1 -1 1 -1 20 -1 -1 -1 -1 11 -1 5 -1 -1 -1 1 -1 -1 -1 7 -1 -1 -1 14 -1 -1 -1 20 -1 -1 -1 -1 5 -1 -1 8 -1 -1 -1 -1 16 19 -1 -1 -1
    
```

### LDPC Code : R=3/4, d<sub>v</sub>=4, N=1296, L=27

```

-1 -1 0 -1 12 -1 -1 -1 0 22 -1 -1 1 -1 -1 -1 11 -1 23 -1 -1 -1 -1 21 1 -1 -1 -1 17 -1 -1 24 -1 0 -1 -1 -1 2 -1 18 -1 11 -1 -1 -1 19 -1
-1 14 -1 23 -1 -1 10 -1 -1 -1 26 -1 -1 -1 20 -1 -1 19 -1 -1 25 -1 21 -1 -1 19 -1 3 -1 -1 24 -1 -1 -1 -1 20 -1 26 -1 -1 -1 20 -1 -1 -1 11 -1 -1 14
16 -1 -1 -1 -1 25 -1 26 -1 -1 -1 7 -1 9 -1 18 -1 -1 -1 24 -1 19 -1 -1 -1 25 -1 -1 13 -1 -1 11 -1 22 -1 18 -1 -1 13 -1 -1 -1 24 -1 9 -1 -1
-1 -1 16 -1 25 -1 17 -1 -1 -1 14 -1 16 -1 -1 -1 1 4 -1 -1 12 -1 6 -1 -1 21 -1 -1 26 -1 -1 -1 9 -1 9 -1 -1 11 -1 -1 -1 3 -1 18 -1
-1 23 -1 -1 -1 15 -1 -1 6 25 -1 -1 -1 23 1 -1 -1 -1 25 -1 -1 -1 20 -1 -1 9 0 -1 -1 23 -1 -1 20 -1 -1 -1 13 -1 -1 3 25 -1 -1 -1 18
17 -1 -1 14 -1 -1 -1 18 -1 -1 -1 22 -1 18 -1 -1 25 -1 -1 -1 15 21 -1 -1 -1 6 -1 -1 -1 24 -1 -1 10 -1 5 -1 23 -1 -1 17 -1 -1 -1 3 -1 7 -1 -1
-1 3 -1 -1 -1 9 3 -1 -1 -1 23 -1 -1 -1 0 -1 -1 17 -1 -1 10 22 -1 -1 -1 0 24 -1 -1 -1 16 -1 -1 20 -1 22 -1 -1 3 -1 -1 -1 13 -1 -1 -1 24
11 -1 -1 2 -1 -1 -1 0 -1 -1 -1 20 -1 26 -1 9 -1 -1 26 -1 -1 -1 23 -1 -1 11 -1 -1 -1 15 -1 -1 16 23 -1 -1 -1 2 -1 -1 24 -1 -1 -1 21 8 -1 -1
-1 -1 21 -1 24 -1 -1 -1 18 15 -1 -1 18 -1 -1 -1 6 -1 -1 20 -1 -1 -1 23 17 -1 -1 -1 14 -1 16 -1 -1 -1 3 -1 -1 13 -1 -1 17 14 -1 -1 -1 24 -1
1 -1 -1 13 -1 -1 6 -1 -1 -1 -1 7 -1 15 -1 -1 -1 12 18 -1 -1 -1 4 -1 -1 -1 15 -1 -1 1 -1 22 -1 8 -1 -1 15 -1 -1 -1 24 -1 -1 -1 18 17 -1 -1
-1 17 -1 -1 -1 19 -1 24 -1 -1 23 -1 -1 -1 11 2 -1 -1 -1 -1 9 25 -1 -1 -1 5 -1 9 -1 -1 -1 18 -1 22 -1 -1 -1 8 10 -1 -1 -1 14 -1 -1 -1 16
-1 -1 14 -1 18 -1 -1 -1 20 4 -1 -1 13 -1 -1 -1 19 -1 -1 0 -1 -1 -1 24 22 -1 -1 -1 24 -1 22 -1 -1 -1 11 -1 23 -1 -1 -1 15 13 -1 -1 -1 8 -1
    
```