UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

Electronics and Computer Science

Southampton Wireless

Timing-Error-Tolerant Iterative Decoders

by

Isaac Pérez Andrade

A thesis submitted in partial fulfillment for the degree of
Doctor of Philosophy
at the University of Southampton

May 2016

Supervisors:

Dr. Robert G. Maunder

Prof. Bashir M. Al-Hashimi

Prof. Lajos Hanzo

Dedicated to my family and friends

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science
Southampton Wireless

Doctor of Philosophy

TIMING-ERROR-TOLERANT ITERATIVE DECODERS

by Isaac Pérez Andrade

Iterative decoders such as Low-Density Parity-Check (LDPC) and turbo decoders have an *inherent* capability to correct the transmission errors that originate during communication over a hostile wireless channel. This capability has engendered the widespread use of LDPC and turbo decoders in current communications standards. As a result, significant research efforts have been made in order to conceive efficient Very-Large-Scale Integration (VLSI) implementations of both LDPC and turbo decoders. Typically, these efforts have focused on optimizing only one of the various trade-offs associated with the hardware implementation of iterative decoders, such as the chip area, latency, throughput, energy efficiency or Bit Error Ratio (BER) performance. However, tolerance to timing errors that occur during the iterative decoding processing are typically not considered in these implementations. Owing to this, the BER performance and hardware efficiency of the proposed designs may be severely degraded, if timing errors occur during the iterative decoding process.

Against this background, this thesis demonstrates that iterative decoders are capable of exploiting their *inherent* error correction capability to correct not only *transmission* errors, but also *timing* errors caused by overclocking and power supply variations. Moreover, we propose modifications to the iterative decoders designs, which further enhance their inherent tolerance to timing errors. We achieve this by considering the close relationship between the different trade-offs associated with the hardware implementation of iterative decoders, with the aim of achieving Pareto optimality, where none of these trade-offs can be further improved without degrading at least one of the others. Owing to this, our proposed timing-error-tolerant design methodology simultaneously considers the design constraints and parameters that affect not only the BER performance, but also the hardware efficiency of each implementation.

We first investigate the benefits of stochastic computing in iterative decoders, by characterizing the inherent timing-error tolerance of Stochastic LDPC Decoders (SLDPCDs) and Stochastic Turbo Decoders (STDs). Moreover, we propose modifications to the

SLDPCD and STD in order to further improve their inherent tolerance to timing errors. This is achieved by performing extensive transistor-level and post-layout simulations, in order to develop different timing analyses for determining the causes and effects of timing errors in these stochastic decoders. Following this, we propose a novel Reduced-Latency STD (RLSTD), which improves the latency of the state-of-the-art STD by an order of magnitude, without increasing its chip area or energy consumption. Our experimental results demonstrate that our proposed RLSTD achieves ultra-low-latencies required by next-generation Mission-Critical Machine-Type Communication (MCMTC).

We also investigate the inherent tolerance to timing errors of a recently-proposed Fully-Parallel Turbo Decoder (FPTD). Furthermore, we propose a novel Reduced-Critical-Path Fully-Parallel Turbo Decoder (RCP-FPTD) algorithm and the employment of Better-Than-Worst-Case (BTWC) design techniques in FPTD and RCP-FPTD implementations, for the sake of improving their throughput and their tolerance to timing errors caused by overclocking. We demonstrate that the FPTD and RCP-FPTD implementations improve the throughput of the state-of-the-art turbo decoder by an order of magnitude. Finally, despite operating in the presence of timing errors, our proposed Better-Than-Worst-Case Reduced-Critical-Path Fully-Parallel Turbo Decoder (BTWC-RCP-FPTD) achieves throughputs on the order of tens of Gbps, which may be expected to be a requirement in next-generation wireless communication standards.

Contents

List of Figures

List of Tables

Nomenclature

APP	A Posteriori Probability
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Ratio
BPSK	Binary Phase Shift Keying
BS	Bernoulli Sequence
BTWC	Better-Than-Worst-Case
BTWC-FPTD	Better-Than-Worst-Case Fully-Parallel Turbo Decoder
BTWC-RCP-FPTD	Better-Than-Worst-Case Reduced-Critical-Path Fully-Parallel Turbo Decoder
CCDF	Complementary Cumulative Distribution Function
CG	Clock Gating
CMOS	Complementary Metal-Oxide Semiconductor
CN	Check Node
CRC	Cyclic Redundancy Check
DC	Decoding Cycle
DFF	D-type Flip Flop
DVFS	Dynamic Voltage and Frequency Scaling
EDAC	Error Detection and Correction
EM	Edge Memory
FP	Floating-Point
FPGA	Field-Programmable Gate Array
FPTD	Fully-Parallel Turbo Decoder
FX	Fixed-Point
IM	Internal Memory
JKFF	JK-Type Flip-Flop
LDPC	Low-Density Parity-Check
LLR	Logarithmic Likelihood Ratio
Log-BCJR	Logarithmic BCJR

Log-SPA	Logarithmic Sum-Product Algorithm
LSB	Least Significant Bit
LTE	Long Term Evolution
LUT	Lookup Table
MCMTC	Mission-Critical Machine-Type Communication
MPA	Message Passing Algorithm
MSA	Min-Sum Algorithm
MSB	Most Significant Bit
MUX	Multiplexer
NDS	Noise-Dependent Scaling
NSW	Non-Sliding Window
PCS	Parity-Check Satisfied
RB	Ring Buffer
RCP-FPTD	Reduced-Critical-Path Fully-Parallel Turbo Decoder
RCTFM	Reduced-Complexity Tracking Forecast Memory
RDFF	Razor D-Type Flip Flop
RLSTD	Reduced-Latency STD
SLDPCD	Stochastic LDPC Decoder
SNR	Signal to Noise Ratio
SoC	System-on-Chip
SPA	Sum-Product Algorithm
SR	Shift Register
STD	Stochastic Turbo Decoder
TET	Timing Error Type
TFM	Tracking Forecast Memory
TSMC	Taiwan Semiconductor
UMTS	Universal Mobile Telecommunications System
VLSI	Very-Large-Scale Integration
VN	Variable Node
WLAN	Wireless Local Area Network

Commonly Used Symbols

α_k	Forward recursion pertaining k^{th} bit in turbo decoders
β_k	Backward recursion pertaining k^{th} bit in turbo decoders
γ_k	Branch metrics pertaining k^{th} bit in turbo decoders
δ_k	Aposteriori branch metric pertaining k^{th} bit in turbo decoders
$\epsilon_{k,m,n}$	Intermediate result in RCP-FPTDs
η	Constant value in Noise-Dependendent Scaling
θ	Detection window in Razor DFFs
μ	Mean of Gaussian distribution
$\prod$	Interleaver
$\prod^{-1}$	De-interleaver
σ	Variance of Gaussian distribution
τ	Scaling factor in FPTDs
ϕ	Relaxation parameter in TFMs
ψ	Constant value in Noise-Dependendant Scaling
$^{\mathrm{a}}$	Superscript pertaining *a priori* information
$^{\mathrm{c}}$	Superscript pertaining channel information
$^{\mathrm{e}}$	Superscript pertaining *extrinsic* information
$^{\mathrm{l}}$	Superscript indicating lower
$^{\mathrm{u}}$	Superscript indicating upper
$\mathbf{b}$	Frame of bits in turbo decoding
$\hat{\mathbf{b}}$	Frame of estimated decoded bits in turbo decoders
$\tilde{\mathbf{b}}$	Frame of received bits in turbo decoders
$\boldsymbol{c}$	LDPC codeword
$\hat{\mathbf{c}}$	Frame of estimated decoded bits in LDPC decoders
$\boldsymbol{G}$	Generator matrix in LDPC decoding
$\boldsymbol{H}$	Parity-check matrix in LDPC decoding

$\boldsymbol{q}$	Vector of soft bits pertaining variable nodes in LDPC decoding
$\boldsymbol{r}$	Vector of soft bits pertaining parity-check nodes in LDPC decoding
$\boldsymbol{u}$	Vector of information bits in LDPC decoding
$\boldsymbol{x}$	Vector of transmitted symbols in LDPC decoding
$\boldsymbol{y}$	Vector of received symbols in LDPC decoding
$C_{i\setminus j}$	Set of CNs connected to the i^{th} VN, with the j^{th} CN excluded
$R_{j\setminus i}$	Set of VNs connected to the j^{th} CN, with the i^{th} VN excluded
d_i	Degree of variable nodes in LDPC decoding
d_j	Degree of parity-check nodes in LDPC decoding
$\mathrm{E_b/N_0}$	Signal to Noise Ratio per bit
IR	Current-Resistance noise in power supplies
k	Number of message bits in LDPC decoding
L	Logarithmic Likelihood Ratio
$\mathrm{L}{\cdot}d_i/d_t$	Inductive noise in power supplies
N	Frame length in turbo decoding
$\mathrm{N_0}$	Noise spectral density
n	Number of encoded bits in LDPC decoding
P	Probability
$\overline{P}$	Complementary probability
q	Quantization bits
R	Coding rate
S	Bernoulli sequence
s	Current state
s'	Previous state
T_{clk}	Clock period
t	Time index
t_h	Hold time
t_p	Propagation delay of path p
t_s	Setup time
$\mathrm{V_{DD}}$	Supply voltage

Declaration of Authorship

I, **Isaac Pérez Andrade**, declare that the thesis entitled ***Timing-Error-Tolerant Iterative Decoders*** and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

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

Signed:.....

Date:.....

Acknowledgements

I would like to express my gratitude to my supervisor, Dr Robert G. Maunder, for his outstanding supervision, support and dedication throughout my PhD study. I would also like to thank Prof. Bashir M. Al-Hashimi for providing me with the opportunity to undertake this research. I would also like to express my gratitude and admiration to Prof. Lajos Hanzo, whose valuable advice and guidance provided during these years have greatly helped me to successfully conclude this research.

Despite the distance, my family has been the main motivation for completing this PhD. I would like to thank my parents, Amelia and Pascual, for all the constant support and love provided not only during my PhD, but throughout my life. I am deeply grateful to them for all their efforts and the sacrifices that they have made to provide me and my siblings with an education. I would also like to thank my brother Roberto for being an exceptional example and my sister Saraí for her constant support and for always drawing a smile upon my face.

This PhD would not have been possible without the unconditional support, care and motivation from Eszter. I sincerely thank you for being there in the good times, but especially during the bad times along these years. Thank you for all the talks, walks, meals and endless activities you always had in your mind so I could distract mine. I am also thankful for all the support that Marianna, György Sr. and György Jr. have constantly provided.

I would also like to thank my friends Marce, Penélope and David for always helping me to clear my mind with non-academia chats, as well as Faby, Luis, Mauricio and Emma for all the help provided outside of University.

Finally, I would like to acknowledge the financial support of 'Consejo Nacional de Ciencia y Tecnología de México (CONACyT)' under the auspices of the scholarship 311103 and of 'Secretaría de Educación Pública y del Gobierno Mexicano'.

To all the people listed above, *Thank you!*

List of Publications

I. Perez-Andrade, X. Zuo, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Analysis of voltage- and clock-scaling-induced timing errors in stochastic LDPC decoders," in *IEEE Wireless Communications and Networking Conf. (WCNC-2013),* April, 2013, pp. 4293-4298.

I. Perez-Andrade, S. Zhong, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Stochastic computing improves the timing-error tolerance and latency of turbo decoders: Design guidelines and trade-offs," in *IEEE Access,* vol. 4, pp. 1008–1038, February, 2016.

X. Zuo, **I. Perez-Andrade,** R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Improving the tolerance of stochastic LDPC decoders to overclocking-induced timing errors: A tutorial and a design example," in *IEEE Access,* vol. 4, pp. 1607–1629, April, 2016.

X. Zuo, S. Zhong, K. Li, **I. Perez-Andrade,** R. G. Maunder, B. M. Al-Hashimi, L. Hanzo, "High throughput timing error tolerant VLSI implementation of LDPC decoding using the base-minus-two fixed-point number representation", in *IEEE J. Solid-State Circuits,* [In preparation].

I. Perez-Andrade, S. Zhong, K. Li, A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Timing-error-tolerant VLSI implementation of fully-parallel turbo decoding," in *IEEE J. Solid-State Circuits,* [In preparation].

Chapter 1

Introduction

During the recent decades, wireless communication systems have greatly benefited from iterative decoding algorithms, which iteratively refine a probabilistic view of the transmitted messages, until the correct decoding decisions can be made. This powerful technique therefore has an *inherent* capability of correcting the *transmission* errors that originate during communication over a hostile wireless channel. As a result, iterative decoders offer a near-optimal decoding performance, close to the theoretical Shannon's capacity limit [1]. This has motivated the widespread use of iterative decoders such as Low-Density Parity-Check (LDPC) [2] decoders and turbo decoders [3] in current digital transmission standards. For example, LDPC decoders are used in the broadband wireless access WiMAX [4], Wireless Local Area Network (WLAN) [5], Ethernet [6] and digital video broadcasting DVB-S2 [7] standards. Meanwhile, turbo decoders are used in the Universal Mobile Telecommunications System (UMTS) [8], Long Term Evolution (LTE) [9] and WiMAX cellular telephony standards. Furthermore, both LDPC and turbo decoders are currently being considered for next-generation wireless communication standards [10, 11], owing to their high-throughput capability.

Owing to their near-optimal decoding performance, substantial research efforts have been invested in conceiving efficient Very-Large-Scale Integration (VLSI) implementations of LDPC and turbo decoders, using both Field-Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) technologies. Typically, these efforts have focused on optimizing only one of the various trade-offs associated with the hardware implementation of iterative decoders, as shown in Figure 1.1. In a Pareto optimal design, none of these trade-offs may be further improved without degrading at least one of the others. To elaborate further, Figure 1.1 shows the close relationship between chip area, latency, throughput, energy efficiency and error correction capabilities in iterative decoders, as well as the factors that may influence each of these trade-offs. As an example of this, the chip area of an implementation may be influenced by the design of the algorithm, as well as its hardware description and its logical and physical synthesis for a

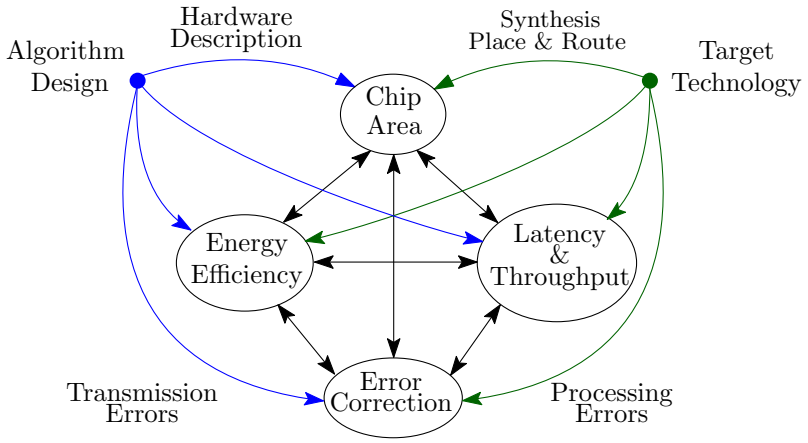


Figure 1.1: Relationship between different design trade-offs in iterative decoders.

target technology. Similarly, the chip area may influence the energy efficiency, latency, throughput and error correction capabilities of the implementation. For example, LDPC decoders benefit from the inherent parallelism of their decoding algorithms. As a result, it is common for VLSI designers to consider fully-parallel LDPC decoders, in order to achieve high processing throughputs. However, this approach incurs a large and probably unacceptable chip area. Likewise, the inherent serial data-dependencies of the turbo decoding algorithms have motivated designers to conceive serial turbo decoder implementations. Although this approach facilitates relatively low chip areas, it incurs large processing latencies. In addition to this, tolerance to timing errors that occur during the iterative decoding processing are typically not considered in these implementations. As a result of this, the error correction and hardware performance of the proposed design may be severely degraded, if timing errors occur during the iterative decoding process, owing to overclocking or power supply noise, for example. Note that throughout this thesis, we employ the term *energy* in order to refer to the capacity to perform work. This is justified since the energy efficiency of an implementation determines its power dissipation in the form of heat, owing to the close relationship between energy and power. As an example of this, the higher the energy consumption of a system, the higher its power dissipation.

As fabrication technology scales below the deep-submicron regime, it is no longer possible to guarantee the reliable operation of ASICs in the presence of process, voltage or temperature variations, as well as other sources of noise such as IR-drop, $L \cdot d_i/d_t$ noise, crosstalk, electrostatic discharges, particle strikes and switching noise [12, 13, 14, 15, 16], among other causes. As a result, error-tolerant design techniques are becoming a critical and essential feature of modern VLSI devices. Traditional error-tolerant design techniques rely on hardware redundancy and voting units [17]. Here, the same logic function is performed by an odd number of identical and independent units, which pass their result to a majority voting module, in order to determine the correct results. However, this approach incurs large chip area extensions. In order to mitigate the effects of

processing errors, VLSI designers adopt conservative design methodologies, where the design is conceived using safety margins above the worst-case scenario. In this way, they avoid the excessive cost of hardware replication, albeit at the cost of operating the system under overly conservative conditions. However, such a worst-case scenario may be rare or even impossible to encounter.

Over the past decade, new error-tolerant design approaches have arisen. For example, stochastic computing [18] has been recently proposed for the iterative decoding of LDPC and turbo codes. Stochastic computing has been proposed as a low-complexity design alternative to Fixed-Point (FX) binary arithmetic. Here, the probabilities in the probabilistic view of the transmitted message are represented by streams of bits, known as Bernoulli Sequences (BSs) [18]. Only a single bit of each BS is processed per clock cycle and the specific fraction of bits having the logical value 1 determines the value of the probability represented. As a benefit of this, arithmetic operations in stochastic computing can be implemented using low-complexity digital circuits. Moreover, stochastic computing offers an inherent tolerance to processing errors. Since every bit of a BS has an identical and relatively-low significance, a processing error causing a bit-flip will only result in a small change to the overall value of the probability represented. This represents a key advantage of the stochastic implementation of iterative decoders, particularly as VLSI technologies scale.

In parallel to the employment of stochastic computing in iterative decoders, Better-Than-Worst-Case (BTWC) design techniques [19] have become a common denominator in current high-performance microprocessor systems. In this design approach, the voltage and/or the clock period of a system are scaled below the recommended safety margins, for the sake of reducing the energy consumption or improving the throughput, respectively. Here, Error Detection and Correction (EDAC) techniques [13, 20, 21, 22] are employed in order to correct and restore the state of the system. This is achieved by increasing the voltage and/or the clock period and repeating those operations that triggered the timing error. However, this functionality is only naturally supported in current high-performance microprocessors with a dedicated instruction replay mechanism [13]. These error tolerant design techniques are discussed in greater detail in Section 1.1.

1.1 Related Work

Section 1.1.1 summarizes previous contributions in the field of stochastic decoders, while Section 1.1.2 summarizes previous contributions in the field of fault-tolerant iterative decoders.

1.1.1 Stochastic Iterative Decoders

Table 1.1 summarizes some of the seminal contributions in the field of stochastic iterative decoders. In 2003, Gaudet [23] and Rapley [24] proposed stochastic computing in iterative decoders for the first time. They demonstrated that it is possible to perform iterative decoding using low-complexity digital gates. As a result, stochastic computing has been recently proposed for the fully-parallel decoding of LDPC codes [23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35], Bose-Chaudhuri-Hocquenghem codes [36], Reed-Solomon codes [36], cortex codes [37], convolutional codes [38] and turbo codes [39, 40, 41, 42]. Among these contributions, several research efforts have been invested in optimizing the decoding capabilities of stochastic decoders. For example, the authors of [27] proposed Edge Memory (EM) and Noise-Dependent Scaling (NDS) in order to overcome the latching problem in Stochastic LDPC Decoders (SLDPCDs). This allowed the practical implementation of SLDPCDs on FPGAs in [28, 29]. Moreover, the authors of [29] further improved the error correction performance of the (1056,528) SLDPCD by using Internal Memories (IMs) and different EMs lengths according to the degree of each stochastic node. This implementation was further optimized in [30, 33], where Shift Register (SR)-based EMs were replaced by Tracking Forecast Memories (TFMs).

Despite the near-optimal decoding performance and the relatively-low-complexity associated with their hardware implementation, stochastic decoders require a large number of clock cycles in order to achieve the same near-optimal error correction capability as their FX implementation counterparts. Regretfully, this degrades their latency, throughput and energy efficiency. Owing to these impediments, stochastic decoders have been previously deemed unsuitable for practical low-latency next-generation Mission-Critical Machine-Type Communication (MCMTC) systems, such as those required by vehicular traffic safety and control, as well as industrial process automation and manufacturing [43]. In these applications, short emergency and control messages constituted by a low number of bits must be reliably transmitted with ultra-low latency, hence motivating the employment of error correction decoders having ultra-low processing latencies on the order of microseconds [43]. Along this road, substantial research efforts have been invested in reducing the number of clock cycles required by stochastic decoders for achieving a near-optimal error correction performance. In particular, the employment of NDS and EMs in [27, 28, 30, 33] reduced the number of clock cycles required by SLDPCDs to achieve near-optimal error correction performance from several thousands to as low as a few hundreds. By contrast, the fully-parallel Stochastic Turbo Decoder (STD) of [39] requires several thousands of processing cycles, despite the employment of NDS and EMs. This problem is partially overcome in [39, 40] by further increasing the grade of processing parallelism. However, this significantly increases the hardware complexity of the STD. The authors of [41, 42] proposed modified BS representations for the implementation of STDs. These contributions significantly reduced

the number of clock cycles required by STDs without significantly increasing the hardware complexity of the design. However, these designs may be considered to represent partially-stochastic turbo decoders, since their operation is based on fixed-point arithmetic circuits, which do not benefit from the inherent tolerance of stochastic decoders to timing errors.

Table 1.1: Selected previous contributions in the field of stochastic iterative decoders.

Year	Author	Contribution
2003	Gaudet and Rapley [23]	Stochastic computing proposed in iterative decoding for the first time.
	Rapley *et al* [24]	Stochastic decoding of a (7,4) Hamming code.
2005	Winstead *et al* [25]	Stochastic computing in block turbo codes.
	Gross, Gaudet and Milner [26]	First FPGA implementation of SLDPCDs.
2006	Sharifi Tehrani, Gross and Mannor [27]	EMs and NDS for avoiding the latching problem in SLDPCDs.
2007	Sharifi Tehrani, Mannor and Gross [28]	First practical SLDPCD on FPGAs.
2008	Sharifi Tehrani, Mannor and Gross [29]	IMs and different EMs lengths in a (1056,528) SLDPCD on FPGA.
	Sharifi Tehrani *et al* [36]	Stochastic decoding of Bose-Chaudhuri-Hocquenghem codes and Reed-Solomon codes.
2009	Sharifi Tehrani *et al* [30]	TFMs in SLDPCDs. First ASIC implementation of SLDPCDs.
	Leduc-Primeau *et al* [31]	Relaxed-half-stochastic LDPC algorithm.
2010	Sarkis and Gross [32]	Non-binary SLDPCD algorithm.
	Sharifi Tehrani *et al* [33]	MBTFM-based SLDPCDs on ASICs.
	Dong *et al* [39]	Stochastic decoding of turbo codes.
2011	Naderi *et al* [34]	Delayed SLDPCDs on ASICs.
	Arzel *et al* [37]	Multiple stream decoding of cortex codes.
	Dong *et al* [40]	STDs on FPGAs.
2013	Te-Hsuan and Hayes *et al* [38]	Stochastic decoding of convolutional codes.
	Hu *et al* [41]	Half-stochastic turbo decoders.
	Chen and Hu *et al* [42]	Improved half-stochastic turbo decoders.
2014	Ceroici and Gaudet [35]	Clockless SLDPCDs on FPGAs.

1.1.2 Fault-Tolerant Iterative Decoders

Table 1.2 summarizes some of the seminal contributions in the field of fault tolerance in iterative decoders. Fault-tolerance in iterative decoders has mostly been explored from the algorithmic point of view in [44, 45, 46, 47, 48, 49, 50, 51, 52]. The vast majority of these contributions rely on probabilistic analysis in order to apply compensation techniques to mitigate the effects of processing errors. However, these contributions do not consider the trade-offs associated with the hardware implementation of their error-tolerant design. Only a small number of the contributions [53, 54, 55, 56] of Table 1.2 have considered the practical implications of fault-tolerant hardware design in iterative decoders. Within these contributions, different approaches have been explored in order to improve the decoder's fault-tolerance. For example, hardware redundancy and voting units are employed in [53] for error detection and correction. Processing errors are corrected on the basis of probabilistic analyses in [54, 55]. Finally, [56] only characterizes the occurrence of timing errors caused by overscaling the supply voltage in LDPC decoders.

Table 1.2: Selected previous contributions in the field of fault-tolerant iterative decoders.

Year	Author(s)	Contribution
2007	Alles, Brack and Wehn [44]	Error resilient LDPC code design.
2008	May, Alles and Wehn [53]	Hardware redundancy in LDPC decoders.
2009	Winstead and Howard [45]	Probabilistic analysis for fault compensation in LDPC decoders.
	Abdallah and Shanbhag [54]	Algorithmic noise tolerant in hardware implementation of Viterbi decoders.
2010	Gaudet [46]	Analysis of the inherent error tolerance of the SPA algorithm in LDPC decoders.
2012	Tang *et al* [47]	LDPC decoding tolerant to transient errors.
	Winstead *et al* [48]	Space-time redundancy algorithms in LDPC decoders.
	Kim and Shanbhag [55]	Statistical error compensation in hardware implementation of LDPC decoders.
2013	Geldmacher and Gotze [49]	EXIT chart-aided analysis of faulty iterative decoders.
2014	Kameni Ngassa, Savin and Declerq [50]	Analysis of fault tolerant SLDPCDs in the binary symmetric channel.
	Andrade *et al* [51]	Analysis of turbo decoders with unreliable memories.
	Sedighi, Prasanth and Suvakovic [56]	Characterization of timing error occurrence in voltage overscaled LDPC decoders.

Table 1.2: Selected previous contributions in the field of fault-tolerant iterative decoders.

Year	Author(s)	Contribution
2015	Huang, Li and Dolecek [52]	Analysis of belief propagation algorithms on noisy hardware.

Note that from the selected contributions of Tables 1.1 and 1.2, only the authors of [50] analyze the inherent tolerance to processing errors of SLDPCDs. However, this is achieved from the algorithmic point of view for the simplified case of the binary symmetric channel.

1.2 High-throughput Iterative decoders

The next-generation of wireless communication standards [10, 11] are expected to require processing throughputs on the order of tens of Gbps. Significant efforts have been made along the road to fulfilling these throughput requirements. As an example of this, the state-of-the-art turbo decoder implementations achieve processing throughputs of 2.15 Gbps [57], 1.67 Gbps [58] and 1.28 Gbps [59]. These turbo decoder implementations rely on increasing the parallelism of the highly-serial turbo decoding algorithm. Despite these efforts, the proposed solutions still require hundreds of clock periods to complete a decoding iteration, resulting in a requirement for thousands of clock periods for completing the iterative decoding process. This may be attributed to the inherently serial nature of the turbo decoding algorithm. Moreover, the implementations of [57, 58, 59] do not consider fault tolerant design. As a result of this, the significant throughput gains offered by these implementations may be diminished by the occurrence of processing errors, if the operating conditions of the system fluctuate below the recommended safety margins.

In parallel to this, in order to fulfill the high throughput requirements of next-generation wireless communication standards, a Fully-Parallel Turbo Decoder (FPTD) [60] algorithm and VLSI [61] implementation has been recently proposed. This FPTD facilitates throughputs in the order of tens of Gbps, since it disposes with the highly-serial data dependencies of the traditional turbo decoding algorithm. Owing to this, each FPTD decoding iteration may be completed in a single clock period, requiring only tens of clock periods to complete the iterative decoding process, albeit at the cost of a large computational complexity.

1.3 Motivation and Thesis Outline

The previous work mentioned in Section 1.1 has mainly focused on enhancing the error correction capabilities of iterative decoders. Similarly, fault-tolerance in iterative decoders has mainly been addressed from the algorithmic point of view. Moreover, the above-mentioned contributions in the field of fault-tolerant iterative decoders have failed to provide a comprehensive analysis of the various design trade-offs involved in the hardware implementation of error tolerant techniques, which is a common approach in performance-oriented design methodologies [62]. To elaborate further, the main design objective of these contributions has been to enhance both the robustness and reliability of the decoders against processing errors. However, this is typically reported without considering the impact of fault-tolerant techniques on the associated hardware design trade-offs, such as the chip area, energy efficiency, latency, throughput and error correction capabilities, despite the strong dependence they have upon each other, as depicted in Figure 1.1. In this context, a compelling Pareto-optimal design has a set of characteristics, where none of them can be further improved without degrading at least one of the others. In parallel to this, previous research efforts have been mainly focused on the implementation of fault-tolerant fully-parallel FX LDPC decoders. However, aspects of fault-tolerant fully-parallel implementations of SLDPCDs and STDs, as well as of fully-parallel FX turbo decoders have not yet been explored, as summarized in Table 1.3.

Table 1.3: Research opportunities in the field of fully-parallel fault-tolerant implementations of LDPC decoders and turbo decoders.

	Fully-parallel LDPC decoders	Fully-parallel Turbo decoders
Fixed-Point	Significantly explored	Not explored
Stochastic	Not explored	Not explored

*Against this background, this thesis demonstrates that iterative decoders are capable of exploiting their **inherent** error correction capability to correct not only **transmission** errors, but also **timing** errors caused by overclocking and power supply variations. Moreover, we propose modifications to the iterative decoders designs, which further enhance their inherent tolerance to timing errors. We achieve this by considering the close relationship between the different trade-offs associated with the hardware implementation of iterative decoders, with the aim of achieving Pareto optimality. Owing to this, our proposed timing-error-tolerant design methodology simultaneously considers the design constraints and parameters that affect not only the Bit Error Ratio (BER) performance, but also the hardware efficiency of each implementation.*

This thesis is structured as depicted in Figure 1.2.

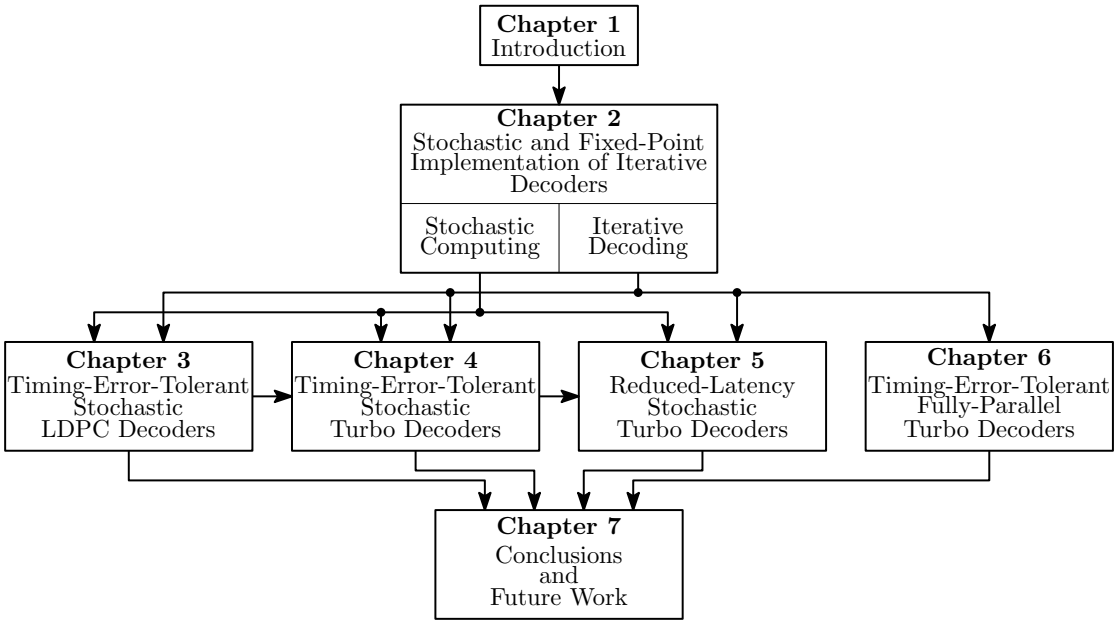


Figure 1.2: Thesis outline.

Chapter 1 presents the introduction of this thesis, including a perspective of the related previous work in the selected areas of fault-tolerance in iterative decoders.

Chapter 2 reviews the basic concepts of iterative decoding and some fundamental aspects of the hardware implementation of iterative decoders. These concepts are employed throughout this thesis. Moreover, this chapter reviews the concept of stochastic computing, which is the foundation for the stochastic implementation of LDPC decoders in Chapter 3 and the stochastic implementation of turbo decoders in Chapters 4 and 5.

Chapter 3 reviews the concept of LDPC decoding and the implementation of the Sum-Product Algorithm (SPA) algorithm using stochastic computing. Section 3.3 characterizes the hardware efficiency and the BER performance of the (1056,528) WiMAX SLDPCD of [29], when using ST 90 nm technology. Moreover, extensive SPICE simulations are used in order to determine the causes and effects of timing errors in SLDPCDs. Furthermore, we present a modified SLDPCD that further improves the timing error tolerance of the stochastic implementation of LDPC decoders. Additionally, we characterize the different trade-offs associated with the hardware implementation of the modified SLDPCDs.

Chapter 4 reviews the concept of turbo decoding and the implementation of the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm using stochastic computing. We presents various modifications to the STD of [39] that enhance its tolerance to timing errors in the presence of power supply variations and prevents the catastrophic

propagation of metastability through the circuit. Moreover, the proposed modifications significantly improve the hardware efficiency of the STDs, when using Taiwan Semiconductor (TSMC) 90 nm technology.

Chapter 5 presents a second set of improvements to the STD of [39], in order to significantly reduce the number of clock cycles required in order to achieve iterative decoding convergence.

Chapter 6 expands the review of the BCJR algorithm into the logarithmic domain, in order to directly compare it with the recently-proposed FPTD algorithm of [60]. We also review the hardware implementation of the FPTD algorithm, as detailed in [61]. Moreover, we present a novel Reduced-Critical-Path Fully-Parallel Turbo Decoder (RCP-FPTD) algorithm that reduces the number of data path stages required in the FPTD and hence the required clock period of its implementation. We also detail the hardware implementation of the proposed RCP-FPTD, as well as its corresponding trade-off analysis, when using TSMC 40 nm technology. Furthermore, we develop a timing error model of the FPTD and RCP-FPTD implementations and employ BTWC design techniques for the implementation of the FPTD, in order to significantly enhance its inherent tolerance to timing errors and its hardware efficiency.

Chapter 7 summarizes the findings of our investigations and provides opportunities regarding future work.

1.4 Novel Contributions

The novel contributions of this thesis are summarized as follows.

In **Chapter 3** we characterize the causes and effects of timing errors caused by overclocking and power supply variations in SLDPCDs. We propose a Ring Buffer (RB)-based SLDPCD that further improves the timing error tolerance of the SR-based SLDPCD of [29]. We characterize the trade-offs associated with the hardware implementation of the SR- and RB-based (1056,528) WiMAX SLDPCDs, when using ST 90 nm technology. Our experimental results demonstrate that our proposed RB-based SLDPCD operated at a supply voltage of 0.8 V, an overclock period of 800 ps and in the presence of timing errors owing to 10% power supply variations, offers the same BER performance as the SR-based SLDPCD of [29] operated a 0.8 V and a clock period of 1160 ps in the absence of timing errors, while increasing the throughput by a factor of 1.22, reducing the energy consumption by a factor of 0.7 and requiring only 0.77 the chip area of that of the SR-based SLDPCD of [29].

In **Chapter 4** we present various modifications to the STD of [39] that enhance its tolerance to timing errors and significantly improve its hardware efficiency, when using TSMC 90 nm technology. More specifically, we propose: i) the employment of synchronizers for preventing the catastrophic cascading of metastability owing to timing errors; ii) the simultaneous decoding of two received frames for improving the processing throughput; iii) the employment of TFMs [30] in STDs for the first time, in order to enhance their hardware implementation and decoding capabilities; and iv) the inclusion of a pipelining stage for enhancing the decoding. Our proposed TFM-based STD design operated at 1.20 V, a clock period of 2.2 ns and in the presence of timing errors owing to 7% power supply variation offers the same BER performance as the state-of-the-art STD of [39], when operated at 1.20 V, a clock period of 4.0 ns and in the absence of power supply variations. This is achieved while increasing the throughput by a factor of 2.42, reducing the latency by a factor of 0.83 and consuming only 0.25 times the energy, without increasing the chip area.

In **Chapter 5** we propose a Reduced-Latency STD (RLSTD) design, which reduces the number of clock cycles required for achieving near-optimal error correction performance by an order of magnitude, without increasing the chip area. This is achieved by employing: i) OR gates for performing approximate stochastic additions; ii) a reduced-complexity TFM design for overcoming the latching problem iii) a single D-type Flip Flop (DFF) for estimating each decoded bit. Our proposed RLSTD design achieves the same BER performance as the state-of-the-art STD of [39], while improving its latency, throughput and energy efficiency by an order of magnitude and without imposing an area extension. More specifically, our proposed RLSTD requires only 0.015 times the latency, 0.005 times the energy and 0.51 times the chip area of the state-of-the-art STD, while offering a 65 times throughput increase.

In **Chapter 6** we present a novel RCP-FPTD algorithm, which further improves the processing throughput of the FPTD of [60]. Our results demonstrate that the FPTD and RCP-FPTD implementations offer significant throughput increases, when compared to both the RLSTD of Chapter 5 and the state-of-the-art turbo decoder implementation of [57], while achieving the same BER performance. The FPTD and RCP-FPTD implementations offer throughputs that are 7.9 and 9.5 times superior to those of the turbo decoder of [57], respectively, albeit at the cost of requiring 5.42 and 5.58 times the chip area. We demonstrate that the FPTD and RCP-FPTD implementations have an inherent tolerance to timing errors, when overclocking is employed for the sake of further improving their throughput. We propose to use the BTWC design approach [19] and EDAC techniques [13, 20, 21, 22] for mitigating the effect of timing errors in the FPTD and RCP-FPTD

implementations. Our results of Section 6.7 demonstrate that the proposed Better-Than-Worst-Case Fully-Parallel Turbo Decoder (BTWC-FPTD) and Better-Than-Worst-Case Reduced-Critical-Path Fully-Parallel Turbo Decoder (BTWC-RCP-FPTD) operated in the presence of timing errors achieve a processing throughput that is 1.7 and 2.47 times superior to that of the proposed RLSTD of Chapter 5, respectively, when employing the shortest frame length of $N = 40$ supported by the LTE standard. Similarly, when employing LTE's longest frame length of $N = 6144$, the proposed BTWC-FPTD and Better-Than-Worst-Case Reduced-Critical-Path Fully-Parallel Turbo Decoder (BTWC-RCP-FPTD) operating in the presence of timing errors achieve processing throughputs that are 9.2 and 12.3 times superior than that of the state-of-the-art turbo decoder of [57] in the absence of timing errors, respectively.

Chapter 2

Fixed-Point and Stochastic Iterative Decoders

This chapter briefly reviews the concept of iterative decoding and aspects of its hardware implementation, since these are the foundations of the algorithms and hardware implementations described in the following chapters. Section 2.1 reviews the operation of iterative decoders in communications systems. Section 2.2 reviews some fundamentals of Fixed-Point (FX) two's complement numbers, while Section 2.3 reviews the concept of stochastic computing and its application to iterative decoders. Finally, Section 2.4 presents the conclusions of this chapter.

2.1 Fundamentals of Iterative Decoding

Figure 2.1 shows a simplified communication scheme, where a source message comprising a sequence of bits is encoded and modulated, before it is transmitted over a channel. Upon its reception, the demodulator will be uncertain of the bit values in the original source message, owing to the effect of noise in the channel. As a result of this, the demodulator provides soft-valued bits, instead of hard-valued bits. These soft bits express not only *what* the most likely values of the corresponding bits are, but also *how* likely these bit values are. Following this, the iterative decoder processes the soft bits, with the ultimate aim of generating hard-valued bits, which correspond to the best estimation of the source message bits.

Iterative decoding may operate on the basis of a serial or parallel concatenation of component decoders, as shown in the lower part of Figure 2.1. In this way, the iterative decoding process consists of the two component decoders feeding and processing soft-valued bits back and forth between each other through an interleaver $\prod$ and a de-interleaver $\prod^{-1}$, which reorder the soft-valued bits in a non-contiguous way. In this

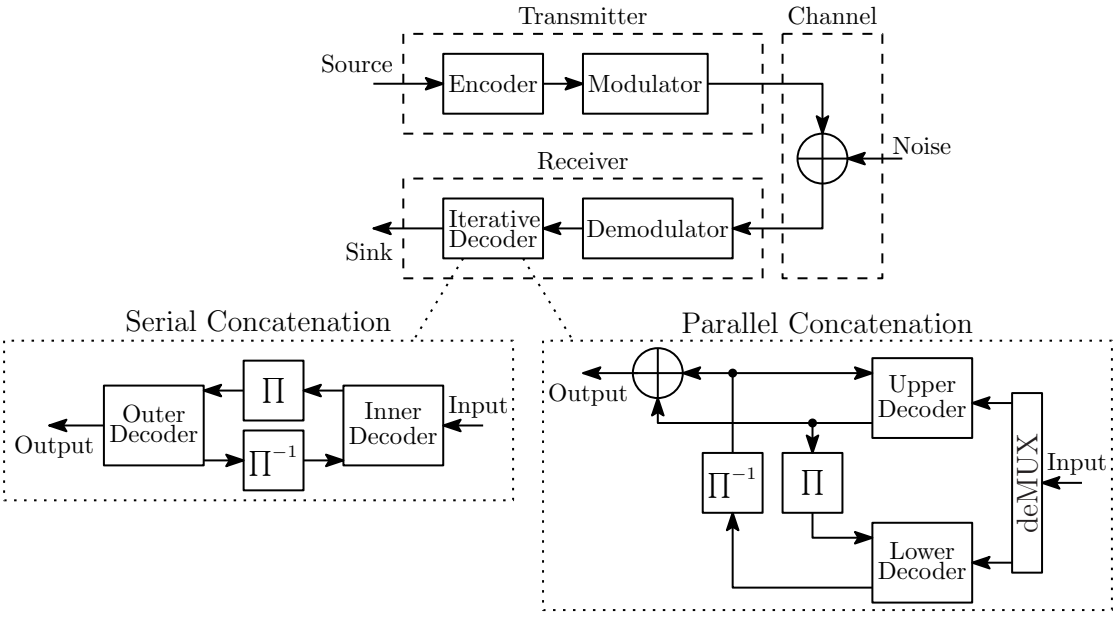


Figure 2.1: Simplified block diagram of a communication scheme using iterative decoding.

configuration, the soft-valued bits improve in each iteration, until the best estimation of the output is achieved, or until a maximum number of affordable iterations has been reached. Low-Density Parity-Check (LDPC) decoders [2] and turbo decoders [3] are examples of serially-concatenated and parallel-concatenated iterative decoders, respectively. More specifically, LDPC decoders can be considered to employ a serial concatenation of a so-called Variable Node (VN) inner decoder and a so-called Check Node (CN) outer decoder [2], where the operation of these decoders differs from each other. By contrast, turbo decoders are an example of a parallel concatenation of so-called upper and lower Logarithmic BCJR (Log-BCJR) decoders [63], where both decoders operate in the same fashion.

The operation of LDPC decoders is further discussed in Chapter 3, while the operation of turbo decoders is further discussed in Chapters 4 to 6. The following sections provide an overview of two different approaches for the hardware implementation of iterative decoders.

2.2 Fixed-Point Two's Complement in Iterative Decoders

As mentioned in Chapter 1, substantial research efforts have been invested in conceiving efficient Very-Large-Scale Integration (VLSI) implementations of LDPC and turbo decoders. Typically, these efforts rely on the FX two's complement number representation, owing to its attractive trade-off between error correction performance and hardware implementation complexity, as shown in Figure 1.1. In this number representation, both positive and negative numbers may be represented using q quantization bits. Here, one

of the bits of the FX number represents the sign of each represented value, while its magnitude is determined by the weighted sum of the other $(q-1)$ bits. More specifically, the sign of the represented number is determined by the Most Significant Bit (MSB) of the FX number, which has a weight of -2^{q-1}. For example, -6 is represented by 1010 $q = 4$-bits two's complement, where the MSB has a weight of -8 and the other 1-valued bit has a weight of 2, since the three Least Significant Bits (LSBs) have weights of 4, 2 and 1 from left to right. In this way, a two's complement FX number F may take any of the integer value within its dynamic range of

$$-(2^{q-1}) \leq F \leq 2^{q-1} - 1. \tag{2.1}$$

Owing to this, the number of quantization bits q used in FX iterative decoders determines both their dynamic range and their hardware complexity [64]. To elaborate further, larger values of q result in larger dynamic ranges, which aid the iterative decoding process [64]. However, larger q values also results in larger hardware complexities, owing to the employment of larger FX arithmetic circuits, larger number of interconnections and larger memory requirements. As a result of this, it is necessary to carefully consider this trade-off when selecting a q value for the soft-valued bits used during the iterative decoding process.

Special considerations must be taken during the hardware implementation of FX numbers, owing to the limitation imposed by their dynamic range. For example, overflow may occur, if the addition of two q-bit FX numbers cannot be represented with the same number of q bits. When overflow occurs, the MSB of the FX number flips its value from logic 0 to logic 1, or from logic 1 to logic 0, resulting in an absolute error of 2^q in the represented number. For example, the computation $4 + 6$ in $q = 4$-bit two's complement arithmetic is performed as 0100+0110=1010, which represents -6, rather than 10 as desired. This problem can be easily overcome by increasing the number of q bits after performing an arithmetic operation, such as addition and subtraction. However, this may cause the number of q bits and the magnitude of the FX numbers to grow rapidly, significantly increasing the hardware complexity from one processing stage to the next. In order to overcome this problem, clipping may be employed for ensuring that the results of FX arithmetic calculations remain within the dynamic range supported using q bits. For example, when using clipping for the case of $q = 4$, the computation $4+6$ gives the result 7, since this is the highest value within the dynamic range of $-8 \leq F \leq 7$.

Chapter 6 expands the discussion presented in this section for the case of the Fully-Parallel Turbo Decoder (FPTD) and Reduced-Critical-Path Fully-Parallel Turbo Decoder (RCP-FPTD) implementations. In addition to this, Section 2.3 reviews the concept of stochastic computing [18], which has been recently proposed as a low-complexity alternative to traditional FX two's complement in iterative decoders.

2.3 Stochastic Computing in Iterative Decoding

In contrast to FX two's complement, in stochastic computing [18], probabilities are represented by means of streams of bits known as Bernoulli Sequence (BS), which are generated by statistically independent processes and with only one bit of each BS being processed in each clock cycle that is referred to as a Decoding Cycle (DC) of stochastic decoders. The probability P represented by a BS is determined by the fraction of its bits having the value of 1. Owing to this, the same probability can be represented by different BSs having the same fraction of bits with the value 1, but in different positions. This is exemplified in Figure 2.2(a), where the probability $P = 0.75$ is represented by different BSs, each having 12 out 16 bits with value 1. As an explicit benefit of this, stochastic computing offers an inherent tolerance to processing errors. To elaborate further, a single bit-flip caused by a processing error will only change the overall value of the represented probability by a fraction proportional to the length of the BS. In this way, if the BS is sufficiently long, the represented probability will not be significantly affected, if some of the bits are corrupted. As an example of this, if any of the 12 bits having the value of logical 1 of the BSs of Figure 2.2(a) is flipped to logical 0, the resultant BSs will represent the probability of $P = 11/16 = 0.6875$, which corresponds to an absolute error of 1/16.

2.3.1 Stochastic Arithmetic

In stochastic computing, arithmetic combinations of the probabilities represented by two or more BSs can be implemented using low-complexity digital circuits. In particular, the binary complement $\bar{P} = (1 - P)$ of a probability P can be obtained using a NOT gate, as exemplified in Figure 2.2(b).

The multiplication $P_{mult} = P_A \cdot P_B$ of two probabilities P_A and P_B can be performed using a bitwise logical AND of the two corresponding BSs S_A and S_B, as exemplified in Figure 2.2(c). Similarly, the multiplication $P_{mult} = \prod_{i=1}^{M} P_i$ of the probabilities represented by M BSs can be performed with the aid of an M-input AND gate.

The weighted mean $P_{add} = P_A \cdot (1 - P_{sel}) + P_B \cdot P_{sel}$ of two probabilities P_A and P_B can be obtained by using a two-input Multiplexer (MUX) to randomly select bits from the corresponding BSs S_A and S_B, with the aid of the BS S_{sel}, which represents the probability $P_{sel} = 0.5$ for the case of a non-weighted mean, as exemplified in Figure 2.2(d). The outgoing bit of the MUX will have the value of 1 only if the bit of the input selected by S_{sel} has the value of 1. More specifically, we have $S_{add} = 1$ only if $S_A = 1$ and $S_{sel} = 0$ or $S_B = 1$ and $S_{sel} = 1$. Similarly, the mean $P_{add} = [\sum_{i=1}^{M} P_i]/M$ of M probabilities can be obtained using an M-input MUX to randomly select bits from the corresponding M BSs.

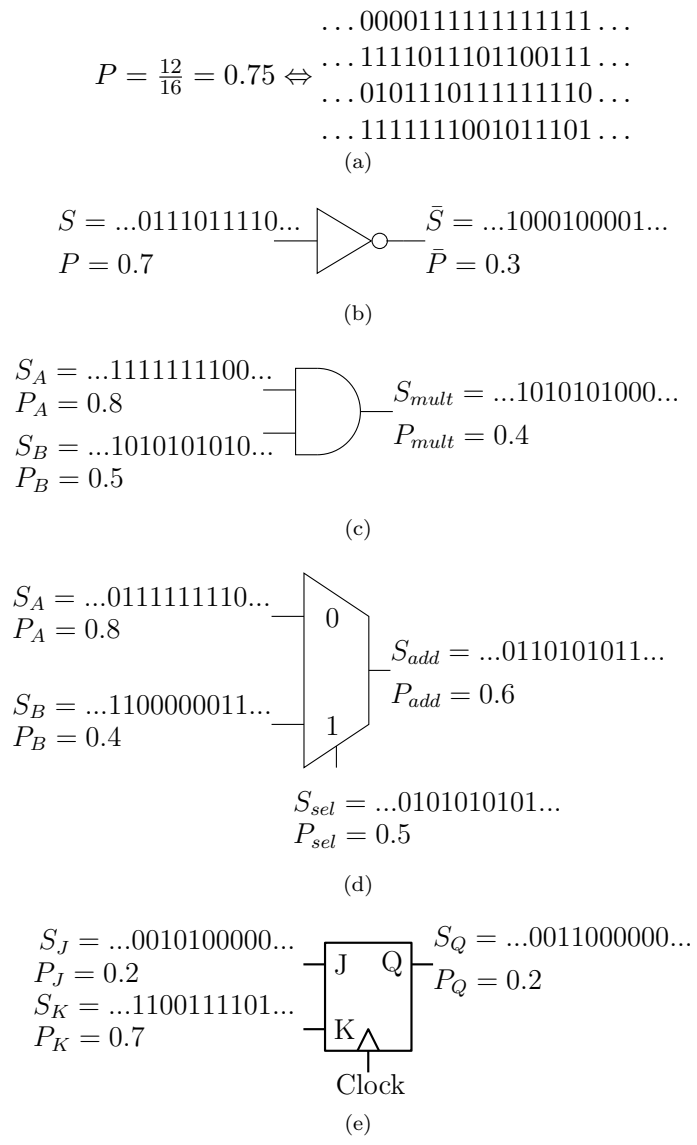


Figure 2.2: Stochastic circuits for arithmetic operations: (a) Different BSs for $P = 0.75$. (b) Complement. (c) Multiplication. (d) Scaled addition. (e) Approximate division and normalization.

The division of the probability P_J by the probability P_K can be approximated by entering the corresponding BSs S_J and S_K into the J and K inputs of a JK-Type Flip-Flop (JKFF), respectively, as exemplified in Figure 2.2(e). Here, the output Q of the JKFF adopts the value of the input J if J and K disagree. By contrast, if $J = 0$ and $K = 0$, then Q retains the same value that it had in the previous clock cycle. If $J = 1$ and $K = 1$, the output Q is toggled relative to its value in the previous clock cycle. The bits in the BS S_Q obtained at the output Q of the JKFF will adopt the value 1 with the probability $P_Q = P_J/[P_J + P_K]$, which corresponds to the normalization of the probabilities P_J and P_K, as well as the approximation of P_J/P_K if $P_J \ll P_K$. In contrast to this, the division $P_Q = P_J/P_K$ may not be represented in stochastic computing if $P_J > P_K$, since this results in the probability P_Q adopting values larger than 1, which

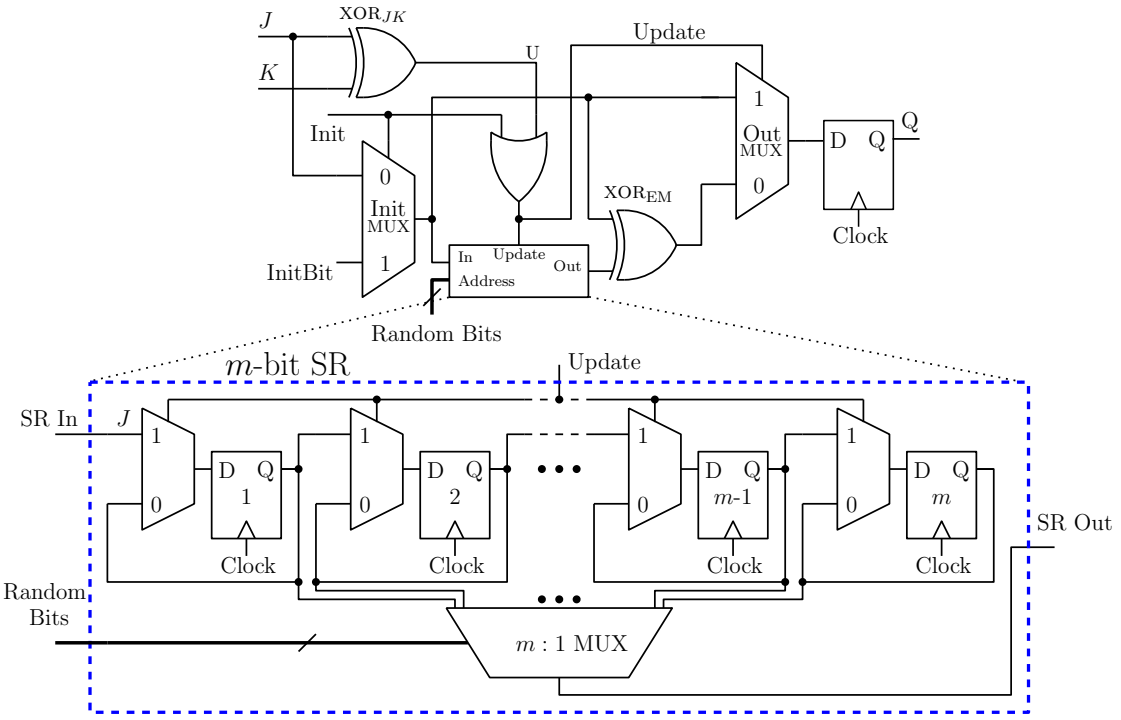


Figure 2.3: EM employed for the re-randomization of BSs in stochastic decoding.

cannot be represented using BSs in stochastic computing. In the example presented in Figure 2.2(e), the outgoing BS S_Q represents the probability $P_Q = 0.2$, whereas the resulting normalization gives $0.2/(0.2+0.7) \approx 0.22$ and the approximated division gives $0.2/0.7 \approx 0.28$. Accurate stochastic computation realizations of division, integration, square and square operations can be obtained by combining the above-described basic stochastic arithmetic circuits, albeit at the cost of an increased complexity, as detailed in [18, 65, 66].

2.3.2 Latching Problem

In the stochastic decoding of LDPC codes, JKFFs are susceptible to the latching problem, as detailed in [25]. This occurs when the bits of the BSs become stuck at 0 or 1 for several DCs, which severely affects the attainable error correction capability of the stochastic LDPC decoder [27]. This problem is particularly acute in LDPC codes with cycles in the factor graph. To elaborate further, cycles in the factor graph of the LDPC code correlate the bits of the BSs, causing the nodes of the cycle to enter into a fixed state for several DCs, as detailed in [67, 25]. As a result of this, short cycles are more susceptible to the occurrence of the latching problem. However, the careful design of LDPC codes may reduce the occurrence of the latching problem, by maximizing the length of the cycles in the factor graph, for example. A number of approaches have been proposed for overcoming the latching problem. For example, the employment of Noise-Dependent Scaling (NDS) was firstly proposed in [29]. This technique induces

switching activity in order to help stochastic LDPC decoders become unstuck when they encounter the latching problem. This is particularly useful at high channel Signal to Noise Ratios (SNRs), where the probabilities represented by the soft bits received from the channel are very close to 0 or 1, causing the BSs to become stuck at 0 or 1, respectively. In this method, the probabilities received from the channel are scaled depending on the channel's noise power spectral density N_0. Assuming a Binary Phase Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel, a bit value probability $P(b = 0)$ is converted into a scaled bit value probability $P'(b = 0)$ according to [29]

$$P'(b=0) = \frac{1}{1 + \exp\left(\frac{\eta N_0}{\psi} \cdot \log \frac{1-P(b=0)}{P(b=0)}\right)}, \tag{2.2}$$

where η and ψ are parameters that can be chosen to optimize the Bit Error Ratio (BER) performance of the stochastic LDPC decoder.

Another method of assisting stochastic LDPC decoders to become unstuck, when encountering the latching problem is replacing the JKFFs used for divisions by re-randomization units known as Edge Memories (EMs), which were first proposed in [27]. The EMs introduced in [27] consist of m-bit Shift Registers (SRs), which can be considered to behave as an m-length JKFF having a selectable output, as shown in the blue box printed using dashed lines in Figure 2.3.

Just like a JKFF, the values stored by an SR-based EM are updated if the input bits J and K differ from each other $J \neq K$, whereupon the regenerative bit J is stored in the first D-type Flip Flop (DFF) of the SR and passed to the output Q of the EM in analogy to the behavior of a JKFF, with the aid of the OutMUX. In this event, the signals U and Update of Figure 2.3 are asserted and the contents of the SR will be shifted by one position, with the oldest bit in the SR being discarded in order to ensure that only the m-most recent regenerative bits are stored. By contrast, when the J and K input bits are equal, one of the previous regenerative bits is randomly chosen from the SR and passed to the output Q of the EM, in analogy to the behavior of a JKFF. This is achieved with the aid of an m:1 MUX with pseudo-random selector bits that change in every DC, as shown in the lower part of Figure 2.3. Additionally, when $J = K = 1$, the outgoing bit of the SR is inverted with the aid of an XOR gate, before being provided to the output Q, in analogy to the behavior of a JKFF. The SR-based EM of Figure 2.3 can be initialized prior the beginning of the decoding process by means of the Init signal. During the initialization phase, the assertion of the signal Init causes the InitMUX to feed the initialization bits InitBits into the SR. The values of InitBit can be chosen to optimize the BER performance of the decoder.

2.4 Chapter Conclusions

In this chapter, we have reviewed the fundamentals of iterative decoders and aspects of their hardware implementation. We have also reviewed the fundamentals of the FX two's complement number representation, as well as stochastic computing. The concepts of stochastic computing presented in this chapter are employed in Chapters 3 to 5 in order to characterize the hardware requirements of Stochastic LDPC Decoders (SLDPCDs), Stochastic Turbo Decoder (STD) and Reduced-Latency STD (RLSTD), respectively. Meanwhile, aspects of two's complement representation are further discussed in Chapter 6 for the FX implementation of FPTDs and RCP-FPTDs.

Chapter 3

Timing-Error-Tolerant Stochastic LDPC Decoders

As mentioned in Chapter 1, Low-Density Parity-Check (LDPC) codes offer near-optimal error-correction decoding performance. This allows LDPC-coded communication schemes to employ lower transmission energies than uncoded schemes, at the cost of introducing a significant processing energy consumption during LDPC decoding. Energy management techniques such as voltage or clock scaling [68] may be employed to reduce the energy consumption of LDPC decoders. However, these techniques may induce timing errors, which occur whenever a signal does not propagate to the input of a memory before it is clocked, degrading the error correction capability of the LDPC decoder. Similarly, the clock period may be reduced below the recommended safety margins, in order to increase the processing throughput of the LDPC decoder. However, this approach also has the potential side effect of introducing timing errors. Previous works [53, 46] have shown that Fixed-Point (FX) LDPC decoders have an inherent, but partial, tolerance to timing errors. In particular, the error correction capability of LDPC decoders is not degraded significantly when timing errors occur in the Least Significant Bit (LSB) of the FX numbers. In contrast to this, if timing errors occur in the Most Significant Bit (MSB), the error correction capability is significantly degraded, as detailed in [46].

Against this background, we demonstrate that Stochastic LDPC Decoders (SLDPCDs) have an inherent tolerance to timing errors owing to power supply variations and overclocking. Moreover, we propose a modified SLDPCD design, which maintains the error

This chapter is based on the following publications.

i) **I. Perez-Andrade,** X. Zuo, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Analysis of voltage- and clock-scaling-induced timing errors in stochastic LDPC decoders," in *IEEE Wireless Communications and Networking Conf. (WCNC-2013),* April, 2013, pp. 4293-4298.

ii) X. Zuo, **I. Perez-Andrade,** R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Improving the tolerance of stochastic LDPC decoders to overclocking-induced timing errors: A tutorial and a design example," in *IEEE Access,* vol. 4, pp. 1607–1629, April, 2016

correction capability of the conventional SLDPCD design and reduces the chip area, reduces the latency and increases the throughput, albeit at the cost of a marginal increase in the energy consumption.

The rest of this chapter is organized as follows. Section 3.1 reviews the basic functionality of LDPC codes and their decoding algorithm. Section 3.2 details the hardware implementation of SLDPCDs, followed by an analysis of the various design trade-offs of the SLDPCD in Section 3.3. Section 3.4 employs extensive SPICE simulations for characterizing the causes and effects of timing errors in SLDPCDs. In addition to this, Section 3.4 characterizes the error correction capability of SLDPCDs in the presence of power supply variations when operating with different voltages and clock periods. Section 3.5 presents a modified SLDPCD design and compares its error correction capability and hardware performance with those of the conventional design. Finally, Section 3.6 concludes that the modified SLDPCD offers an improved tolerance to timing errors and an improved hardware implementation performance.

3.1 LDPC Codes

An (n, k) LDPC code uses n encoded bits to represent $k < n$ message bits and may be described by means of a sparse $(n-k) \times n$ parity-check matrix $\boldsymbol{H}$. LDPC codewords $\boldsymbol{c} = [c_i]_{i=1}^{n}$ comprising a valid sequence of n encoded bits satisfy

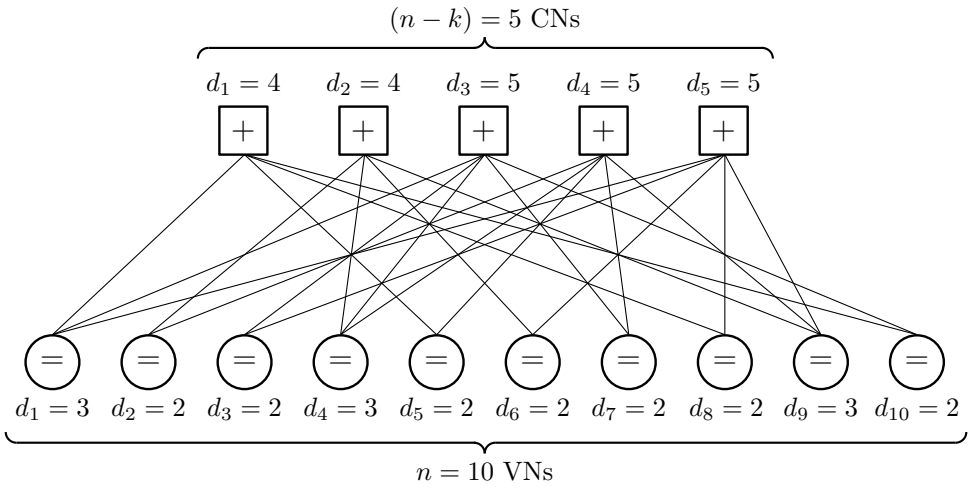
$$\boldsymbol{H}\boldsymbol{c}^T = \boldsymbol{0}, \tag{3.1}$$

where the codeword $\boldsymbol{c}$ is a $1 \times n$ sequence comprising k message bits and $(n-k)$ parity bits, $\boldsymbol{c}^T$ denotes the transpose of $\boldsymbol{c}$ and $\boldsymbol{0}$ is an $(n-k) \times 1$ null vector. Each of the $(n-k)$ rows of $\boldsymbol{H}$ represents a parity-check equation of the LDPC code. Similarly, each of the n columns of $\boldsymbol{H}$ corresponds to a bit in the LDPC-encoded codeword. As a result of this, an (n, k) LDPC code can be described by means of $(n-k)$ parity-check equations and by means of factor graphs [67], which comprise n Variable Nodes (VNs) and $(n-k)$ Check Nodes (CNs), as exemplified by the (10,5) LDPC code of Figure 3.1. In Figure 3.1, VNs are represented with a circle with an equal sign, in order to indicate that these nodes compute the probability that their incoming messages have the same value. Similarly, CNs are represented with a box with a plus sign, in order to indicate that these nodes compute the modulo-2 addition of their incoming messages.

The relationship between a parity-check matrix $\boldsymbol{H}$ and its parity-check equations is illustrated in Figure 3.1(a). Here, the parity-check equation of the j^{th} row of $\boldsymbol{H}$ corresponds to the modulo-2 addition of the i^{th} columns having the value of 1 in that particular row. Additionally, Figure 3.1(b) illustrates the corresponding factor graph of the (10,5) LDPC code of Figure 3.1(a). A non-zero element in the j^{th} row and i^{th} column of $\boldsymbol{H}$ corresponds to the presence of an edge connecting between the j^{th} CN and the i^{th}

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \Leftrightarrow \begin{array}{l} c_1 \oplus c_5 \oplus c_8 \oplus c_{10} = 0 \\ c_2 \oplus c_4 \oplus c_6 \oplus c_9 = 0 \\ c_1 \oplus c_3 \oplus c_4 \oplus c_7 \oplus c_{10} = 0 \\ c_2 \oplus c_4 \oplus c_5 \oplus c_7 \oplus c_9 = 0 \\ c_1 \oplus c_3 \oplus c_6 \oplus c_8 \oplus c_9 = 0 \end{array}$$

(a)



(b)

Figure 3.1: Parity-check matrix, parity-check equations and factor graph of a $(10, 5)$ LDPC code. (a) $\boldsymbol{H}$ matrix and parity-check equations. (b) Factor graph of $\boldsymbol{H}$. VNs are represented with a circle with equal sign and CNs are represented with a box with a plus sign.

VN, which are represented by a box with a plus sign and a circle with an equal sign, respectively. In this way, the total number of edges in the factor graph corresponds to the total number of 1's in the parity-check matrix. The number of edges attached to a particular node is referred to as the degree of the node. The degree of the i^{th} VN is equal to the total number of 1's in the i^{th} column of $\boldsymbol{H}$ and is referred to as d_i. Similarly, the number of 1's in the j^{th} row of $\boldsymbol{H}$ determines the degree of the j^{th} CN represented as d_j. An LDPC code is called regular if the degrees of all VNs are equal and if the degrees of all CNs are equal. By contrast, an irregular LDPC code comprises different degree distributions for VNs and CNs. As a result of this, the LDPC code illustrated in Figure 3.1 corresponds to an irregular code having the degree distributions $d_i \in \{2, 3\}$ and $d_j \in \{4, 5\}$.

3.1.1 Sum-Product Algorithm

Figure 3.2 shows a simplified communication scheme, where the transmitter employs an LDPC encoder and the receiver employs an LDPC decoder. Here, the factor graph of Figure 3.1(b) is represented by the interleaver box in order to facilitate the following discussions.

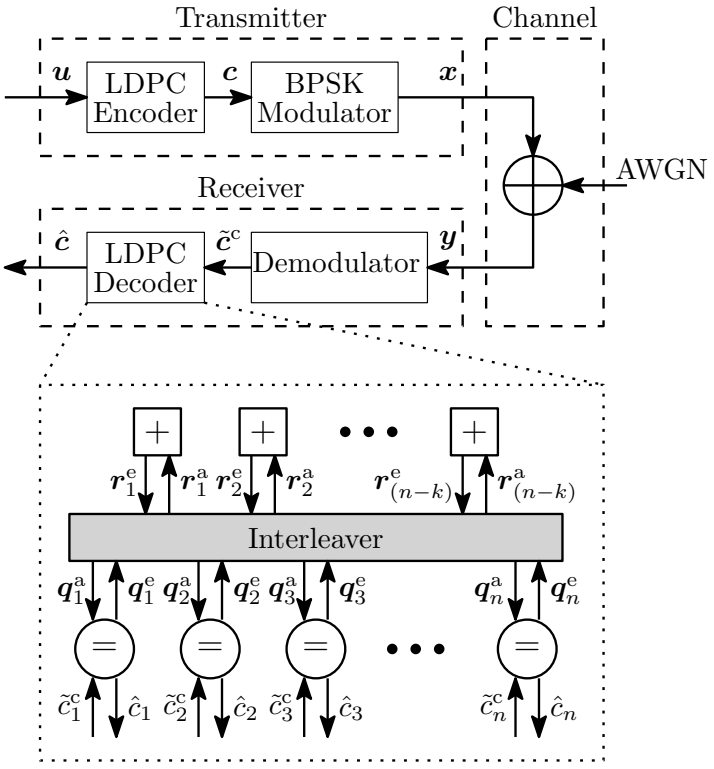


Figure 3.2: Simplified block diagram of a communication scheme employing LDPC codes.

A sequence of k information bits $\boldsymbol{u} = [u_i]_{i=1}^{k}$ can be LDPC-encoded into the n-bit encoded sequence $\boldsymbol{c} = [c_i]_{i=1}^{n}$ with the aid of a $(k \times n)$ generator matrix $\boldsymbol{G}$, such that $\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}$. Here, $\boldsymbol{G}$ is designed to produce codewords that satisfy the corresponding parity-check matrix $\boldsymbol{H}$ [69]. Following this, $\boldsymbol{c}$ is modulated by employing Binary Phase Shift Keying (BPSK) modulation to obtain the sequence of n symbols $\boldsymbol{x} = [x_i]_{i=1}^{n}$, where $x_i \in \{-1, +1\}$. Here, the modulated symbol $x_i = -1$ is used to represent the LDPC-encoded bit $c_i = 1$. Similarly, $x_i = +1$ represents $c_i = 0$. After their transmission over an Additive White Gaussian Noise (AWGN) channel, the received sequence $\boldsymbol{y} = [y_i]_{i=1}^{n}$ is soft decision demodulated [70] and the sequence of *a priori* soft bits $\tilde{\boldsymbol{c}}^{\mathrm{c}} = [\tilde{c}_i^{\mathrm{c}}]_{i=1}^{n}$ is provided to the LDPC decoder. These soft bits express not only *what* the most likely value of the corresponding bits are, but also *how* likely these bit values are. As an example of this, each soft bit $\tilde{c}_i^{\mathrm{c}}$ expresses the two probabilities $P(c_i = 0)$ and $P(c_i = 1)$. Each VN of Figure 3.2 employs the corresponding soft bit $\tilde{c}_i^{\mathrm{c}}$ to generate the sequence of d_i *extrinsic* soft bits $\boldsymbol{q}_i^{\mathrm{e}} = [q_{ij}^{\mathrm{e}}]_{j=1}^{d_i}$. Following this, the *extrinsic* soft bits $\boldsymbol{q}_i^{\mathrm{e}}$ are provided as the sequence of d_j *a priori* soft bits $\boldsymbol{r}_j^{\mathrm{a}} = [r_{ji}^{\mathrm{a}}]_{i=1}^{d_j}$ to each CN through the edges of the factor graph, which are represented by the interleaver of Figure 3.2. Each CN employs the sequence $\boldsymbol{r}_j^{\mathrm{a}}$ to provide the sequence of d_j *extrinsic* soft bits $\boldsymbol{r}_j^{\mathrm{e}} = [r_{ji}^{\mathrm{e}}]_{i=1}^{d_j}$, which are provided through the interleaver to each VN as the sequence of d_i *a priori* soft bits $\boldsymbol{q}_i^{\mathrm{a}} = [q_{ij}^{\mathrm{a}}]_{j=1}^{d_i}$. Finally, each VN employs the corresponding *a priori* soft bits $\tilde{c}_i^{\mathrm{c}}$ provided from channel and the sequence of d_i *a priori* soft bits $\boldsymbol{q}_i^{\mathrm{a}}$ provided from the CNs to estimate the decoded bit $\hat{c}_i$. This process iterates several times, before the decoded bits

$$\tilde{c}_i^{\mathrm{c}} = \frac{1}{1 + e^{2y_i/\sigma^2}} \tag{3.2}$$

$$q_{ij}^{\mathrm{e}} = \frac{\tilde{c}_i^{\mathrm{c}} \prod_{j' \in C_{i\setminus j}} q_{ij'}^{\mathrm{a}}}{\tilde{c}_i^{\mathrm{c}} \prod_{j' \in C_{i\setminus j}} q_{ij'}^{\mathrm{a}} + [1 - \tilde{c}_i^{\mathrm{c}}] \prod_{j' \in C_{i\setminus j}} (1 - q_{ij'}^{\mathrm{a}})} \tag{3.3}$$

$$r_{ji}^{\mathrm{e}} = \frac{1}{2} - \frac{1}{2} \prod_{i' \in R_{j\setminus i}} (1 - 2(1 - r_{i'j}^{\mathrm{a}})) \tag{3.4}$$

$$Q_i = \frac{\tilde{c}_i^{\mathrm{c}} \prod_{j \in C_i} q_{ij}^{\mathrm{a}}}{\tilde{c}_i^{\mathrm{c}} \prod_{j \in C_i} q_{ij}^{\mathrm{a}} + [1 - \tilde{c}_i^{\mathrm{c}}] \prod_{j \in C_i} (1 - q_{ij}^{\mathrm{a}})} \tag{3.5}$$

$$\hat{c}_i = \begin{cases} 0 & Q_i < 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{3.6}$$

are concatenated to form the decoded bit sequence $\hat{C} = [\hat{c}_i]_{i=1}^{n}$. Note each VN provides its *extrinsic* soft bits only to those CNs that are connected through edges in the factor graph. Similarly, each CN only provides its *extrinsic* soft bits to those VNs that are connected through edges in the factor graph.

VNs and CNs operate iteratively on the basis of the Sum-Product Algorithm (SPA) [67], which comprises Equations (3.2) to (3.6). During the first iteration of the SPA, VNs initialize their *extrinsic* soft bits q_{ij}^{e} according to the received channel probabilities $\tilde{c}_i^{\mathrm{c}}$ of Equation 3.2. Here σ^2 denotes the variance of the Gaussian noise of the channel. Following this, the *extrinsic* soft bits q_{ij}^{e} are provided to the CNs as the *a priori* soft bits r_{ji}^{a}. CNs employ Equation 3.4 to provide the *extrinsic* sequence r_{ji}^{e} as the *a priori* soft bits q_{ij}^{a} to the VNs. Here, $i' \in R_{j\setminus i}$ denotes the set of VNs connected to the j^{th} CN, with the i^{th} VN excluded. Following this, VNs employ Equation 3.5 to estimate the *a posteriori* probability of the decoded sequence, where $j \in C_i$ denotes the set of CNs connected to the i^{th} VN. The first iteration of the SPA is finalized with the hard decision estimation of the decoded sequence, which is performed by the VNs employing Equation 3.6. In the subsequent iterations, VNs employ Equation 3.3 to provide the *extrinsic* soft bits q_{ij}^{e}. Here, $j' \in C_{i\setminus j}$ denotes the set of CNs connected to the i^{th} VN, with the j^{th} CN excluded. This process is repeated until all parity-check equations are satisfied by $\boldsymbol{H}\hat{\boldsymbol{c}}^T = \mathbf{0}$ or until a maximum affordable number of iterations has been reached.

Note that the SPA relies on the multiplication of probabilities in the range $P \in [0, 1]$, which can lead to numerical stability issues, when the probabilities adopt very small values. In order to overcome this problem, the SPA can be operated in the logarithmic domain, where multiplications of probabilities are transformed into additions of logarithms. The resulting algorithm is referred to as Logarithmic Sum-Product Algorithm (Log-SPA) [70] and its operation is based on the employment of Logarithmic

$$L_{\tilde{c}_i^{\mathrm{c}}} = \frac{2y_i}{\sigma^2} \tag{3.8}$$

$$L_{q_{ij}^{\mathrm{e}}} = L_{\tilde{c}_i^{\mathrm{c}}} + \sum_{j' \in C_{i\setminus j}} L_{q_{ij'}^{\mathrm{a}}} \tag{3.9}$$

$$L_{r_{ji}^{\mathrm{e}}} = 2\tanh^{-1} \prod_{i' \in R_{j\setminus i}} \tanh \frac{L_{r_{i',j}^{\mathrm{a}}}}{2} \tag{3.10}$$

$$L_{Q_i} = L_{\tilde{c}_i^{\mathrm{c}}} + \sum_{j \in C_i} L_{q_{ij}^{\mathrm{a}}} \tag{3.11}$$

$$\hat{c}_i = \begin{cases} 0 & Q_i < 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.12}$$

Likelihood Ratios (LLRs), which are defined as

$$L_P = \log \frac{P(b=0)}{P(b=1)}. \tag{3.7}$$

As a result of this, Equations (3.2) to (3.6) can be expressed for the case of the Log-SPA as in Equations (3.8) to (3.12).

The complexity of the Log-SPA can be further reduced by approximating the *box plus* operation of CNs in Equation 3.10 as follows:

$$\begin{aligned} a \boxplus b &= 2\tanh^{-1}(\tanh\frac{a}{2}\tanh\frac{b}{2}) \\ &\approx \mathrm{sgn}(a)\mathrm{sgn}(b)\min(|a|,|b|). \end{aligned} \tag{3.13}$$

As a benefit of this, Equation 3.10 can be expressed as

$$L_{r_{ji}^{\mathrm{e}}} \approx \prod_{i' \in R_{j\setminus i}} \mathrm{sgn}(L_{r_{i',j}^{\mathrm{a}}}) \cdot \min_{i' \in R_{j\setminus i}} |L_{r_{i',j}^{\mathrm{a}}}|. \tag{3.14}$$

In this case, CNs operate on the basis of determining the minimum absolute value of their incoming *a priori* soft bits, while VNs operate on the basis of additions. Owing to this, the resulting algorithm is referred to as the Min-Sum Algorithm (MSA) [71],

Note that the stochastic implementation of LDPC decoders detailed in Section 3.2 is based on the SPA. However, owing to its improved numerical stability and relatively low-complexity, the Log-SPA is used as a benchmark throughout the rest of this chapter.

3.2 Stochastic Implementation of LDPC Decoders

SLDPCDs operate on the basis of Equations (3.2) to (3.6) and represent probabilities using Bernoulli Sequences (BSs) [18], which represent probabilities by the fraction of bits in the BS having the value of 1. These bits are gradually exchanged between the VNs and CNs along the edges of the factor graph, using just one bit per decoding iteration, which is referred to as Decoding Cycle (DC). Moreover, the stochastic implementation of LDPC decoders employs combinations of the stochastic arithmetic circuits presented in Section 2.3.1. More specifically, stochastic VNs convert the FX received channel probabilities of Equation 3.2 into individual bits of BSs representing $\tilde{c}_i^{\mathrm{c}}$. Following this, stochastic VNs use the stochastic representation of Equation 3.3 to combine the bits of the BSs of $\tilde{c}_i^{\mathrm{c}}$ and q_{ij}^{a}, in order to provide bits of the BSs representing the *extrinsic* soft bits q_{ij}^{e} to the stochastic CNs. Stochastic CNs employ the stochastic implementation of Equation 3.4 to combine the bits of the BSs representing r_{ji}^{a}, in order to provide BSs representing the *extrinsic* soft bits r_{ji}^{e}. Finally, stochastic VNs use the stochastic implementation of Equations (3.5) and (3.6) to combine the bits of the BSs of $\tilde{c}_i^{\mathrm{c}}$ and q_{ij}^{a}, in order to provide an estimation of the decoded bit $\hat{c}_i$. This process is repeated until all parity-check equations are satisfied or until a maximum affordable number of DCs has been reached. The following sections describe the stochastic implementation of VNs and CNs employed in the fully-parallel SLDPCD of [29].

3.2.1 Variable nodes

Stochastic VNs accept q-bit FX representations of the received channel probabilities $\tilde{c}_i^{\mathrm{c}}$ of Equation 3.2 from the demodulator and convert them into BSs. In addition to this, stochastic VNs generate BSs representing the *extrinsic* soft bits q_{ij}^{e} of Equation 3.3, compute the *a posteriori* probability Q_i of Equation 3.5 and estimate the decoded bit $\hat{c}_i$ of Equation 3.6.

The stochastic implementation of VNs is based on the observation that every node of a factor graph of an LDPC code can be decomposed into subnodes having degrees 2 and 3, with higher degree nodes resulting from the combination of these subnodes, as detailed in [67, 29]. This is exemplified in Figure 3.3, where the output corresponding to an edge of a VN having the degree of d_i is obtained by employing subVNs having degree of 2 connected in a tree structure. Owing to this, Section 3.2.1.1 presents the stochastic implementation of VNs having degree $d_i = 2$. Building on this, Section 3.2.1.2 presents the stochastic implementation of VNs having degree $d_i = 3$ and $d_i = 6$, as employed in the SLDPCD of [29].

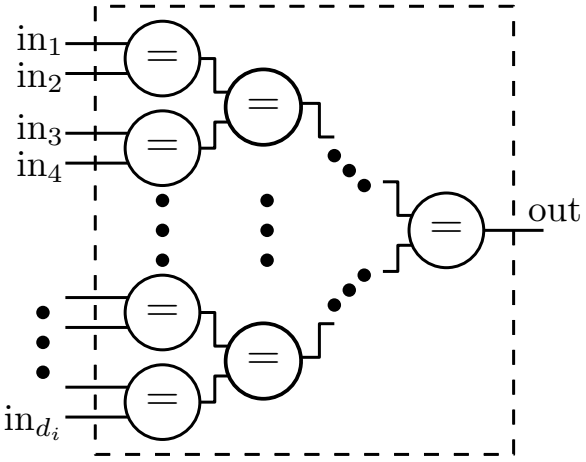


Figure 3.3: Example of the implementation of one edge of a VN having the degree of d_i employing only subnodes of degree 2.

3.2.1.1 Variable nodes having degree two

The conversion of the received channel probabilities $\tilde{c}_i^c$ into BSs is achieved with the aid of q-bit FX comparators and q-bit pseudo-random numbers, as shown in the blue box of Figure 3.4. Here, q represents the number of quantization bits employed for representing the received channel probabilities and its value can be chosen based on the best trade-off between Bit Error Ratio (BER) performance and hardware requirements, where $q = 7$ in the fully-parallel SLDPCD of [29], for example. In this structure, $\tilde{c}_i^c$ remains constant throughout the decoding process and the pseudo-random numbers change in every DC. The outgoing bit of each comparator is set to 1 if the corresponding probability is larger than the pseudo-random number and 0 otherwise. As a result of this, the majority of the bits of the BSs will adopt the value of 1, when $\tilde{c}_i^c$ adopts large values. Similarly, the majority of the bits of the BSs will adopt the value of 0, when $\tilde{c}_i^c$ adopts small values.

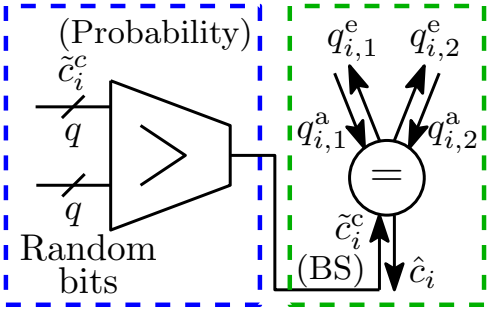


Figure 3.4: Inputs and outputs of a VN having degree of $d_i = 2$.

Following their conversion to BSs, the received channel probabilities are employed for obtaining the *extrinsic* probabilities q_{ij}^e of Equation 3.3, as shown in the blue box of Figure 3.4. As described in Section 3.1.1, each VN employs Equation 3.3 to provide the *extrinsic* probabilities q_{ij}^e to its d_i connected CNs. As a result of this, the VN having degree $d_i = 2$ of Figure 3.4 provides the *extrinsic* probabilities q_{i1}^e and q_{i2}^e. Note

that the VN provides the *extrinsic* probability q_{i2}^{e} in a similar manner to the *extrinsic* probability q_{i1}^{e}. For this reason, the following discussion considers only the stochastic implementation of q_{i1}^{e}. Equation 3.3, for the case of q_{i1}^{e}, can be expressed as

$$q_{i1}^{e} = \frac{\tilde{c}_i^{c} q_{i2}^{a}}{\tilde{c}_i^{c} q_{i2}^{a} + [1 - \tilde{c}_i^{c}][1 - q_{i2}^{a}]} = \frac{\tilde{c}_i^{c} q_{i2}^{a}}{\tilde{c}_i^{c} q_{i2}^{a} + \overline{\tilde{c}_i^{c}}\ \overline{q_{i2}^{a}}} = \frac{J}{J + K}, \tag{3.15}$$

where $\overline{\tilde{c}_i^{c}} = 1 - \tilde{c}_i^{c}$ and $\overline{q_{i2}^{a}} = 1 - q_{i2}^{a}$ are the complementary probabilities of $\tilde{c}_i^{c}$ and q_{i2}^{a}, respectively, $J = \tilde{c}_i^{c} q_{i2}^{a}$ and $K = \overline{\tilde{c}_i^{c}}\ \overline{q_{i2}^{a}}$. Owing to this, the stochastic implementation of Equation 3.15 can be performed employing NOT gates, AND gates and JK-Type Flip-Flops (JKFFs), as shown in Figure 3.5, for the complement, multiplication and division of the BSs representing the probabilities $\tilde{c}_i^{c}$ and q_{i2}^{a}, respectively, as described in Section 2.3.1.

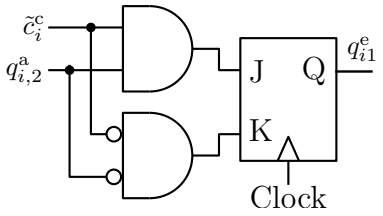


Figure 3.5: Stochastic implementation of an edge of a VN having the degree of $d_i = 2$.

The operation of the stochastic implementation of the VN of Figure 3.5 is summarized in Table 3.1.

Table 3.1: Truth table of the stochastic VN of Figure 3.5.

$\tilde{c}_i^{c}$	q_{i2}^{a}	J	K	q_{i1}^{e}
1	0	0	0	previous q_{i1}^{e}
0	1	0	0	previous q_{i1}^{e}
0	0	0	1	0
1	1	1	0	1

Here, q_{i1}^{e} retains the same value that it had in the previous DC if $\tilde{c}_i^{c} \neq q_{i2}^{a}$. By contrast, if the bits of the BS of $\tilde{c}_i^{c}$ and q_{i2}^{a} have the same value, this value is passed to the output q_{i1}^{e}. However, as mentioned in Section 2.3.2, JKFFs are susceptible to the latching problem, which can be overcome with the employment of Edge Memories (EMs). As a result of this, the JKFF of Figure 3.5 can be replaced with an EM structure similar to that of Figure 2.3 for avoiding the latching problem, as shown in Figure 3.6.

Here, the XOR_{JK} gate of Figure 2.3, which determines if J and K have the same value, is replaced with an OR gate. This is justified since the contents of the Shift Register (SR) are updated if $J = 1$ or $K = 1$, as described in Table 3.1. Likewise, according to

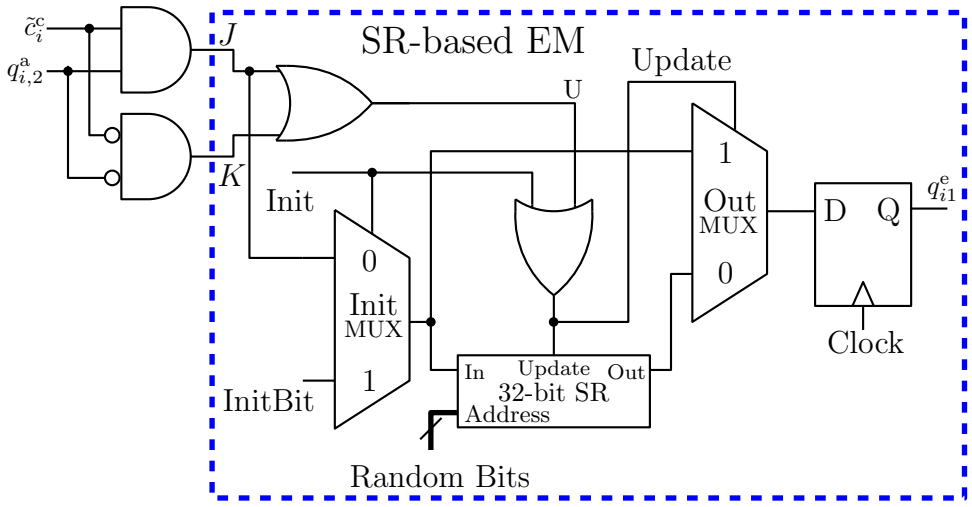


Figure 3.6: Stochastic implementation of an edge of a VN having a degree of $d_i = 2$ and employing EMs.

Table 3.1, the condition $J = K = 1$ will never occur. As a result of this, the XOR_{EM} gate employed to toggle the output of the SR of Figure 2.3 is no longer required. In analogy to the behavior of the EM of Figure 2.3, when $\tilde{c}_i^c$ and q_{i2}^a have the same value, the regenerative bit J of Figure 3.6 is stored in the first D-type Flip Flop (DFF) of the SR and passed to the DFF output q_{i1}^e, with the aid of the OutMUX, In this event, the signals U and Update of Figure 3.6 are asserted and the contents of the SR will be shifted by one position, with the oldest bit in the SR being discarded. By contrast, when $\tilde{c}_i^c \neq q_{i2}^a$, one of the previous regenerative bits is randomly chosen from the SR and passed to the DFF output q_{i1}^e. The SR-based EM of Figure 3.6 can be initialized by means of the Init signal. During the initialization phase, the assertion of the signal Init causes the InitMUX to feed the initialization bits provided by the input InitBit into the SR during successive clock cycles. Here, the values of InitBit can be chosen to optimize the BER performance of the decoder. As an example of this, BSs representing the received channel probabilities $\tilde{c}_i^c$ are employed for the initialization of the SRs in the fully-parallel SLDPCD of [29]. The length of the SR can be chosen based on the best trade-off between BER performance and hardware implementation requirements. Moreover, VNs having different degrees can employ different SR lengths, as shall be discussed in Section 3.2.1.2.

As explained above, the VN of Figure 3.4 provides the *extrinsic* soft bits q_{i1}^e and q_{i2}^e. Owing to this, the *extrinsic* soft bits q_{i2}^e must be provided by a similar structure to that of Figure 3.6, but with $\tilde{c}_i^c$ and q_{i1}^a as the input bits, as shown in the green box of Figure 3.7.

In addition to the *extrinsic* probabilities q_{ij}^e, VNs employ Equations (3.5) and (3.6) to provide the decoded bit $\hat{c}_i$. Following the same principles of the stochastic implementation of Equation 3.3, Equation 3.5 for the case of a VN having the degree of $d_i = 2$ can

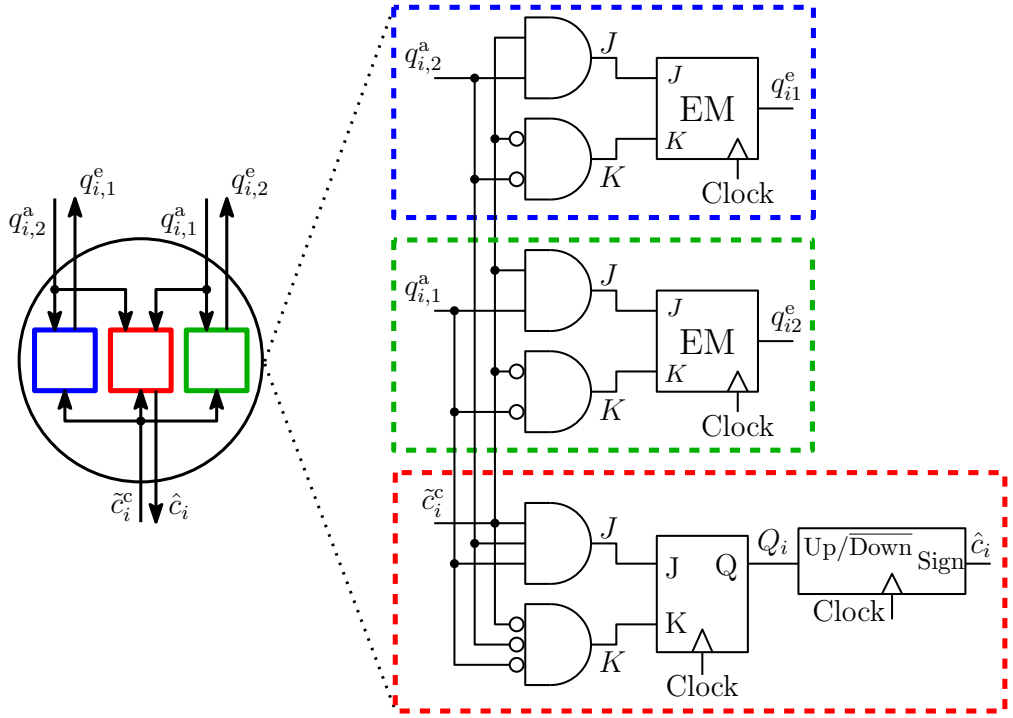


Figure 3.7: Stochastic implementation of a VN having a degree of $d_i = 2$.

be expressed as follows:

$$Q_i = \frac{\tilde{c}^{c}_{i} q^{a}_{i1} q^{a}_{i2}}{\tilde{c}^{c}_{i} q^{a}_{i1} q^{a}_{i2} + \overline{\tilde{c}^{c}_{i}}\ \overline{q^{a}_{i1}}\ \overline{q^{a}_{i2}}} = \frac{J}{J+K}, \tag{3.16}$$

where $J = \tilde{c}^{c}_{i} q^{a}_{i1} q^{a}_{i2}$ and $K = \overline{\tilde{c}^{c}_{i}}\ \overline{q^{a}_{i1}}\ \overline{q^{a}_{i2}}$. The stochastic implementation of Equation 3.16 can be performed with a structure similar to that of Figure 3.6, but employing a pair of 3-input AND gates, 3 NOT gates and a JKFF, instead of an EM, as shown in the red box of Figure 3.7. Here, the value of J will be stored in the JKFF and passed to Q_i, if $J \neq K$. By contrast, if $J = K$, Q_i will adopt the value that it had in the previous DC. Additionally, the estimation of the decoded bit $\hat{c}_i$ of Equation 3.6 can be performed with a signed up/down saturated counter, as shown in the red box of Figure 3.7. The up/down counter increments its value if the BS representing Q_i takes the value of 1 and decreases its value otherwise. The counter saturates its value if either the maximum or the minimum count value has been reached. In the SLDPCD of [29], a 4-bit up/down counter with a maximum value of +7 and a minimum value of -8 is employed. The estimation of the decoded bit $\hat{c}_i$ is performed in each DC by considering the sign bit of the saturated counter. In this way, if the value of the counter is ≥ 0, the estimated decoded bit is $\hat{c}_i = 0$ and 1 otherwise. Note that the JKFF delays the estimation of the decoded bit $\hat{c}_i$ by one DC, owing to the counter updating its value one DC after Q_i. However, this does not degrade the error correction capability or the hardware implementation performance of the decoder, as we will demonstrate in Section 3.3.

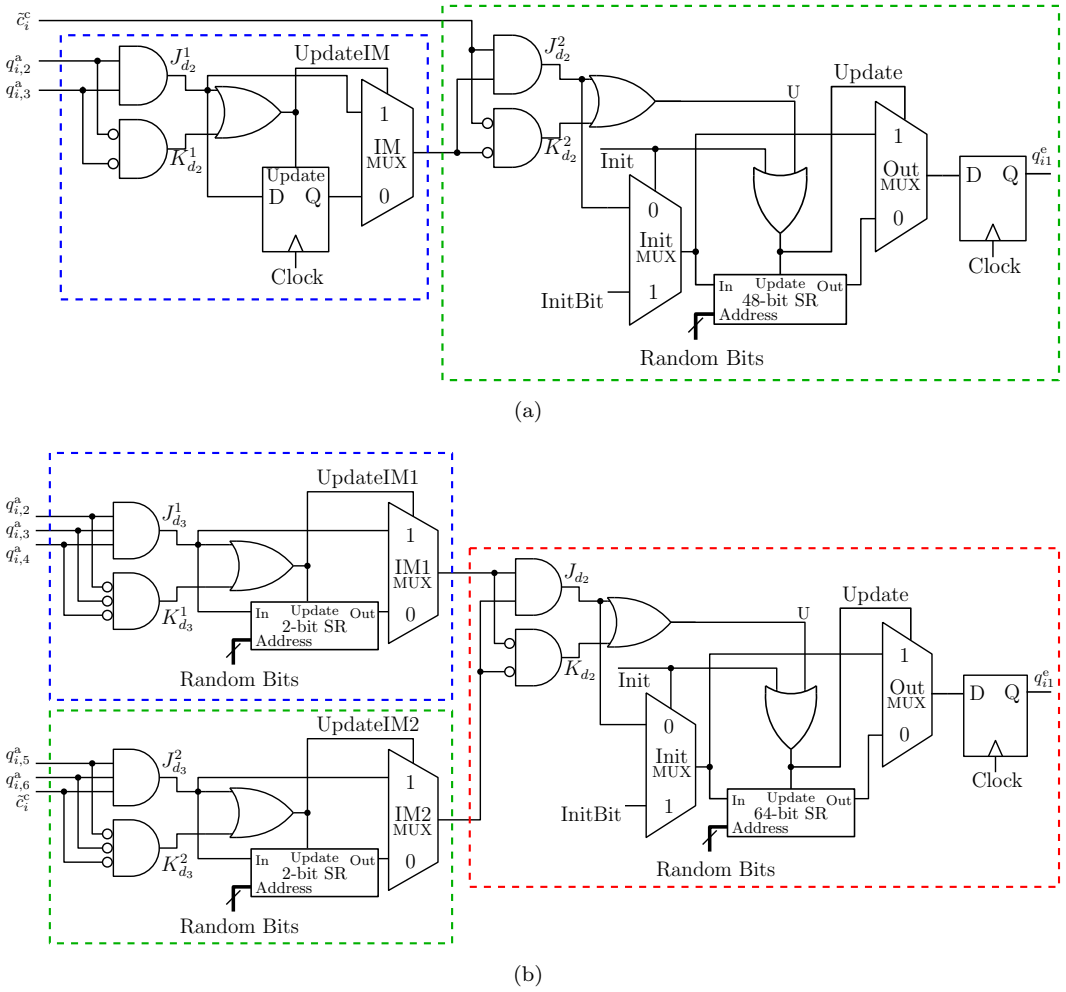


Figure 3.8: Stochastic implementation of VNs having different degrees. Here, only one output of each VN is shown. (a) $d_i = 3$. (b) $d_i = 6$.

3.2.1.2 Variable nodes having higher degrees

The same principle described in Section 3.2.1.1 can be applied to VNs having different degrees. To elaborate further, the stochastic implementation of Equation 3.3 for a VN having a degree d_i can be implemented with the aid of two d_i-input AND gates, d_i NOT and d_i EM structures. Furthermore, in the SLDPCD of [29], VNs having a degree $d_i = 3$ and $d_i = 6$ are built by employing VNs having lower degrees. More specifically, VNs having a degree $d_i = 3$ employ two subVNs having a degree $d_i = 2$, as shown in Figure 3.8(a). Here, a 48-bit SR is employed in the EM of the subVN shown in the green box of Figure 3.8(a). Likewise, VNs having a degree $d_i = 6$ employ two subVNs having a degree $d_i = 3$, as well as a VN having a degree $d_i = 2$ with an EM employing a 64-bit SR, as shown in Figure 3.8(b).

A scaled version of the EMs of Figure 3.6 is employed for the initial stages of VNs having degree $d_i \geq 3$. These structures are referred to as Internal Memories (IMs) and their operation follows the same principle as that of the EMs described in Section 3.2.1.1.

More specifically, in the subVN degree $d_i = 2$ shown in the blue box of Figure 3.8(a), the 32-bit SR of the VN of Figure 3.6 is replaced with an IM consisting of a single DFF. In analogy to the operation of the EM discussed in Section 3.2.1.1, the DFF of the IM updates its contents when $J^1_{d_2} \neq K^1_{d_2}$, whereupon the subVN provides the value of J_{d_2} to the following subVN, shown in the green box of Figure 3.8(a). Additionally, if $J^1_{d_2} = K^1_{d_2}$, the value that the DFF had in the previous DC is provided to the following subVN. The subVN of the green box of Figure 3.8(a) employs a 48-bit SR and its operation follows the same principle of that of the VN described in Section 3.2.1.1.

In the VN degree $d_i = 6$ of Figure 3.8(b), the two subVNs having the degree $d_i = 3$, employ IMs comprising two-bit SRs, as shown in the blue and green boxes. The operation of these subVNs follows the same principle described above for the operation of the subVNs having the degree $d_i = 2$ of Figure 3.8(a). Similarly, the subVN having the degree $d_i = 2$ shown in the red box of Figure 3.8(b) employs a 64-bit SR and its operation follows the same principle as that of the VN described in Section 3.2.1.1.

Note that the authors of [30, 33] employ enhanced implementations of EMs for VNs, which significantly improve the hardware efficiency of SLDPCDs. However, these implementations rely on FX numbers, which do not benefit from the low-complexity and inherent timing error tolerance. Owing to this, we limit our discussions to the EM structures presented in the fully-parallel SLDPCD of [29].

3.2.2 Parity-check nodes

In the following discussion, the stochastic implementation of CNs is described for the case of a CN having the degree $d_j = 3$, as shown in Figure 3.9. The stochastic implementation of CNs having different degrees can be performed following the same principles.

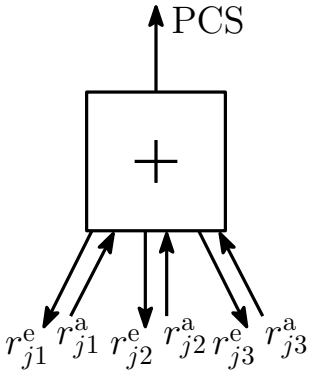


Figure 3.9: Inputs and outputs of a CN having the degree of $d_j = 3$.

As described in Section 3.1.1, each CN employs Equation 3.4 to provide the *extrinsic* probabilities r^{e}_{ji} to its d_j connected VNs. In addition to this, in the fully-parallel

SLDPCD of [29], each CN provides a Parity-Check Satisfied (PCS) signal, which indicates that the parity-check equation of that particular CN has been satisfied. As a result of this, the CN of Figure 3.9 provides the *extrinsic* soft bits r^{e}_{j1}, r^{e}_{j2} and r^{e}_{j3} and the signal PCS. Note that the CN provides the *extrinsic* soft bits r^{e}_{j2} and r^{e}_{j3} in a similar manner to the *extrinsic* soft bits r^{e}_{j1}. For this reason, the following discussion considers only the stochastic implementation of r^{e}_{j1}. Equation 3.4 for the case of r^{e}_{j1} can be expressed as follows:

$$\begin{aligned} r^{e}_{j1} &= \frac{1}{2} - \frac{1}{2}\left[1 - 2(1 - r^{a}_{j2})\right]\left[1 - 2(1 - r^{a}_{j3})\right] \\ &= r^{a}_{j2}[1 - r^{a}_{j3}] + r^{a}_{j3}[1 - r^{a}_{j2}] \\ &= r^{a}_{j2}\overline{r^{a}_{j3}} + r^{a}_{j3}\overline{r^{a}_{j2}}. \end{aligned} \tag{3.17}$$

Equation 3.17 involves the multiplication, addition and complement of BSs represented probabilities. However, Equation 3.17 also corresponds to the logical XOR operation of the BSs representing the probabilities r^{a}_{j2} and r^{a}_{j3}, which is expressed as

$$r^{e}_{j1} = r^{a}_{j2} \oplus r^{a}_{j3}. \tag{3.18}$$

Owing to this, the stochastic implementation of Equation 3.18 is shown in Figure 3.10(a).

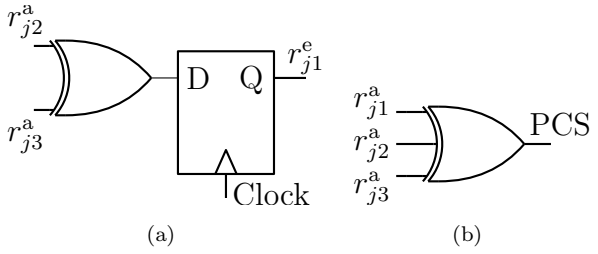


Figure 3.10: Stochastic implementation of CNs having the degree $d_j = 3$: (a) One edge of a CN. (b) Parity-check satisfied.

As mentioned in Section 3.1, the parity-check equation of a CN is computed with modulo-2 additions of the *a priori* probabilities provided by the connected VNs. Here, the stochastic implementation of a CN having the degree d_j employs a d_j-input XOR gate for the computation of the signal PCS. As a result of this, stochastic CNs having the degree d_j employ one d_j-input XOR gate and d_j XOR gates having $(d_j - 1)$-inputs for providing the *extrinsic* probabilities $r^{e}_{j,i}$ and the PCS signal, respectively. However, the complexity of the stochastic CNs can be reduced by combining Figures 3.10(a) and 3.10(b), Here, the signal PCS is XORed with the i^{th} incoming *a priori* bit for providing the i^{th} outgoing *extrinsic* bit. By doing this, the $(d_j - 1)$-input XOR gates can be replaced with 2-input XOR gates, as shown in Figure 3.11. If the PCS signal of all CNs is asserted in the same DC, the SLDPCD determines that all parity-check equations of the LDPC code have

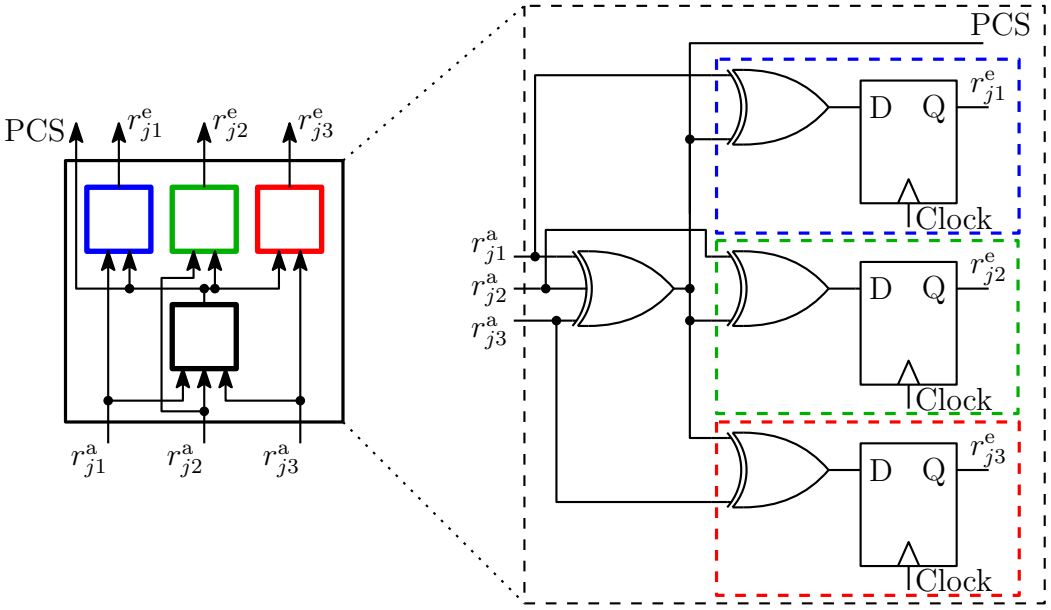


Figure 3.11: Stochastic implementation of a CN having a degree of $d_j = 3$.

been satisfied, whereupon the decoding process is stopped and VNs output the decoded bits $\hat{\boldsymbol{c}}$.

3.3 Hardware Implementation

Table 3.2 characterizes the hardware implementation of the SLDPCD presented in Section 3.2 in terms of chip area, minimum clock period and average energy consumption per DC of the (1056,528) WiMAX SLDPCD presented in [29], when using ST 90 nm technology. Here, the average energy consumption is presented for the case of the individual VNs and CNs and for the accumulated total according to the degree distribution of the LDPC code. Similarly, Table 3.2 provides the total average energy consumption of the SLDPCD per DC.

The results of area of Table 3.2 are obtained from the physical synthesis using automatic place and route of the individual nodes of the above mentioned SLDPCD using Cadence's Encounter [72]. Similarly, the results of energy consumption of Table 3.2 are obtained from SPICE [73] simulation of the individual nodes of the above mentioned SLDPCD, for the cases where the supply voltage is set to either 1.0 V or 0.8 V. The critical clock period is obtained by performing a static timing analysis of the SLDPCD using Synopsys PrimeTime [73].

In addition to the results presented in Table 3.2, Figure 3.12 presents the BER decoding performance of a SLDPCD implementation of the (1056,528) IEEE 802.16e (WiMAX) LDPC decoder, assuming BPSK transmission over an AWGN channel, when allowing a maximum of 10^4 DCs and when Noise-Dependent Scaling (NDS) with $\eta = 1$ and

Table 3.2: Hardware implementation performance of the SLDPCD.

Node		Area	V_{DD}	T_{clk}	Energy per DC	
					per node	all nodes
VN	$d_i = 2$ (484×)	1290 μm^2 (294 gates)	1.0 V	618 ps	1.50 pJ	726 pJ
			0.8 V	959 ps	0.92 pJ	445 pJ
	$d_i = 3$ (352×)	1909 μm^2 (435 gates)	1.0 V	716 ps	3.27 pJ	1151 pJ
			0.8 V	1120 ps	2.07 pJ	729 pJ
	$d_i = 6$ (220×)	2539 μm^2 (578 gates)	1.0 V	**745 ps**	8.15 pJ	1793 pJ
			0.8 V	**1160 ps**	5.09 pJ	1120 pJ
CN	$d_j = 6$ (352×)	222 μm^2 (51 gates)	1.0 V	318 ps	0.30 pJ	105 pJ
			0.8 V	505 ps	0.19 pJ	67 pJ
	$d_i = 7$ (176×)	257 μm^2 (59 gates)	1.0 V	370 ps	0.33 pJ	58 pJ
			0.8 V	560 ps	0.20 pJ	35 pJ
TOTAL		1.98 mm^2 (450×10^3 gates)	1.0 V	745 ps	N/A	3833 pJ
			0.8 V	1160 ps	N/A	2396 pJ

$\psi = 2$ is employed, as described in Section 2.3.2. C++ simulations were used to obtain the BER performance of the SLDPCD. More specifically, we simulate the stochastic VNs shown in Figures 3.7 and 3.8, as well as stochastic CNs with structures similar to that of Figure 3.11, for the cases of CNs having the degree of $d_j \in \{6, 7\}$. In each DC, stochastic VNs and stochastic CNs process and exchange bits of the BSs, until all parity-check equations are satisfied or 10^4 DCs has been reached. Figure 3.12 compare the BER of the SLDPCD with that of the Floating-Point (FP) Log-SPA LDPC decoder as a benchmark. As shown in Figure 3.12, the BER of the SLDPCD is very similar to that of the FP implementation. Note that an error floor is manifested at a BER $< 10^{-6}$, since our simulations employ early stopping to halt the iterative decoding process as soon as the corresponding degree of confidence is attained for the decoded bits. In Section 3.4, we will consider the effects that timing errors impose in the BER performance of the SLDPCD.

C++ simulations of the SLDPCD were used to measure the average number of DCs required for successfully decoding a frame, when employing early stopping and allowing a maximum of 10^4 DCs, as shown in Figure 3.13. The critical clock periods of Table 3.2 and the average number of DCs of Figure 3.13 were employed to obtain the latency of the SLDPCD shown in Figure 3.14, where Latency = $DC_{avg} \times T_{clk}$. The uncoded throughput, also known as information throughput, of Figure 3.15 was obtained according to The energy consumption per decoded bit of Figure 3.16 was quantified by using the average number of DCs of Figure 3.13 and the average energy consumption per DC of Table 3.2, where

Figures 3.14 to 3.16 show the effect that two different supply voltages have on the performance of the SLDPCD. More specifically, the SLDPCD exhibits reduced latencies and increased throughputs, when operated at V_{DD} =1.0 V instead of at 0.8 V, albeit at the cost of an increased energy consumption. Moreover, Figure 3.15 demonstrates that

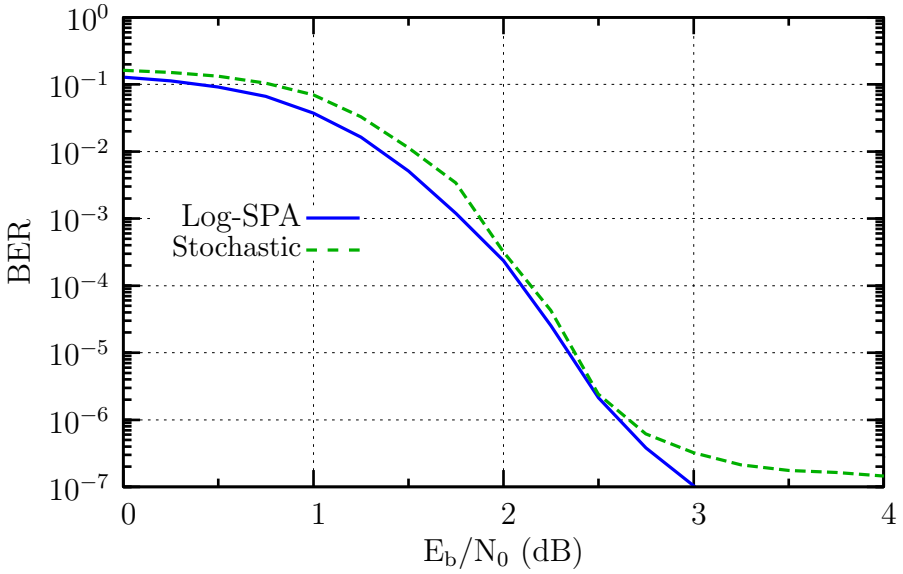


Figure 3.12: BER of the (1056,528) SLDPCD.

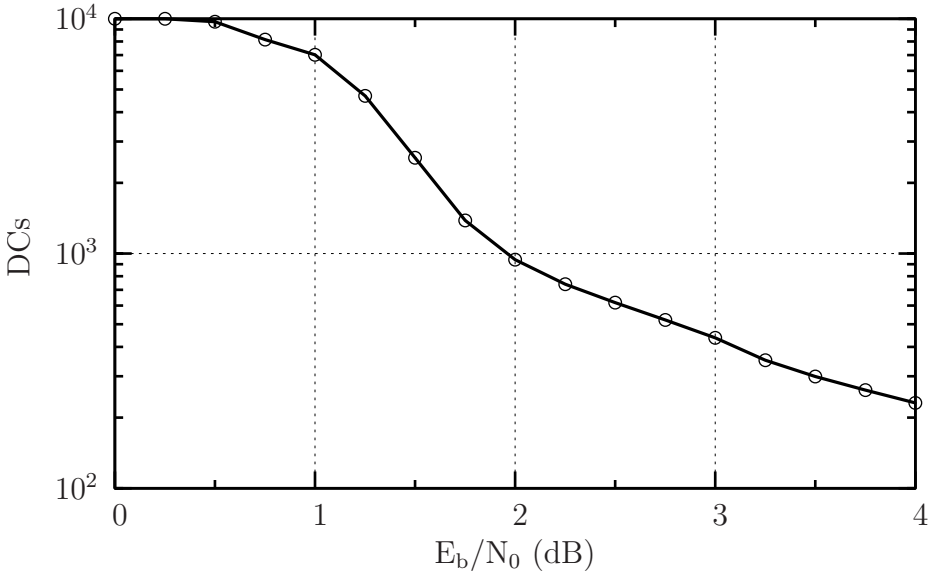


Figure 3.13: Average number of DCs for successfully decoding a frame for the SLDPCD.

the SLDPCD may achieve throughputs in the order of Gbps, when operated at V_{DD} =1.0 V for $E_b/N_0 > 2.25$ dB or V_{DD} =0.8 V for $E_b/N_0 \geq 3.0$ dB.

3.4 Error Correction Capabilities in the Presence of Timing Errors

This section characterizes the error correction capabilities of the SLDPCD in the presence of timing errors, when it operates with different voltages and clock periods. This

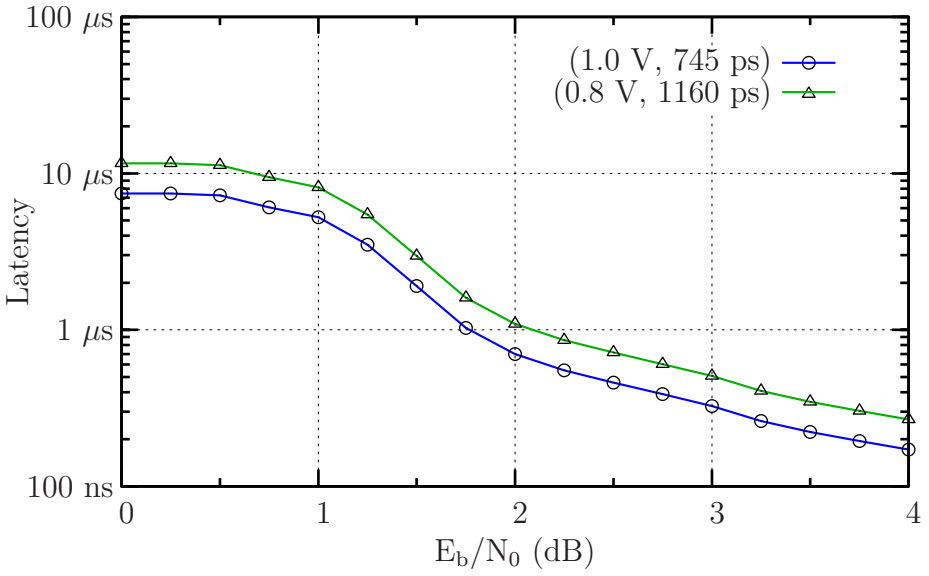


Figure 3.14: Latency of the SLDPCD.

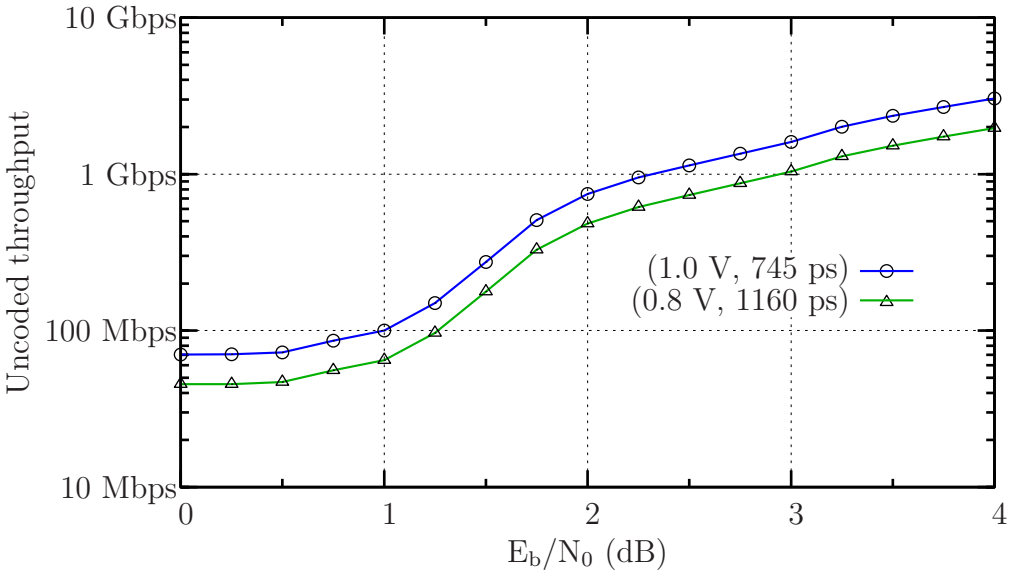


Figure 3.15: Uncoded throughput of the SLDPCD.

is achieved by performing a timing analysis for characterizing the specific causes end effects of timing errors in the SLDPCD.

3.4.1 Timing error analysis of the stochastic LDPC decoder

Timing errors in synchronous systems occur when the propagation delay t_p of the signal path p exceeds the clock period T_{clk}. In this case, an incorrect bit value will be clocked into the corresponding DFFs. This occurs when the power supply voltage is reduced, owing to power supply noise or for the sake of reducing the energy consumption of the system, as well as when overclocking is employed for the sake of increasing the throughput. The prevalence of timing errors when the supply voltage V_{DD} is reduced

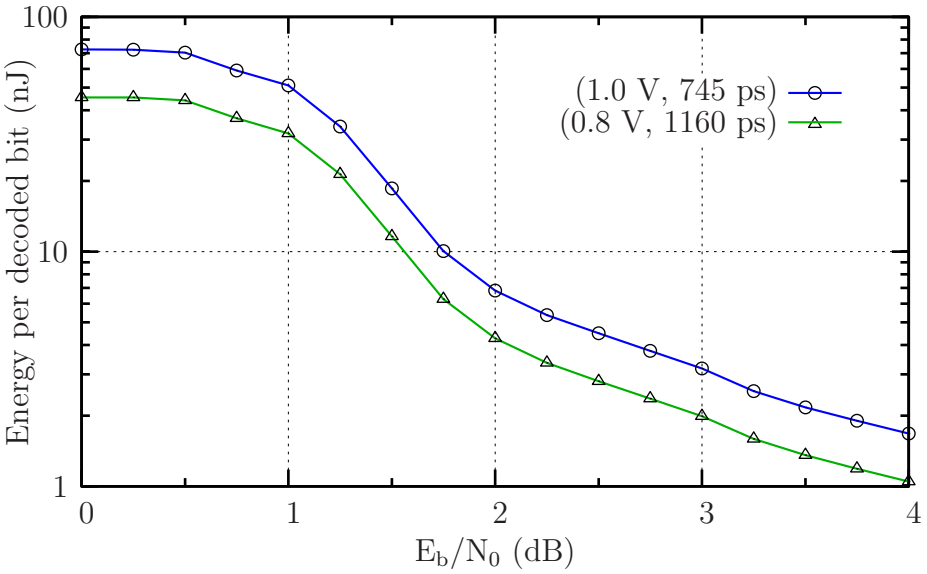


Figure 3.16: Energy consumption per decoded bit of the SLDPCD.

and the clock period T_{clk} is not adjusted accordingly, owing to voltage scaling or to power supply noise, is a decreasing quadratic function of the nominal supply voltage. In Section 3.4.1.1, we characterize the signal propagation delays t_p within stochastic VNs and CNs having various degrees and nominal supply voltages. Following this, Section 3.4.1.2 details the conditions and effects of timing errors in VNs having a degree of $d_i = 3$, as a particular example. Note that the timing analysis of VNs having degree $d_i = 2$ and $d_i = 6$ can be performed following the same principles and so will only be summarized.

3.4.1.1 Propagation delays

In this section, we characterize the nominal signal propagation delays t_p of stochastic VNs and CNs having degrees of $d_i \in \{2, 3, 6\}$ and $d_j \in \{6, 7\}$, respectively, as employed in the $(1056, 528)$ LDPC code defined in the IEEE 802.16e (WiMAX) standard [4]. We model the propagation delay t_p of a signal according to its end DFF. In this context, a path delay is comprised of the Clock-to-Q delay of the initial DFF, The propagation delay of a logic gate depends on the previous and current values of its inputs as well as its supply voltage and external factors such as temperature and fabrication process variations [74]. Hence, the cumulative propagation delay of a path varies between consecutive clock cycles [75].

A stochastic VN having a degree of $d_i = 3$ comprises 9 input bits, each of them corresponding to one bit of a BS. More specifically, 3 input bits correspond to BSs provided by CNs and 6 input bits corresponds to pseudo-random bits employed by the 48-bit SR, as shown in Figure 3.8(a) and again with more detail in Figure 3.17. Note that the Init and InitBit inputs are not considered in this analysis since they are only relevant during

the initialization stage of the SLDPCD. In addition to this, a stochastic VN employs 48 DFFs in the SR, 1 DFF in the IM and 1 output DFF. Owing to this, there are a total of $2^{9+50} \approx 6 \times 10^{17}$ combinations of current states and about 6×10^{34} combinations of current and previous states in a VN having a degree of $d_i = 3$. Similarly, a VN having a degree of $d_i = 6$ comprises $\approx 10^{50}$ combinations of current and previous states. As a result of this, it is not feasible to perform a timing analysis that considers all possible combinations of current inputs and current and previous states. In analogy to the methodology proposed in [54], and This is justified because the Multiplexer (MUX) selector signals determine how signals are propagated through the stochastic VN circuit, as detailed in [54]. More specifically, when a MUX selector signal remains constant during consecutive clock cycles, the MUX propagation delay is governed by the selected signal. By contrast, if the selector signal is toggled, the propagation delay is governed by the maximum delay of the MUX selector signal and the selected signal. To further simplify our analysis, we do not consider all combinations of IMMUX and OutMUX selector signals. Instead, we focus on those that were found to exceed the maximum CN propagation delay t_{CN}. This is because we assume that the degree of power supply variations and overclocking is limited, in order to avoid timing errors in the CNs, which were found to break down the inherent fault tolerance of the SLDPCD in our experiments.

Table 3.3 summarizes the maximum propagation delays Here, 'Input' refers to the input

Table 3.3: Maximum propagation delays according to pairs of starting and ending points in stochastic VNs operating at $\mathrm{V_{DD}} = 1.0$ V.

Degree	Starting point	Ending point	Combination		Delay (t_p)
			IMMUX (S_4)	OutMUX (S_5)	
$d_i = 6$	Input	Output/SR	$1 \rightarrow 1$	$1 \rightarrow 0$	**745 ps** (t_1)
		Output/SR	toggle	$0 \rightarrow 1$	**576 ps** (t_6)
		IM (D)	$0 \rightarrow 1$	any	245 ps
	IM (Q)	Output/SR	$1 \rightarrow 1$	$1 \rightarrow 0$	**552 ps** (t_7)
	SR (Q)	Output (D)	any	$0 \rightarrow 0$	**642 ps** (t_3)
$d_i = 3$	Input	Output/SR	toggle	$1 \rightarrow 0$	**716 ps** (t_2)
		Output/SR	toggle	$0 \rightarrow 1$	**455 ps** (t_8)
		IM (D)	$0 \rightarrow 1$	any	218 ps
	IM (Q)	Output/SR	any	$1 \rightarrow 0$	**382 ps** (t_{10})
	SR (Q)	Output (D)	any	$0 \rightarrow 0$	**599 ps** (t_5)
$d_i = 2$	Input	Output/SR	-	$1 \rightarrow 0$	**618 ps** (t_4)
		Output/SR		$0 \rightarrow 1$	354 ps
	SR (Q)	Output (D)		$0 \rightarrow 0$	**426 ps** (t_9)

of the corresponding stochastic VN; 'IM (Q)' refers to the output Q of the corresponding IM; 'SR (Q)' refers to the output Q of any of the DFFs of the corresponding SR; 'Output/SR' refers to both the input D of the output DFF of the VN and to the input D of the first DFF of the corresponding SR; 'Output (D)' refers to the input D of the output DFF of the corresponding VN; and 'IM (D)' refers to the input D of the IM of the

corresponding VN. These start and end points are shown in Figure 3.17 for illustrative purposes. The fourth and fifth columns of Table 3.3 describe the combination of the the IMMUX and OutMUX selector signals that cause the corresponding delay t_p, where '$0 \rightarrow 1$' refers to the case of a 0 to 1 transition of the corresponding signal in two consecutive DCs. Similarly, '$1 \rightarrow 0$' represents a 1 to 0 transition in two consecutive DCs, while '$0 \rightarrow 0$' and '$1 \rightarrow 1$' represent a constant value of 0 and 1 between the consecutive DCs, respectively. Finally, 'any' indicates that the particular propagation delay is not dependent on the corresponding MUX selector signal. Note that stochastic VNs having the degree of $d_i = 6$ employ two sets of IMs with MUX selector signals IM1MUX and IM2MUX, respectively. However, owing to the symmetry in the circuit, the maximum propagation delay of these VNs may be caused when either IM1MUX or IM2MUX adopt specific values. As a result of this, our analysis considers only one of the two MUX selector signals of stochastic VNs having the degree of $d_i = 6$, which is simply denoted as IMMUX in the fourth column of Table 3.3. Moreover, stochastic VNs having the degree of $d_i = 2$ do not employ IMs. Propagation delays in bold, which are labeled as $[t_p]_{p=1}^{10}$, correspond to those paths that exceed the maximum propagation delay $t_{\text{CN}} = 370$ ps of CNs having the degree of $d_j = 7$.

Figure 3.17 exemplifies a possible critical path starting in the input of the VN and ending in the output DFF as well as in the first DFF of the SR. Here, a transition from 1 to 0 of the OutMUX selector signal S_5 and a constant value of 1 of the IMMUX selector signal S_4 may result in a propagation delay $t_1 = 745$ ps, as described in Table 3.3.

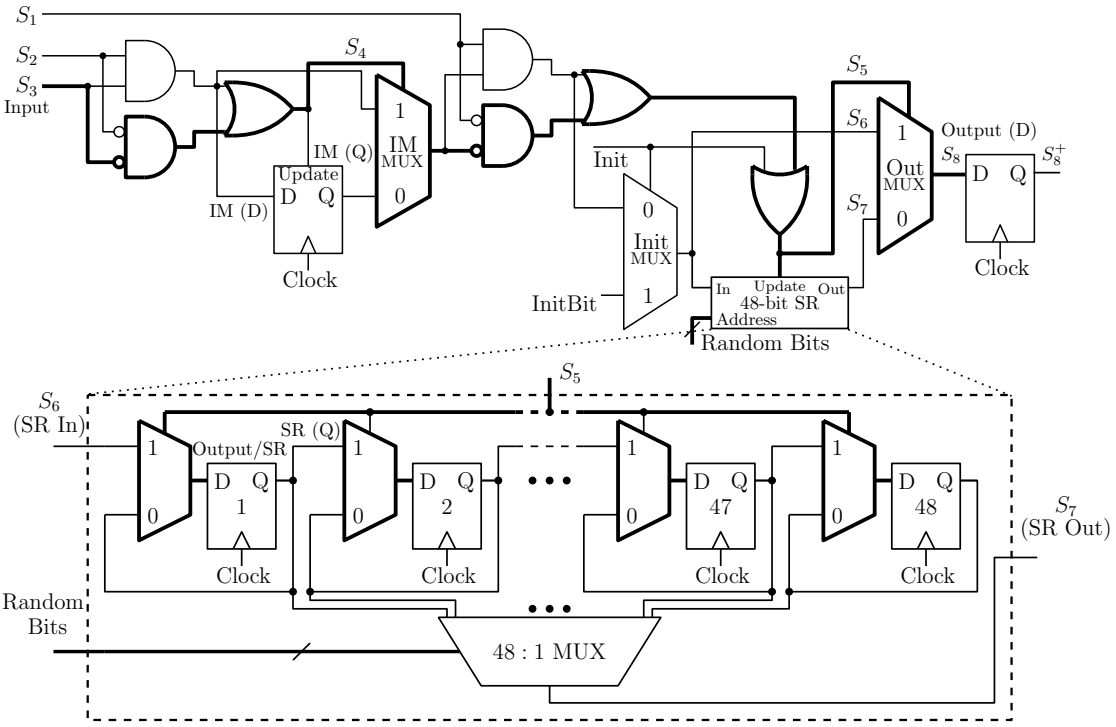


Figure 3.17: A critical path in a stochastic VN $d_i = 3$.

As described above, Table 3.3 shows the combination of IMMUX and OutMUX selector signals that triggers the maximum propagation delay for each pair of starting and ending point in stochastic VNs. However, in our analysis of Section 3.4.1.2, we also consider all other combinations of IMMUX and OutMUX selector signals that cause a propagation delay larger than the propagation delay t_{CN} of a CN having the degree of $d_j = 7$.

3.4.1.2 Causes and effects of timing errors

As described above, propagation delays are a function of the supply voltage, which can vary from one clock cycle to the next due to the switching activity of registers, temperature variation or other sources of noise, such as IR-drop, $L \cdot d_i/d_t$ noise, crosstalk, electrostatic discharges, particle strikes, switching noise and fabrication process variations [12, 13, 14, 15, 16], among other causes. Owing to this, the supply voltage V_{DD} in consecutive clock cycles can be assumed to have a Gaussian distribution having a mean of μ, which represents the nominal supply voltage, and a standard deviation σ, related to the power supply variations [76, 77].

Figure 3.18 presents the propagation delay for each of the highlighted t_p of Table 3.3 and for t_{CN}, as a function of the supply voltage. These results were obtained using SPICE simulations of the individual VNs and CNs of the SLDPCD, implemented using ST 90 nm technology. Depending on the supply voltage in each clock cycle, Table 3.3 and Figure 3.18 can be employed to determine the corresponding propagation delay t_p of each considered path p. Furthermore, Figure 3.18 can be employed to determine the occurrence of timing errors, when the clock period T_{clk} adopts a particular value. For example, Figure 3.18 illustrates that when T_{clk} =750 ps, no timing errors occur if the supply voltage in the current clock cycle is 1.0 V. However, in clock cycles where the supply voltage drops to 0.95 V, timing errors occur in paths $p = 1$ and $p = 2$ of the SLDPCD.

The following discussion exemplifies the causes and effects of timing errors for the case of a stochastic VNs having a degree of $d_i = 3$, as shown in Figure 3.19. Moreover, in order to facilitate the following explanations, Table 3.4 summarizes the effect of each timing error shown in Figure 3.19.

Table 3.4: Summary of the effects of timing errors in stochastic VNs.

TET	Update state of SR	Update of output DFF	
		Error-free case	Error case
I	Not correctly updated	current SR output	old SR output
IIa	Fails to get updated	current SR input	current SR output
IIb	Erroneously updated	current SR output	current SR input
IIIa	Fails to get updated	current SR input	old SR output
IIIb	Erroneously updated	current SR output	current SR input

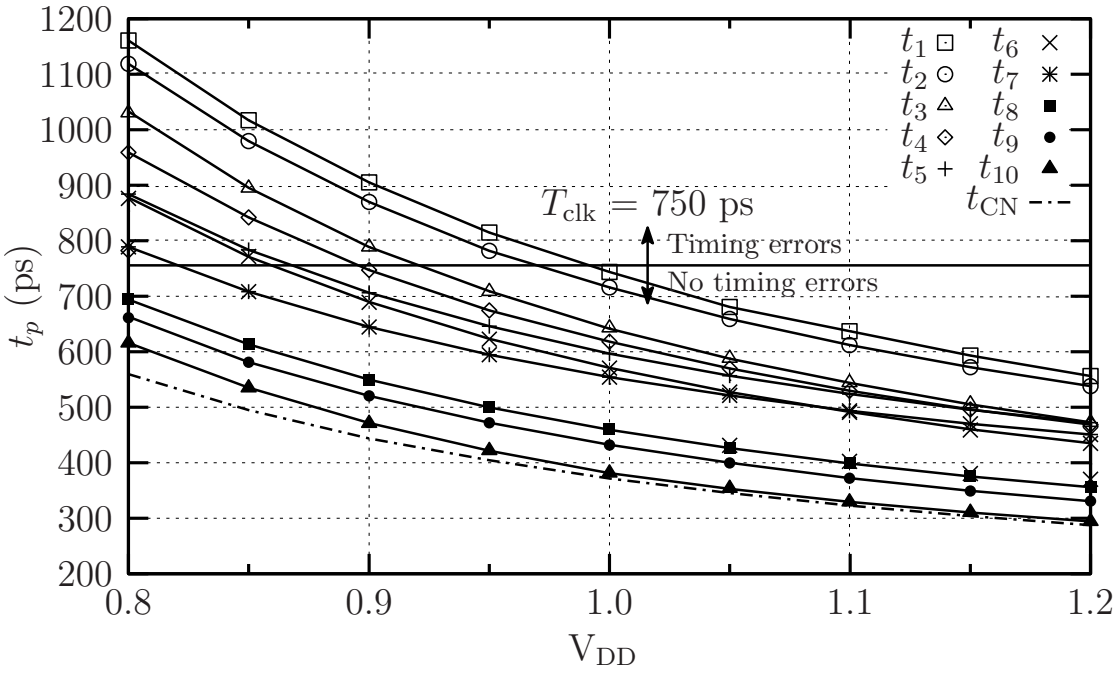


Figure 3.18: Propagation delays as a function of supply voltage for the VNs and CNs employed in the SLDPCD implemented in ST 90 nm technology. Here, t_p denotes the propagation delay of the path p.

The Timing Error Type (TET) I of Figure 3.19 occurs if the OutMUX selector signal S_5 correctly propagates before the clock edge, but S_7 arrives late. In this case, if $S_5 = 1$, the OutMUX will not select the late S_7. However, if $S_5 = 0$, the OutMUX will select the late S_7 inflicting a timing error. Therefore, $S_5 = 0$ is a condition for a TET I to occur. The effect of this error is that the value that is clocked into the output DFF of the VN corresponds to value of the previous clock cycle S_7^- instead of clocking its current value S_7, as described in the third and fourth columns of Table 3.4. In this type of error, the SR DFFs are not erroneously updated, since S_7 is not an input to the SR.

The TET II of Figure 3.19 occurs when S_5 is toggled and arrives after the clock edge, but S_7 arrives on time. In this case, the previous value of the OutMUX selector signal, S_5^-, determines the updating of the SR and the signal that is clocked into the output DFF of the VN. Owing to this, TET II can be categorized into TET IIa and TET IIb, depending on the value of S_5^-. A TET IIa occurs when the selector signal OutMUX is toggled according to $S_5^- = 0$ and $S_5 = 1$. The effect of this fault is that the SR fails to get updated and the value of S_7 will be clocked into the output DFF of the VN, rather than S_6, as described in Table 3.4. By contrast, a TET IIb occurs when the OutMUX selector signal is toggled according to $S_5^- = 1$ and $S_5 = 0$. In this error, S_6 is erroneously clocked into the SR as well as into the output DFF.

Finally, the TET III of Figure 3.19 occurs when S_5 and S_7 arrive after the clock edge and S_5 has been toggled. In a similar manner as in the TET II, S_5^- controls the updating of the SR DFFs and the signal that is clocked into the output DFF of the VN. Therefore,

TET III can also be categorized into TET IIIa and TET IIIb, depending on the value of S_5^-. More specifically, TET IIIa occurs when $S_5^- = 0$ and $S_5 = 1$. The effect of this fault is that the SR fails to get updated and since S_7 is late, S_7^- is clocked into the output DFF, rather than S_6, as described in Table 3.4. By contrast, a TET IIIb occurs when $S_5^- = 1$ and $S_5 = 0$. In this case, the effect of the timing error is that S_6 is clocked into the SR DFF as well as into the output DFF of the VN, rather than clocking the late S_7.

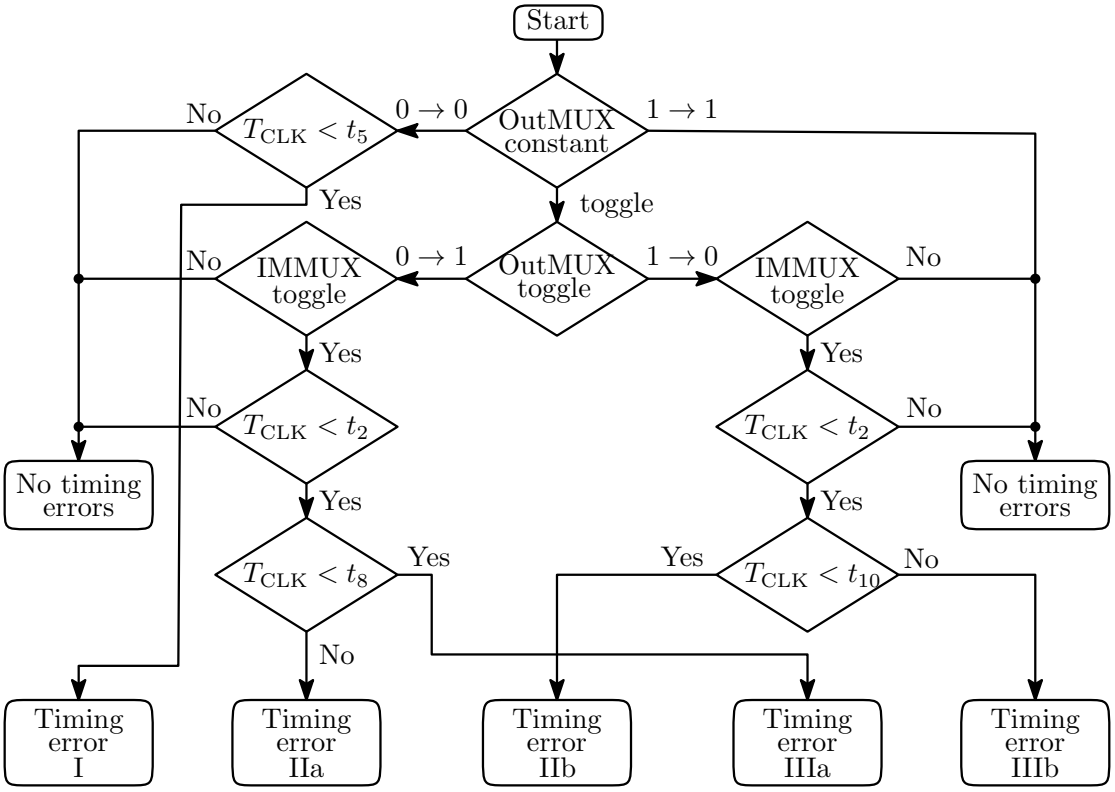


Figure 3.19: Flowchart illustrating causes and effects of timing errors in stochastic VNs having a degree of $d_i = 3$.

SPICE simulations were employed to obtain the waveforms of Figure 3.20, which present the error-free zero-delay response of a stochastic VN having the degree of $d_i = 3$, as well as the corresponding error-free real-delay response when the supply voltage is 1.0 V and the occurrence of timing errors when the supply voltage is 0.8 V.

For the case of $V_{DD} = 0.8$ V, a TET IIa occurs in clock cycles 2 and 3. In this case, $S_5 = 1$ fails to propagate in time and the signal that is clocked into the output DFF of the VN is S_7 instead of S_6. Similarly, a TET IIb is present in clock cycles 4 and 5 when $S_5 = 0$. In this situation, the ideal output value corresponds to $\hat{S}_8 = S_7$, however, the actual value that is clocked into the output DFF of the VN is $S_8 = S_6$.

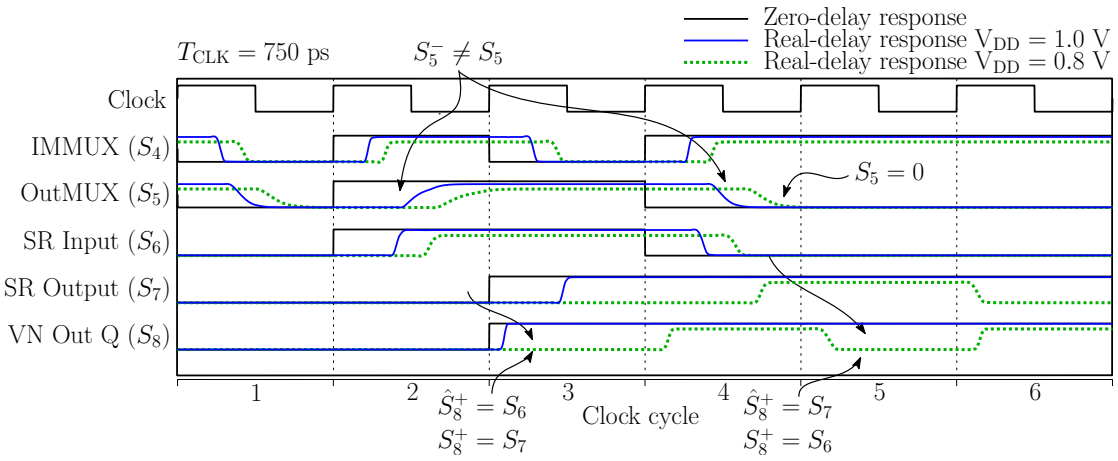


Figure 3.20: SPICE simulation demonstrating the occurrence of TET II in the stochastic VN having a degree $d_i = 3$.

A similar approach can be followed for VNs having degree $d_i = 2$ and $d_i = 6$. More specifically, Figures 3.21 and 3.22 can be employed for determining the occurrence and the effects of timing errors in VNs having degree $d_i = 2$ and $d_i = 6$, respectively.

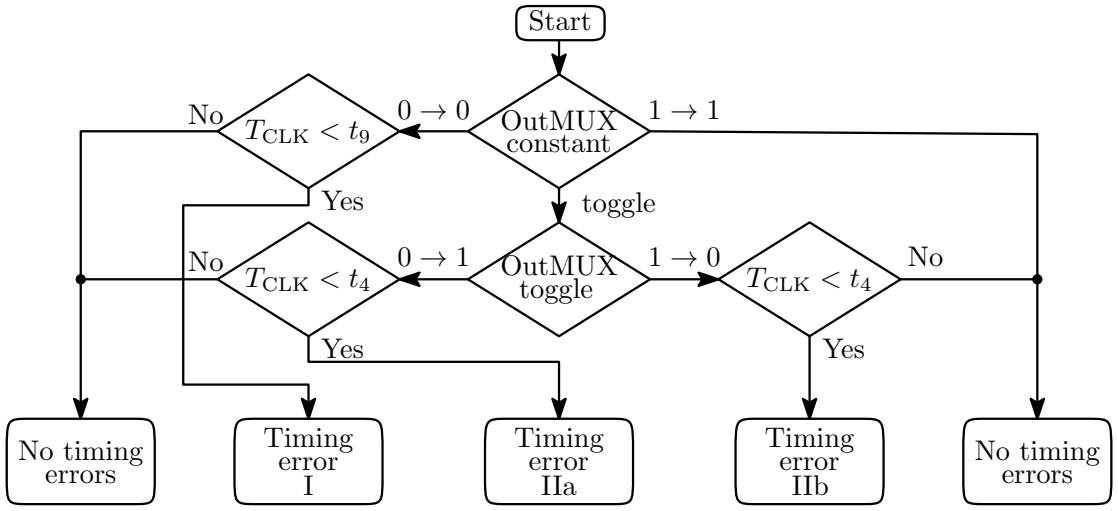


Figure 3.21: Flowchart illustrating causes and effects of timing errors in stochastic VNs having a degree of $d_i = 2$.

Note that since TET III occurs when both the OutMUX selector signal and the output signal of the SR arrive after the clock edge, this type of error is not considered in the analysis of degree $d_i = 2$ VNs of Figure 3.21, owing to this propagation delay not exceeding the maximum CN delay.

3.4.2 Decoding performance in the presence of timing errors

Monte Carlo simulations were performed to characterize the error correction capability of the SLDPCD in the presence of timing errors, when employing particular combinations

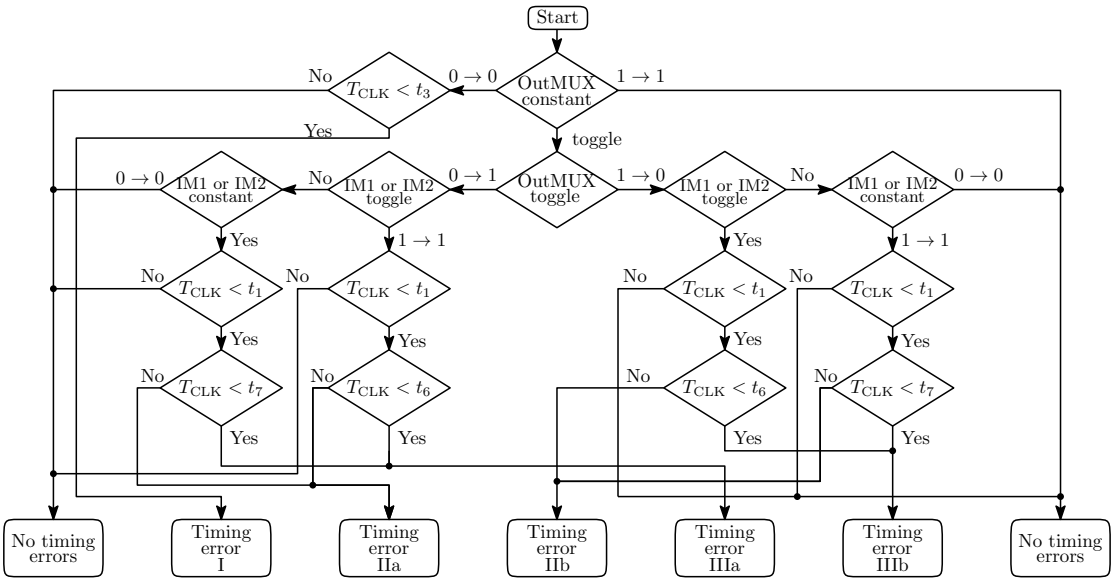


Figure 3.22: Flowchart illustrating causes and effects of timing errors in stochastic VNs having a degree of $d_i = 6$.

of $(\mu, \%, T_{\text{clk}})$, which corresponds to a particular value of the nominal supply voltage μ, three standard deviation percentage of supply voltage noise, which is obtained by expressing $3\sigma/\mu$ as a percentage, and clock period T_{clk}, respectively. In our analysis, the nominal supply voltage μ, the degree of power supply variations $3\sigma/\mu$ and the clock period T_{clk} are fixed for the whole operation of the decoder [77]. However, a different random value of V_{DD} is selected from the Gaussian distribution having a mean of μ and standard deviation of σ for each clock cycle. The selected value of V_{DD} is then used for all gates in the SLDPCD during the current clock cycle. The selected values of μ are 1.0 V and 0.8 V, corresponding to the nominal supply voltage of ST 90 nm technology and this value scaled down 20%, respectively. The value of σ is selected to obtain a particular value of $3\sigma/\mu$, namely 0.01, 0.10 and 0.30, which correspond to the three standard deviation variation fraction of the selected supply voltage of 1%, 10% and 30%, respectively, as detailed in [78]. In this configuration, the causes and effects of the timing errors are modeled using the technique exemplified in Figures 3.19, 3.21 and 3.22.

The combinations of $(\mu, \%, T_{\text{clk}})$ that we consider are listed in Table 3.5. We consider the cases where the SLDPCD operates at a clock period of $T_{\text{clk}} < t_4$ to be examples of aggressive overclocking. Similarly, we consider the cases where the SLDPCD operates at a clock period such that $t_2 < T_{\text{clk}} \leq t_4$ to be moderate overclocking. Finally, we consider $T_{\text{clk}} > t_1$ as relaxed overclocking, since the clock period is large enough to ensure that all circuit paths can correctly propagate their signals when $\sigma = 0$. The performed simulations correspond to a limit of 10^4 DCs of the (1056,528) IEEE 802.16e (WiMAX) LDPC decoder, assuming BPSK transmission over an AWGN channel. Figure 3.23 and Figure 3.24 plot the BER of the SLDPCD in the presence of timing errors, for the cases

Table 3.5: Combinations of (μ, %, T_{clk}) that are considered for BER simulations.

μ	%	T_{clk} (ps)	Overclocking
1.0	1%	600	Aggressive
		750	Relaxed
	10%	700	Moderate
		750	Relaxed
		800	Relaxed
	30%	550	Aggressive
		750	Relaxed
		950	Relaxed
0.8	1%	800	Aggressive
		1200	Relaxed
	10%	800	Aggressive
		1200	Relaxed
		1400	Relaxed
	30%	1000	Moderate
		1200	Relaxed
		1600	Relaxed

of nominal supply voltages of μ =1.0 V and 0.8 V, respectively. We present different BER plots for each chosen combination of $(\mu, \%, T_{\text{clk}})$ of Table 3.5. The BER is also plotted for two benchmark, namely the corresponding FP Log-SPA LDPC decoder and the SLDPCD in the absence of timing errors, as characterized in Section 3.3.

As shown in Figure 3.23, the BER of the SLDPCD in the absence of timing errors is very similar to the FP implementation. Figure 3.23 shows that relaxed overclocking corresponding to clock periods of 800 ps and 950 ps are required to guarantee the unimpaired BER performance of the SLDPCD when it operates with supply power variations of 10% and 30%, respectively. This represents 1.07 and 1.27 times, the critical clock period of 745 ps, respectively. The error correction capabilities of the SLDPCD are degraded by about 0.5 dB, when applying the aggressive overclocking of (1.0 V, 1%, 600 ps). By contrast, the same degradation is offered by the relaxed overclocking of (1.0 V, 10%, 750 ps) and (1.0 V, 30%, 750 ps), owing to the increased percentage of power supply variations. Finally, a degradation of about 0.8 dB is imposed, when the SLDPCD operates with the moderate overclocking of (1.0 V, 10%, 700 ps) and the aggressive overclocking of (1.0 V, 30%, 550 ps).

In analogy to the case when the SLDPCD operates at 1.0 V, Figure 3.24 shows that, when operated at 0.8 V and with power supply variations of 10% and 30%, the clock period needs to be set to 1400 ps and 1600 ps, respectively, in order to guarantee the unimpaired BER performance of the SLDPCD. This represents 1.21 and 1.38 times, respectively, the critical clock period of the SLDPCD of 1160 ps, when operated at 0.8 V. The error correction capabilities of the SLDPCD are degraded by about 0.5 dB, when applying the aggressive overclocking of (0.8 V, 1%, 800 ps). By contrast, the

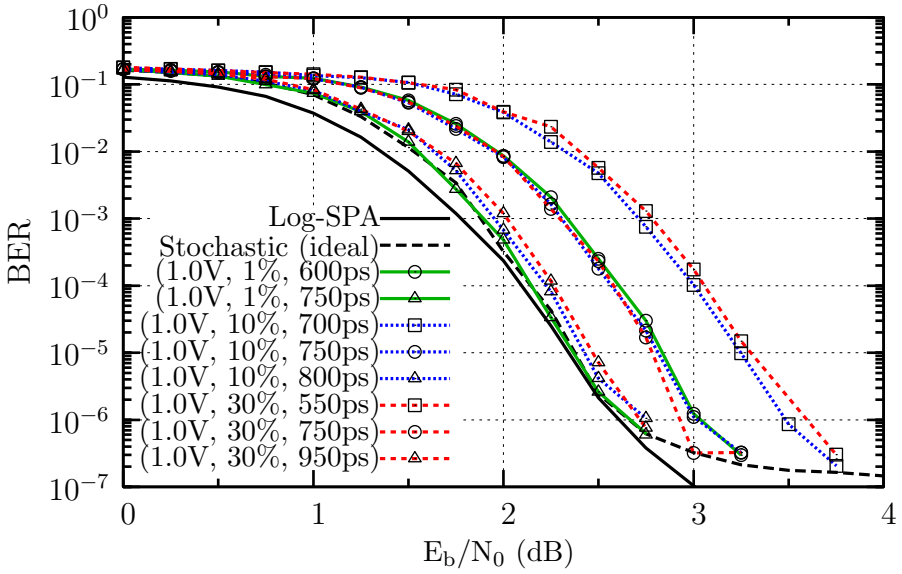


Figure 3.23: BER of the SLDPCD with V_{DD} = 1.0 V

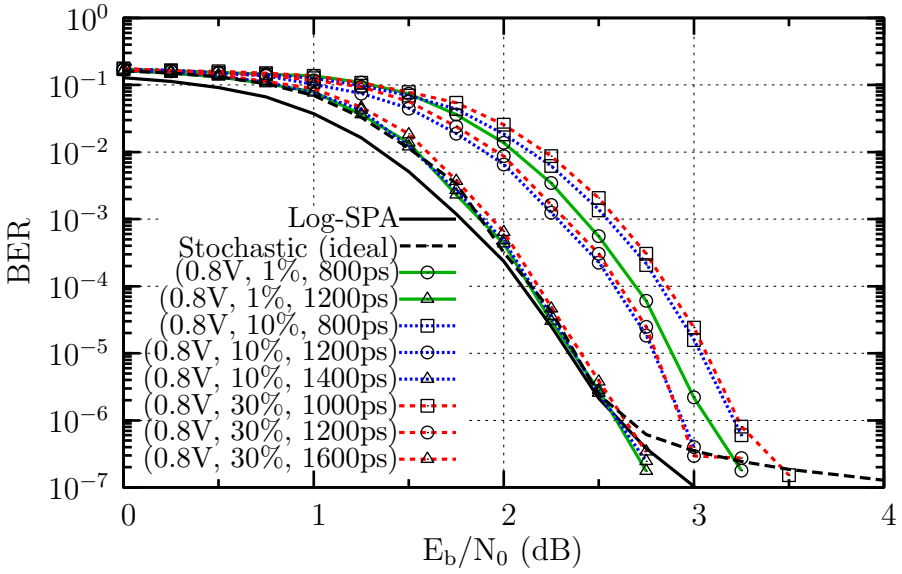


Figure 3.24: BER of the SLDPCD with V_{DD} = 0.8 V

same degradation is offered by the relaxed overclocking of (0.8 V, 10%, 1200 ps) and (0.8 V, 30%, 1200 ps), owing to the increased percentage of power supply variations. Finally, a degradation of about 0.7 dB is imposed, when the SLDPCD operates with the aggressive overclocking of (0.8 V, 10%, 800 ps) and the moderate overclocking of (0.8 V, 30%, 1000 ps).

Based on these observations, we consider that the SLDPCD offers an inherent tolerance to timing errors. However, this tolerance is significantly degraded when the SLDPCD operates with percentage of power supply variations of 10% and 30% and when moderate or aggressive overclocking is employed. Owing to this, Section 3.5 presents a modified

EM structure designed for improving the inherent tolerance to timing errors in the SLDPCD.

3.5 Modified Stochastic LDPC Decoder

As mentioned in Section 3.4, the SLDPCD exhibits a near optimal decoding performance in the presence of timing errors when relaxed overclocking is employed. However, this inherent tolerance to timing errors is degraded in the cases when moderate and aggressive overclocking is employed. Moreover, according to the analysis presented in Figures 3.19, 3.21 and 3.22, TET II and TET III occur when the OutMUX selector signal changes its value in consecutive clock cycles. This may be attributed to the late arrival of the OutMUX signal, owing to its large capacitive load. More specifically, the OutMUX selector signals is employed by the 32, 48 and 64 DFFs of the SRs in VNs of degree $d_i = 2$, 3 and 6, respectively. Motivated by this, the following sections present modifications to the EM structure of the stochastic VNs that reduce the propagation delay of the OutMUX selector signal, reduce the likelihood of timing error occurrences and grants the SLDPCD a significantly increased tolerance to TET II and TET III.

3.5.1 Modified EM

In the proposed modified EM, the propagation delay of the OutMUX selector signal is reduced by reducing its capacitive load, as shown in Figure 3.25. Note that in the modified stochastic VN, the OutMUX signal drives only one MUX gate within the EM of Figure 3.25(b), rather than the 32, 48 and 64 MUXs of the EM of Figure 3.25(a) of the stochastic VNs having degrees of $d_i = 2, 3$ and 6, respectively. Moreover, the proposed scheme changes the functionality of the EM from behaving as a SR to a Ring Buffer (RB). In this configuration, when the update signal S_5 of the RB adopts the value of 0, the output Q of the m-th DFF is provided as the input D of the first DFF of the RB. Owing to this, a new input is not guaranteed to replace the oldest value in the RB. Despite this, the error correction capability of the RB-based SLDPCD is not degraded, as will be shown in Section 3.5.4. The highlighted paths in Figure 3.25 illustrate the difference in the driving load between the SR-based and the RB-based EMs, shown in Figure 3.25(a) and Figure 3.25(b), respectively.

3.5.2 Hardware implementation

In analogy to the hardware implementation results of the SR-based SLDPCD presented in Section 3.3, Table 3.6 presents the chip area, minimum clock period and energy consumption of the RB-based SLDPCD.

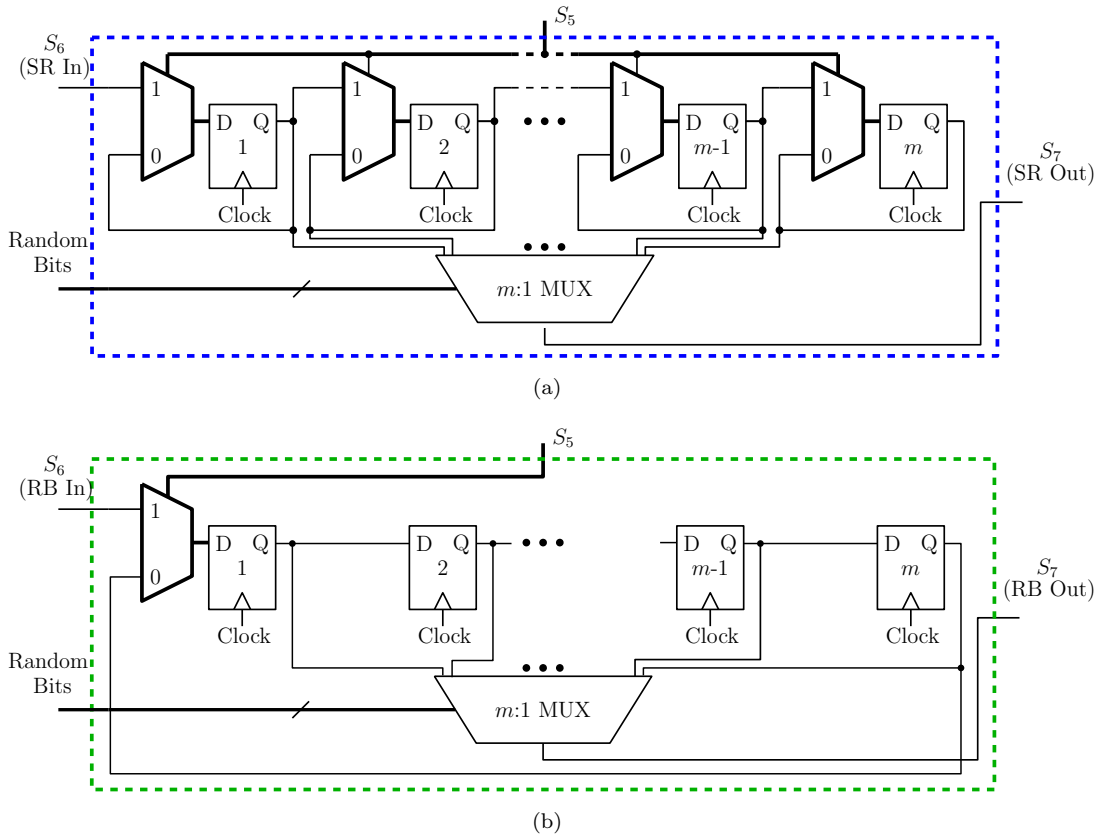


Figure 3.25: Comparison of driving loads in SR-based and RB-based EMs: (a) SR-based. (b) RB-based.

Table 3.6 shows that the chip area of the RB-based SLDPCD is reduced to about 0.77 times the chip area of the SR-based SLDPCD. This may be attributed to the employment of only one MUX for updating the DFFs of the RBs, compared to the 32, 48 and 64 employed in SRs for the cases of VNs of degree $d_i = 2$, 3 and 6, respectively. Moreover, the employment of RBs facilitates lower critical clock periods. This may be attributed to the reduced capacitive load of the OutMUX selector signal S_5. As an example of this, the maximum propagation delay of the RB-based SLDPCD, when operated at 1.0 V is 642 ps, which represents 0.84 times the clock period of the SR-based SLDPCD of 745 ps. Similarly, the maximum propagation delay of the RB-based SLDPCD, when operated at 0.8 V is 1033 ps, which represents 0.89 times the clock period of 1160 ps of the SR-based SLDPCD. However, Table 3.6 shows that the energy consumption per DC in the RB-based SLDPCD is slightly increased, when compared to that of the SR-based SLDPCD. More specifically, the energy consumption per DC of the RB-based SLDPCD, when operated at 1.0 V and 0.8 V is 4009 pJ and 2484 pJ, respectively. This represents the energy consumption increased by factors of 1.045 and 1.036, when compared to the energy consumption of 3833 pJ and 2396 pJ of the SR-based SLDPCD, respectively. This may be attributed to the increased switching of the DFFs of the RB-based EMs, owing to the continually-updated operation of the RB, rather than the occasionally-updated operation of the SR. To elaborate further, in the SR shown in the blue box of

Table 3.6: Hardware implementation performance of the RB-based SLDPCD.

Node		Area	V_{DD}	T_{clk}	Energy per DC	
					per node	all nodes
VN	$d_i = 2$ (484×)	983 μm^2 (224 gates)	1.0 V	426 ps	1.55 pJ	750 pJ
			0.8 V	799 ps	0.98 pJ	474 pJ
	$d_i = 3$ (352×)	1444 μm^2 (329 gates)	1.0 V	599 ps	3.29 pJ	1159 pJ
			0.8 V	947 ps	2.03 pJ	716 pJ
	$d_i = 6$ (220×)	1916 μm^2 (437 gates)	1.0 V	**642 ps**	8.03 pJ	1937 pJ
			0.8 V	**1033 ps**	5.42 pJ	1192 pJ
CN	$d_j = 6$ (352×)	222 μm^2 (51 gates)	1.0 V	318 ps	0.30 pJ	105 pJ
			0.8 V	505 ps	0.19 pJ	67 pJ
	$d_i = 7$ (176×)	257 μm^2 (59 gates)	1.0 V	370 ps	0.33 pJ	58 pJ
			0.8 V	560 ps	0.20 pJ	35 pJ
TOTAL		1.53 mm^2 (348×10^3 gates)	1.0 V	642 ps	N/A	4009 pJ
			0.8 V	1033 ps	N/A	2484 pJ

Figure 3.25, the DFFs will shift their contents only if the update signal S_5 is asserted and will maintain their value otherwise. By contrast, the DFFs of the RB shown in the green box of Figure 3.25 will shift their contents by one position in every DC, regardless of the value of the update signal S_5.

In addition to the results presented in Table 3.6, Figure 3.26 presents the BER decoding performance of the RB-based (1056,528) IEEE 802.16e (WiMAX) SLDPCD, assuming BPSK transmission over an AWGN channel, when allowing a maximum of 10^4 DCs and when NDS with $\eta = 1$ and $\psi = 2$ is employed, as described in Section 2.3.2. Figure 3.26 also plots the BER of the FP Log-SPA LDPC decoder and of the SR-based SLDPCD as benchmarks. As shown in Figure 3.26, the BER of the RB-based SLDPCD is very similar to that of the FP and SR-based implementations. In Section 3.5.4, we will consider the effects that timing errors impose in the BER performance of the SLDPCD.

C++ simulations of the RB-based SLDPCD were used to measure the average number of DCs required for successfully decoding a frame, when employing early stopping and allowing a maximum of 10^4 DCs, as shown in Figure 3.27.

Figure 3.27 shows that RB-based SLDPCDs exhibit a slightly increased average number of DCs for decoding a frame, when compared to the SR-based counterpart. This may be attributed to the operation of the RB, in which a new regenerative bit in the RB is not guaranteed to replace older inputs. However, Figure 3.28 shows that the latency of RB-based SLDPCDs is actually slightly reduced, owing to the reduced T_{clk} values of these schemes. As a result of this, RB-based SLDPCDs exhibit an slightly increased throughput, albeit at the cost of an increased energy consumption, as shown in Figures 3.29 and 3.30, respectively.

Note that the SLDPCDs of [33, 34] report throughputs of 61 and 172 Gbps, which are one order of magnitude superior to the throughput of the RB-based and SR-based SLDPCDs.

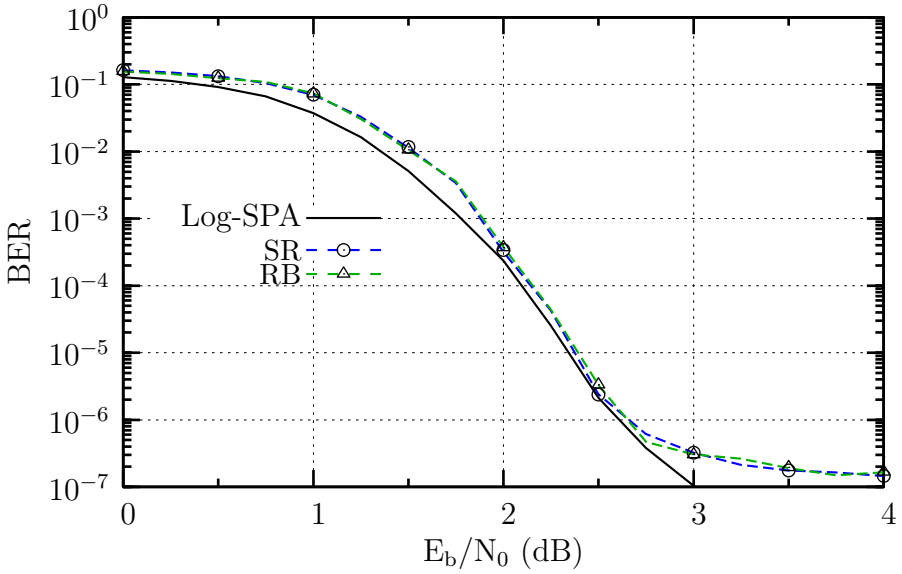


Figure 3.26: BER of the RB-based (1056,528) SLDPCD.

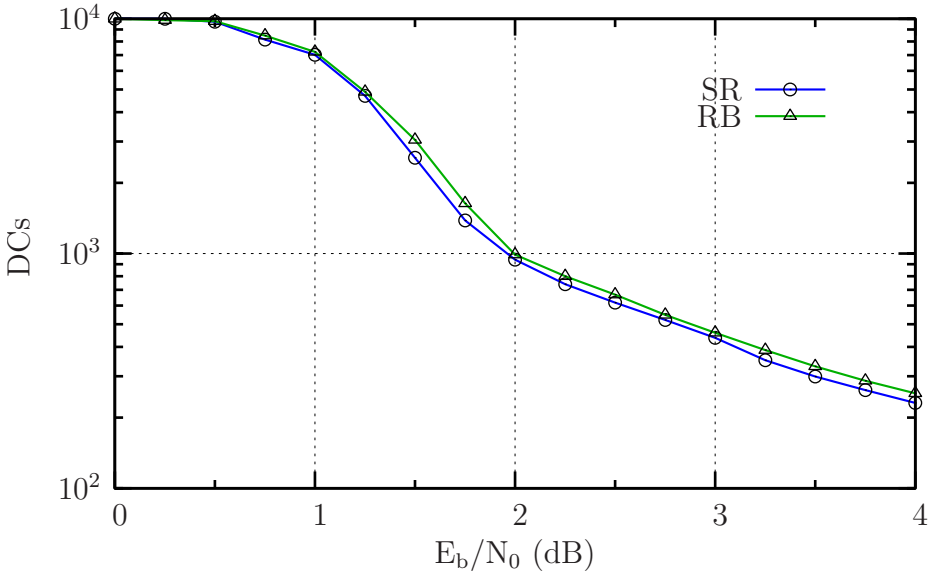


Figure 3.27: Average number of DCs for successfully decoding a frame in the RB-based SLDPCD.

However, the authors of [33, 34] achieve this high throughput for a (2048,1723) SLDPCD operated at Signal to Noise Ratio (SNR) values in excess of $E_b/N_0 = 5.5$ dB, in order to achieve target BERs below 10^{-12}. Moreover, these high throughput implementations do not rely on SR-based or RB-based EMs. Instead, the stochastic VN structures have been carefully designed and optimized for the particular case of the (2048,1723) LDPC code. Owing to this, it is not possible to directly compare the RB-based and SR-based SLDPCDs with these high throughput implementations.

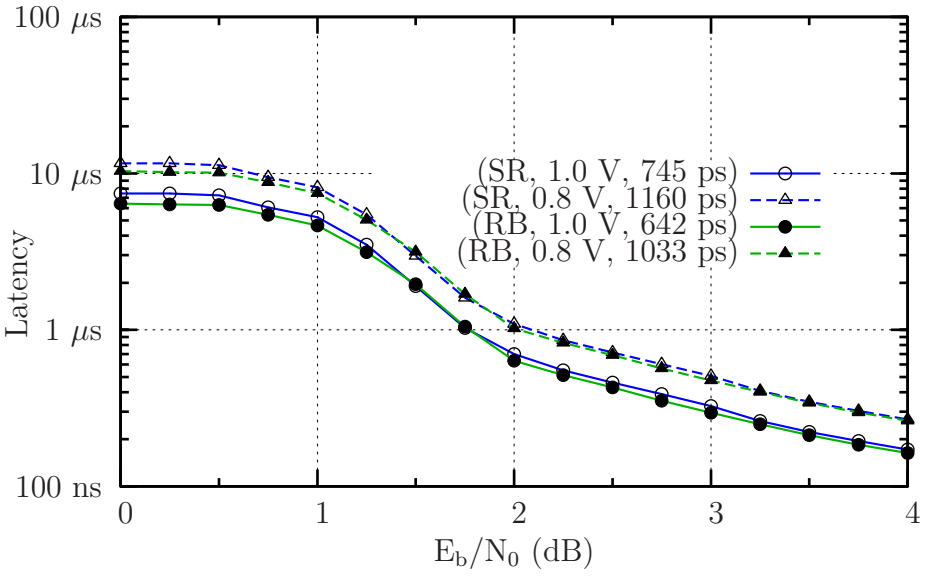


Figure 3.28: Latency of the RB-based SLDPCD.

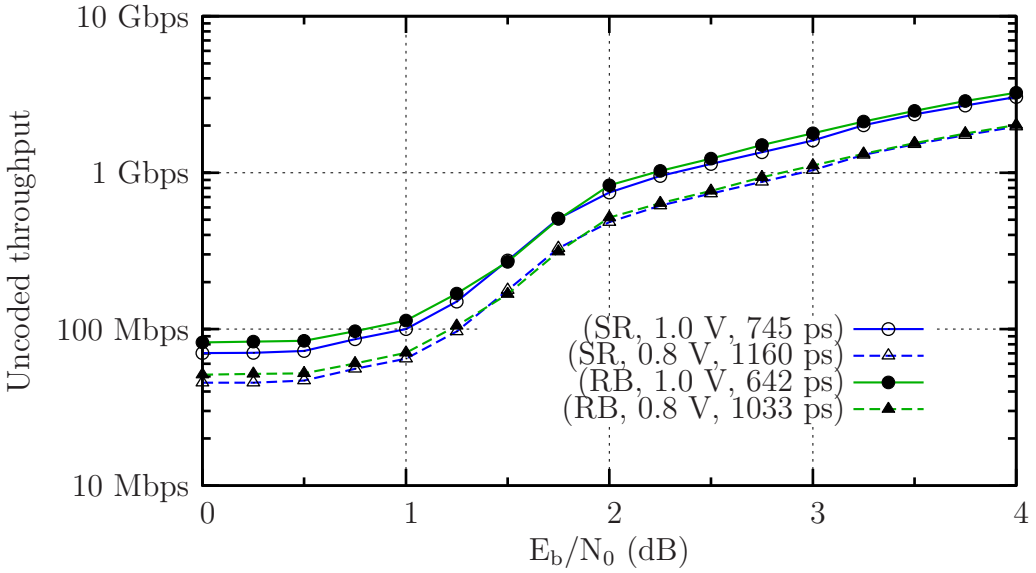


Figure 3.29: Uncoded throughput of the RB-based SLDPCD.

3.5.3 Error correction capabilities in the presence of timing errors

In analogy to the analysis presented in Section 3.4.1, this section characterizes the error correction capabilities of the RB-based SLDPCD in the presence of timing errors, when it operates with different voltages and clock periods. Moreover, this section presents a trade-off analysis of the hardware implementation of both SR- and RB-based SLDPCD in the presence of timing errors. More specifically, we provide a comparison of the average number of DCs, latency, throughput and energy consumption of the SLDPCDs when operated at different clock periods and different degrees of power supply variation.

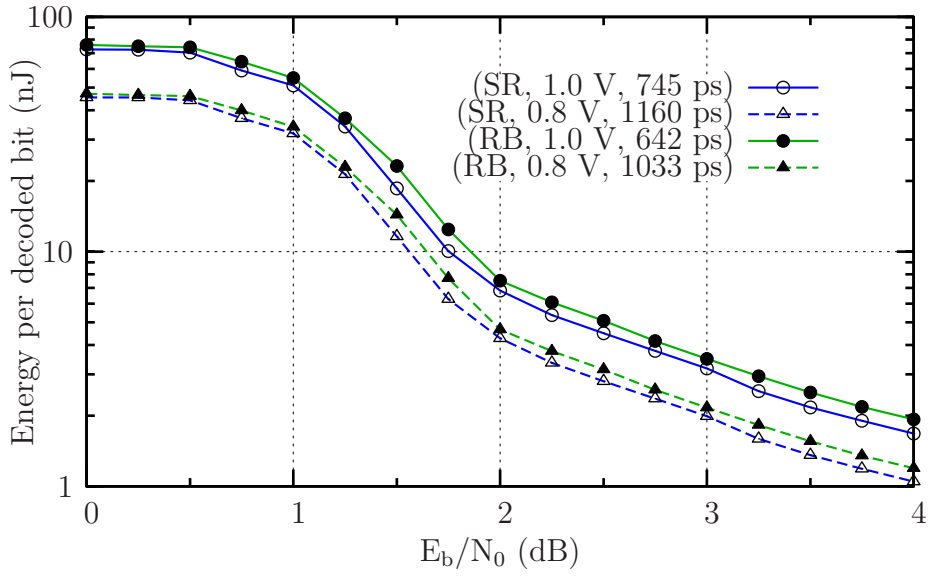


Figure 3.30: Energy consumption per decoded bit of the RB-based SLDPCD.

3.5.3.1 Timing error analysis

The analysis of Section 3.4.1 can also be applied to characterize the occurrence of timing errors within the modified SLDPCD. Figure 3.31 demonstrates that the RB-based SLDPCD offers reduced propagation delays t_p^{RB}, when compared to the propagation delays t_p^{SR} of the SR-based SLDPCD. This may be attributed to the reduced capacitive load of the OutMUX selector signal S_5.

Note that identical propagation delays t_3, t_5 and t_9 were found for both the SR- and the RB-based SLDPCD, since these propagation delays are dominated by the 64:1, 48:1 and 32:1 MUX that is common to both SR- and RB-based EMs in stochastic VNs having the degree of $d_i = 6$, 3 and 2, respectively. Propagation delays t_7^{RB}, t_8^{RB} and t_{10}^{RB} are not considered in Figure 3.31, since these delays have been reduced below the maximum propagation delay of the CNs t_{CN}. These reduced propagation delays improve the BER performance of the RB-based SLDPCD in the presence of timing errors, as we will demonstrate in Section 3.5.4.

The analysis presented in Section 3.4.1.2 can also be employed for the SLDPCD based on the modified EM to determine the causes and effects of timing errors in the RB-based stochastic VN. Figures 3.32 and 3.33 show the causes and effects of timing errors in RB-based VNs having degree of $d_i = 6$, and 3, respectively, with the analysis of VNs degree $d_i = 2$ of Figure 3.21 presenting no changes for the case of RB-based VNs. The combinations of the propagation delays of IMMUX and OutMUX selector signals that trigger TET IIa and TET IIIb in RB-based VNs having degree of $d_i = 3$ have been reduced below the maximum propagation delay t_{CN} of the CNs. Owing to this, these timing errors are eliminated from Figure 3.33. Furthermore, Figure 3.31 demonstrates that lower clock periods are required for the occurrence of timing errors in RB-based

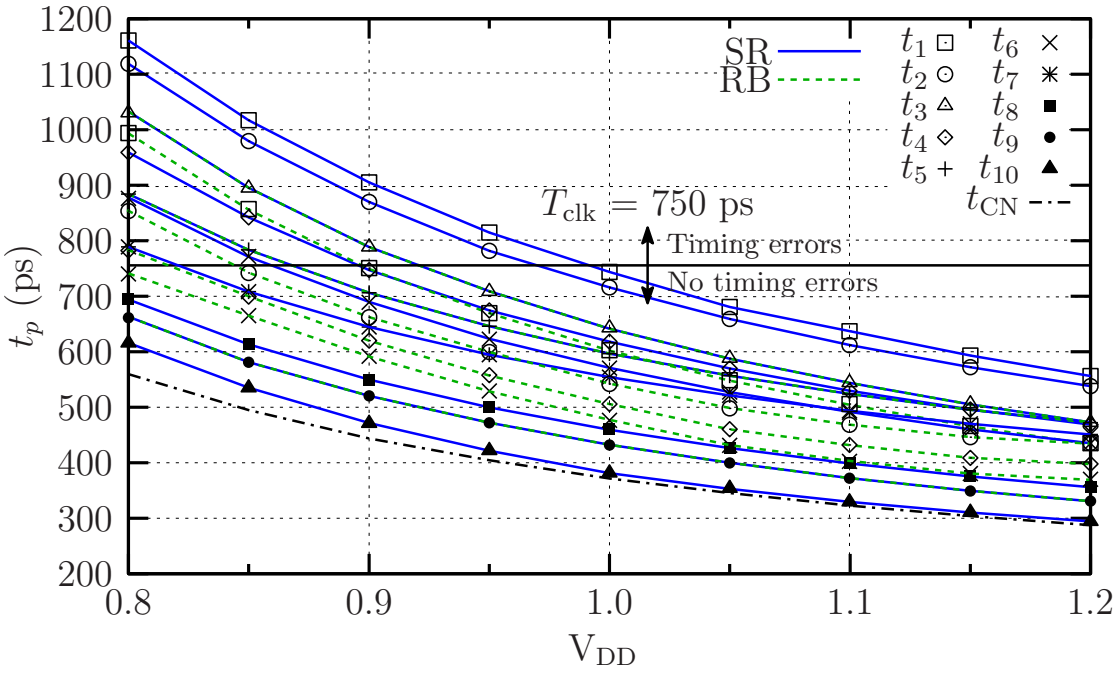


Figure 3.31: Comparison of propagation delays of the SR-based and RB-based SLDPCDs as a function of the supply voltage. Here, t_p^{SR} and t_p^{RB} denote the propagation delay of the path p in the SR-based and RB-based stochastic VNs, respectively.

VNs. As a result, timing errors can be expected to occur significantly less often in the RB-based VNs, compared to the SR-based VNs, when operated at the same clock period.

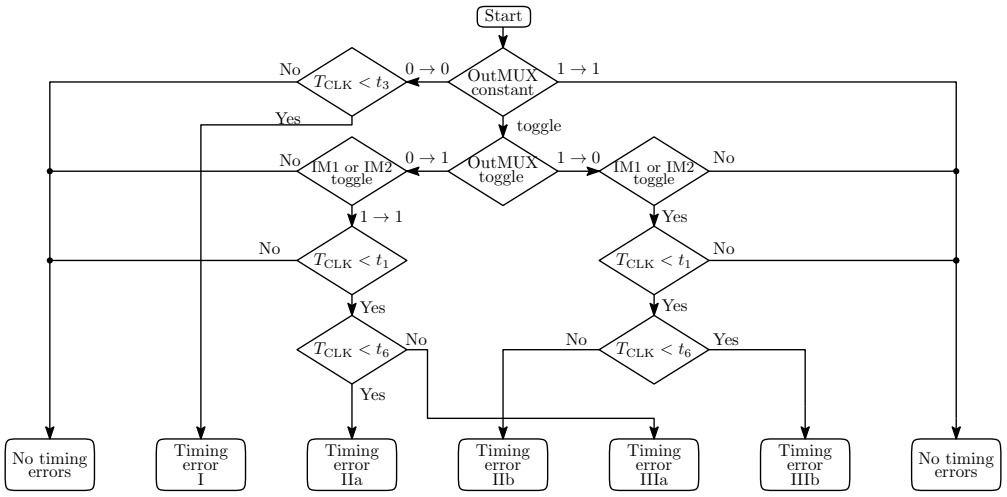


Figure 3.32: Flowchart illustrating causes and effects of timing errors in RB-based stochastic VNs having a degree of $d_i = 6$.

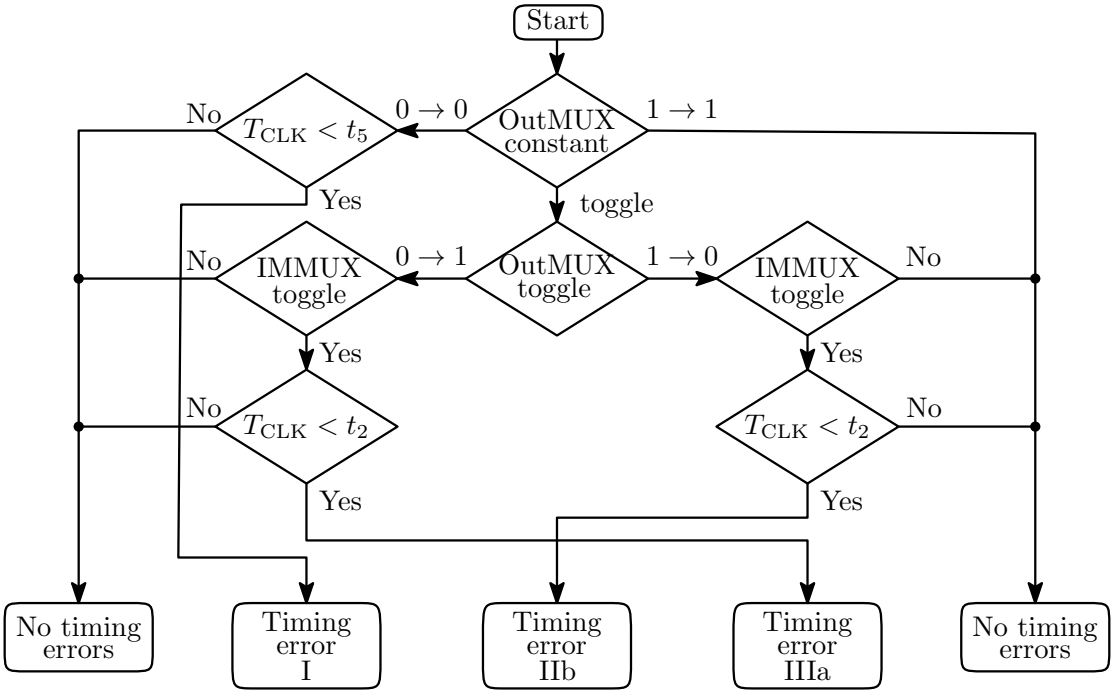


Figure 3.33: Flowchart illustrating causes and effects of timing errors in RB-based stochastic VNs having a degree of $d_i = 3$.

3.5.4 Decoding performance in the presence of timing errors

The Monte Carlo simulation of Section 3.4.1 can be employed to characterize the error correction capability of the modified RB-based SLDPCD. Figures 3.34 and 3.35 show the BER of the RB-based SLDPCD in the presence of timing errors, for the cases where the nominal supply voltage is μ=1.0 V and 0.8 V, respectively. Here, we employ the same combinations of $(\mu, \%, T_{\text{clk}})$ of Table 3.5 employed in the SR-based SLDPCD. This demonstrates that the RB-based SLDPCD offers an improved BER performance, when operated in the same conditions as the SR-based SLDPCD. As an example of this, Figure 3.34 shows that relaxed and moderate overclocking of Table 3.5 do not impose a significant degradation to the BER performance in the RB-based SLDPCD. Moreover, when the aggressive overclocking of Table 3.5 is employed in the RB-based SLDPCD, the E_b/N_0 degradation is reduced by 0.5 dB. More specifically, (1.0 V, 1%, 600 ps) presents an unimpaired BER performance, compared to the 0.5 dB degradation of the SR-based SLDPCD. Similarly, (1.0 V, 30%, 550 ps) offers an E_b/N_0 degradation of 0.3 dB, compared to 0.8 dB offered by the SR-based SLDPCD.

Figure 3.35 shows that the E_b/N_0 degradation of 0.7 dB offered by the SR-based SLDPCD in (0.8 V, 10%, 800 ps) and (0.8 V, 30%, 1000 ps) is reduced to 0.2 dB in the RB-based SLDPCD. Similarly, the E_b/N_0 degradation of 0.5 dB in the SR-based SLDPCD in (0.8 V, 1%, 800 ps) is reduced to 0.3 dB for the case of the RB-based SLDPCD.

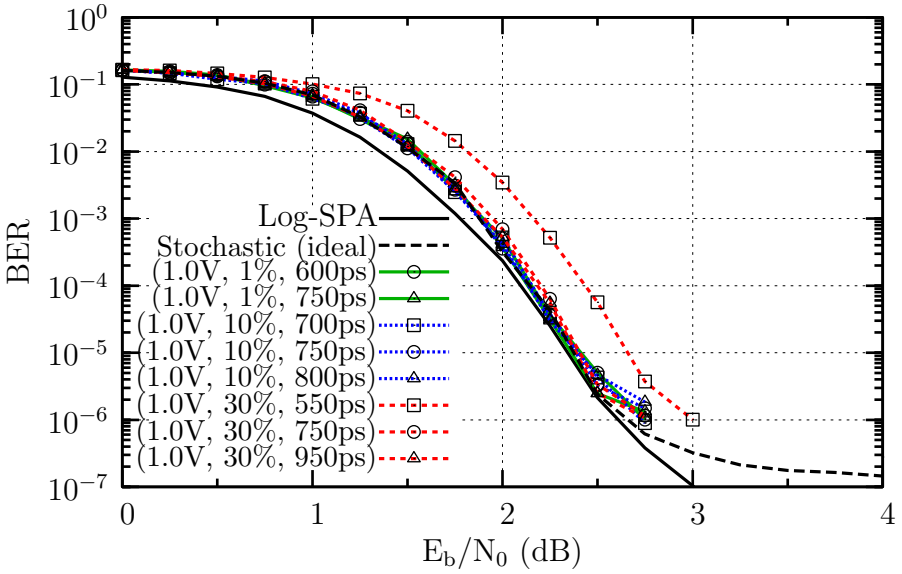


Figure 3.34: BER of the RB-based SLDPCD with V_{DD} = 1.0 V

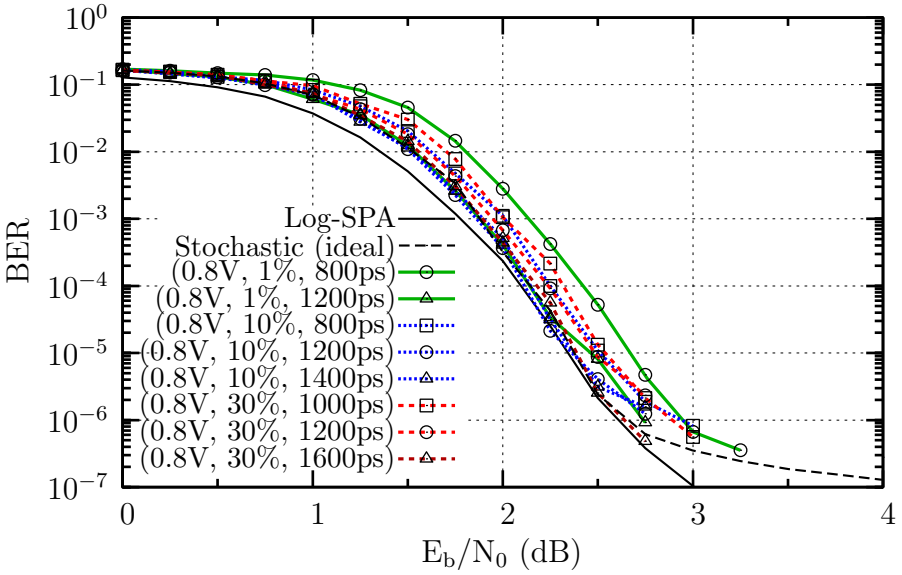


Figure 3.35: BER of the RB-based SLDPCD with V_{DD} = 0.8 V

In addition to the BER performance of Figures 3.34 and 3.35, Figures 3.36 and 3.37 characterize the error correction capabilities of the RB-based SLDPCD when employing different clock periods to those presented in Table 3.5.

Figure 3.36 shows that, when the critical clock period of the RB-based SLDPCD is reduced by a factor of 0.77 times in (1.0 V, 1%, 500 ps), the decoder exhibits an E_b/N_0 degradation of 0.5 dB. Similarly, an E_b/N_0 degradation of 0.4 dB is encountered if the selected clock period is set to 0.86 times the critical clock period in (1.0 V, 10%, 550 ps) and (1.0 V, 30%, 550 ps). The E_b/N_0 degradation is increased to 1.0 dB and 0.9 dB, when the RB-based SLDPCD operates with a clock period that is 0.63 and 0.70 times

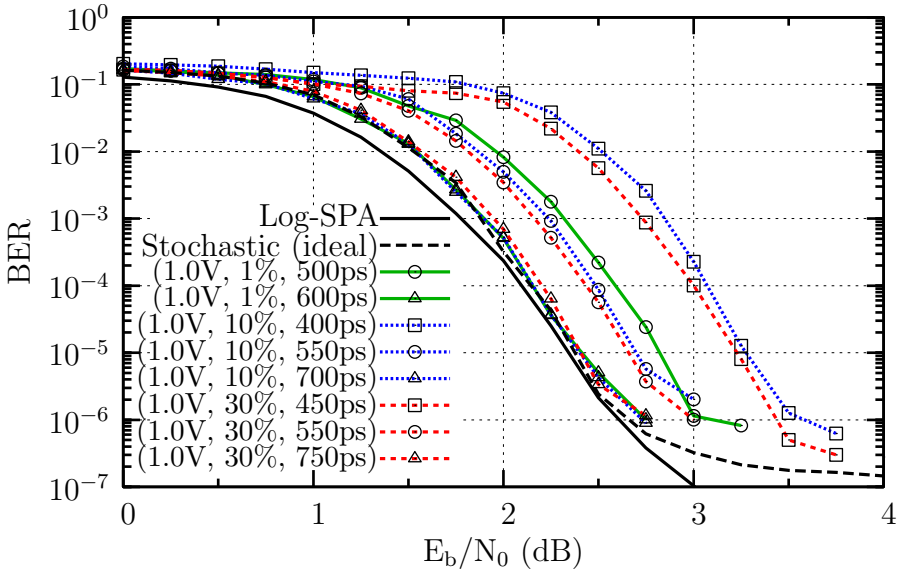


Figure 3.36: BER of the RB-based SLDPCD with V_{DD} = 1.0 V and aggressive overclocking

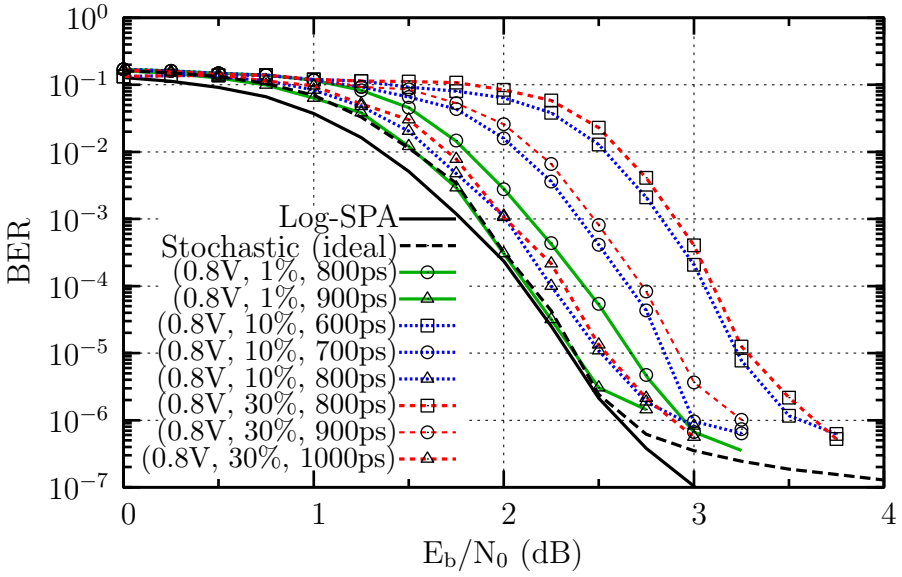


Figure 3.37: BER of the RB-based SLDPCD with V_{DD} = 0.8 V and aggressive overclocking

the critical clock, respectively. This is observed in schemes (1.0 V, 10%, 400 ps) and (1.0 V, 30%, 450 ps), respectively.

Similarly, when the critical clock period is reduced by factors of 0.58 and 0.78 times in (0.8 V, 10%, 600 ps) and (0.8 V, 30%, 800 ps), respectively, the RB-based SLDPCD exhibits an E_b/N_0 degradation of about 1.0 dB, These results demonstrate that the RB-based SLDPCD offer an increased tolerance to timing errors, when compared to the SR-based SLDPCD.

In addition to the BER performance, This trade-off analysis is presented normalized

relative to the SR-based SLDPCD operated at its critical path, in the absence of timing errors and when $E_b/N_0 = 3.00$ dB. The selected combinations of $(\mu, \%, T_{clk})$ of Figure 3.38 correspond to the cases when the SLDPCDs present a near-optimal decoding capabilities in the presence of timing errors, according to the BER results of Figures 3.23, 3.24 and 3.34 to 3.37.

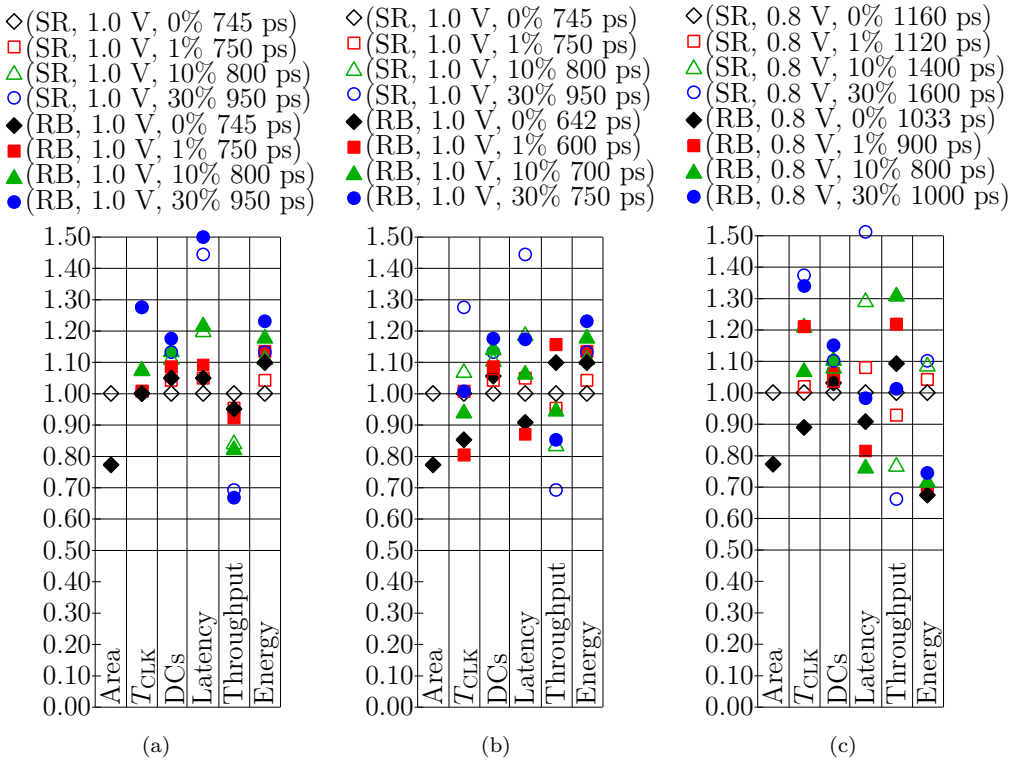


Figure 3.38: Hardware implementation results of the RB-based SLDPCD in the presence of timing errors. (a)$V_{DD} = 1.0$ V, same T_{clk} values for both the SR-and RB-based SLDPCD. (b)$V_{DD} = 1.0$ V, different T_{clk} values for the SR-and RB-based SLDPCD. (c)$V_{DD} = 0.8$ V, different T_{clk} values for the SR-and RB-based SLDPCD.

Figure 3.38(a) shows that when both RB- and SR-based SLDPCDs operate at 1.0 V, the same clock periods and the same degree of power supply variations, RB-based SLDPCDs suffer from increased latencies, reduced throughputs and increased energy consumptions. This may be attributed to the increased number of DCs required by RB-based decoders and to the values of clock period employed, which are larger than the critical clock period of the RB-based SLDPCD. However, Figure 3.38(b) shows that reduced T_{clk} values in RB-based SLDPCDs facilitate reduced latencies and increased throughputs, when compared to the SR-based SLDPCDs, albeit at the cost of increased energy consumptions. As an example of this, when moderate overclocking is applied in the RB-based SLDPCD, the scheme (RB, 1.0 V, 1%, 600 ps) exhibits a latency of 0.87 times that of (SR, 1.0 V, 0%, 745 ps), despite the occurrence of timing errors. By contrast, a relaxed overclocking in the SR-based SLDPCD in (SR, 1.0 V, 1%, 750 ps) exhibits a latency

increased by a factor of 1.04, when compared to the ideal SR-based decoder. Similarly, the throughput of (RB, 1.0 V, 1%, 600 ps) is 1.16 times the throughput of (SR, 1.0 V, 0%, 745 ps), compared to the reduced throughput by factor of 0.96 in (SR, 1.0 V, 1%, 750 ps). However, the energy consumption of (RB, 1.0 V, 1%, 600 ps) is 1.14 times that of (SR, 1.0 V, 0%, 745 ps), compared to the energy consumption increased by a factor of 1.04 in (SR, 1.0 V, 1%, 750 ps).

Figure 3.38(c) shows that when SLDPCDs operate at 0.8 V, RB-based decoders offer reduced latencies, increased throughputs and reduced energy consumptions, when compared to SR-based decoders. As an example of this, when the relaxed overclocking of (SR, 0.8 V, 10%, 1400 ps) is employed, the latency, throughput and energy consumption are 1.29, 0.78 and 1.09 times those of (SR, 0.8 V, 0%, 1160 ps), respectively. By contrast, the latency, throughput an energy consumption of the scheme (RB, 0.8 V, 10%, 800 ps) are 0.82, 1.22 and 0.70 times those of (SR, 0.8 V, 0%, 1160 ps), respectively, despite the occurrence of timing errors and aggressive overclocking.

3.6 Chapter Conclusions

In this chapter, we have analyzed the inherent error tolerance of SLDPCDs to timing errors owing to power supply variations and overclocking for the first time. This has been achieved by performing a timing analysis of the SLDPCDs to determine the causes and effects of timing errors. We have applied this timing error model into our BER simulations to determine the error correction capabilities of SR- and RB-based SLDPCDs, when operated at different nominal supply voltages, different clock periods and different degrees of power supply variations. In addition to this, we have presented a trade-off analysis of the chip area, critical clock period, number of DCs, latency, throughput and energy consumption of the SLDPCDs in the presence of timing errors. This analysis demonstrate that the chip area requirements of the proposed RB-based SLDPCDs is only 0.77 times that of SR-based SLDPCDs. When operated at 1.0 V and in the presence of timing errors, the RB-based SLDPCD offer reduced latencies and increased throughputs, when compared to the SR-based SLDPCD, albeit at the cost of increased energy consumptions. However, RB-based SLDPCDs operated at 0.8 V offer reduced latencies, increased throughputs and reduced energy consumptions, when compared to the SR-based SLDPCDs operated at the same nominal supply voltage.

The timing analysis presented in Section 3.4.1 can be applied to determine the causes and effects of timing errors in different designs. However, the complexity of this analysis may be increased depending on the structure and behavior of each design. As an example of this, a significant complexity increase is observed in the flowcharts of Figures 3.19, 3.21 and 3.22 for determining the causes and effects of timing errors in VNs having the degree of $d_i = 3$, 2 and 6, respectively. This complexity may be further

increased if the analysis is extended to VNs having higher degrees or if the occurrence of metastability owing to timing errors is considered, for example. As a result, this chapter has not explored timing-error-tolerant design techniques for preventing the catastrophic propagation of metastability in SLDPCDs. In contrast to this, Chapter 4 explores error-tolerant design techniques in order to prevent the propagation of metastability through the circuit, with Stochastic Turbo Decoders (STDs) as a particular example. In addition to this, the result presented in this chapter demonstrate that SLDPCDs achieve processing throughputs on the order of Gbps. As a result of this, SLDPCD have the potential for being considered in ultra-high-throughput and ultra-low-latency next-generation communication standards [10, 11]. This is achieved at a low implementation complexity, while offering an inherent tolerance to processing errors. Motivated by this, Chapter 4 explores the employment of stochastic computing in turbo codes. More specifically, Chapter 4 characterizes the tolerance to timing errors of the STD presented in [39, 79]. This is achieved by employing a timing analysis of the causes and effects of timing errors in STDs. Additionally, Chapter 4 presents modifications to the STD that improve its tolerance to timing errors in the presence of power supply variations, whilst considering a trade-off analysis of the hardware implementation of STDs, in analogy to the results presented in this chapter for SLDPCDs.

Chapter 4

Timing-Error-Tolerant Stochastic Turbo Decoders

The Stochastic LDPC Decoders (SLDPCDs) presented in Chapter 3 offer an attractive trade-off between Bit Error Ratio (BER) performance, throughput, energy efficiency and chip area. This may be attributed to the employment of stochastic computing [18], where operations such as additions, multiplications and divisions are performed using low-complexity digital circuits, as detailed in Chapter 2. Moreover, the SLDPCDs presented in Chapter 3 exhibit an inherent tolerance to timing errors imposed by the reduction of the clock period. Furthermore, timing errors are more likely to occur when the supply voltage is reduced and the clock period is not adjusted accordingly, owing to the quadratic dependency of propagation delays on the supply voltage. Whenever a timing error occurs, there is a chance that the affected D-type Flip Flop (DFF) might enter into a metastable state, in which the digital signals have an indeterminate value that does not correspond to either a logic 0 or 1. However, given sufficient time, the metastable state will randomly evolve to a stable but unpredictable logic value of 0 or 1 [80]. This effectively imposes an additional propagation delay on the affected signal, hence increasing the likelihood of timing errors and metastability occurring at the next DFF. In this way, a single metastable event can trigger subsequent metastability occurrences in successive DFFs, causing the catastrophic propagation of metastability that destroys the operation of the entire circuit.

Against this background, we propose enhancements of the Stochastic Turbo Decoder (STD) of [39] for improving its tolerance to timing errors in the presence of power supply variations. More specifically, the enhanced STD design proposes:

This chapter is partially based on the following publications.

I. Perez-Andrade, S. Zhong, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Stochastic computing improves the timing-error tolerance and latency of turbo decoders: Design guidelines and trade-offs," in *IEEE Access,* vol. 4, pp. 1008–1038, Feb 2016.

1) The employment of synchronizers for preventing the catastrophic cascading of metastability owing to timing errors.

2) The simultaneous decoding of two received frames for improving the processing throughput.

3) The employment of Tracking Forecast Memories (TFMs) [30] in STDs for the first time, in order to enhance their hardware implementation and decoding capabilities.

4) The inclusion of a pipelining stage for enhancing the decoding capabilities of the STD in the presence of power supply variations.

Moreover, we analyze the different trade-offs presented in Figure 1.1 for both of the improved STD designs.

The rest of this chapter is structured as follows. Section 4.1 reviews the operation of turbo decoders. Section 4.2 details the hardware implementation of STDs. Section 4.3 describes how to improve the STD's tolerance to timing errors in order to avoid the catastrophic propagation of metastability through the circuit. Section 4.4 details the hardware implementation trade-offs of the proposed STD designs in the absence of timing errors. Section 4.5 characterizes the error correction performance of the proposed STD designs in the presence of timing errors. Section 4.6 presents our concluding remarks and offers design guidelines for timing-error-tolerant STDs.

4.1 Turbo Codes

In this section, we review the concepts of turbo encoding and turbo decoding [3] presented in Figure 4.1.

A turbo code comprises the parallel concatenation of two convolutional codes. As a result of this, a message frame comprising N bits $\mathbf{b}_1^{\mathrm{u}} = [b_{1,k}^{\mathrm{u}}]_{k=1}^{N}$ can be turbo encoded with the aid of the two parallel concatenated convolutional encoders, as shown in Figure 4.1(a). Each convolutional encoder operates in the same manner, with the upper convolutional decoder having the bits $\mathbf{b}_1^{\mathrm{u}}$ for its input. However, these bits are reordered by the interleaver Π to provide $\mathbf{b}_1^{\mathrm{l}}$, which is then input into the lower encoder. Here, the superscripts 'u' and 'l' indicate relevance to the upper and lower convolutional encoders, respectively. However, these superscripts will be omitted, when the discussion applies equally to both convolutional encoders.

Each convolutional encoder operates on the basis of a trellis such as the 8-state Long Term Evolution (LTE) trellis of Figure 4.1(c). Here, each bit of $\mathbf{b}_1$ triggers one of 16 different transitions among these states and the encoding of the first bit $b_{1,1}$ commences from a particular previous state s'. If tailbiting [81] is employed, then this initial state is

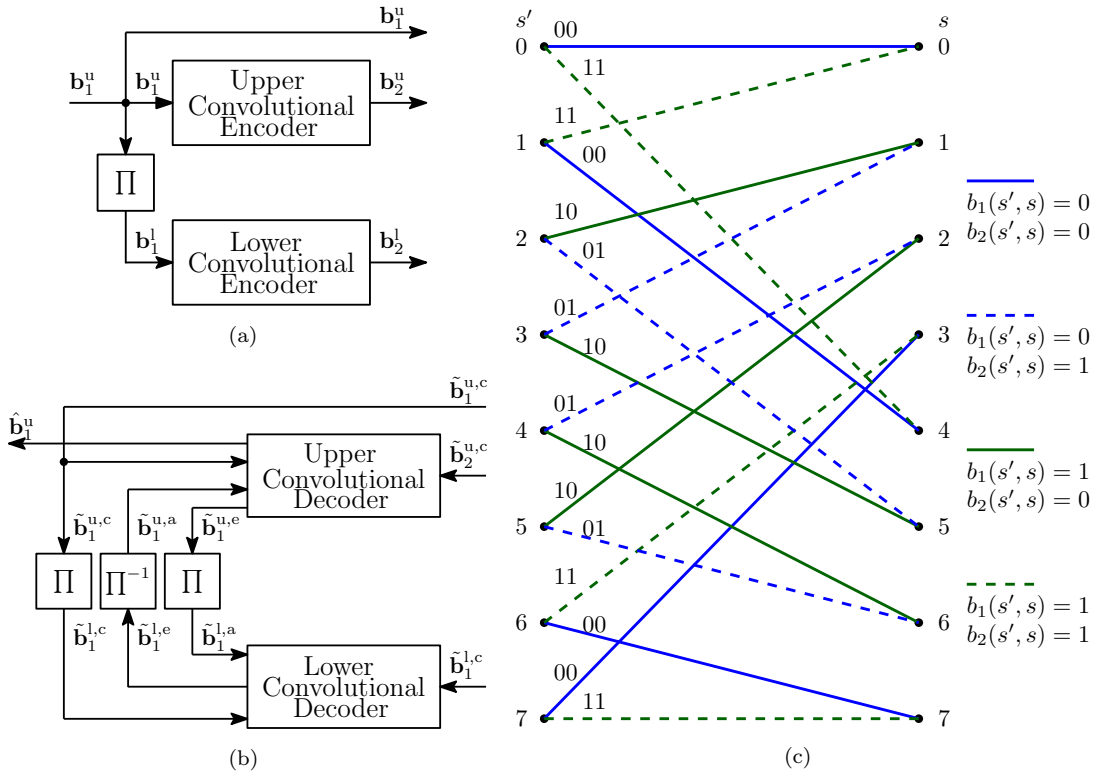


Figure 4.1: (a) Simplified turbo encoder. (b) Conventional structure of a turbo decoder. (c) State transition diagram of the LTE turbo code.

carefully selected such that it is equal to the final state s reached after encoding the final bit $b_{1,N}$. The simplified turbo encoder of Figure 4.1(a) outputs a frame of systematic bits $\mathbf{b}_1^{\mathrm{u}} = [b_{1,k}^{\mathrm{u}}]_{k=1}^{N}$ and two frames of parity bits $\mathbf{b}_2^{\mathrm{u}} = [b_{2,k}^{\mathrm{u}}]_{k=1}^{N}$ and $\mathbf{b}_2^{\mathrm{l}} = [b_{2,k}^{\mathrm{l}}]_{k=1}^{N}$, which are provided by the upper and lower convolutional encoder, respectively. After their transmission over a wireless channel, the received frames $\tilde{\mathbf{b}}_1^{\mathrm{u,c}} = [\tilde{b}_{1,k}^{\mathrm{u,c}}]_{k=1}^{N}$, $\tilde{\mathbf{b}}_2^{\mathrm{u,c}} = [\tilde{b}_{2,k}^{\mathrm{u,c}}]_{k=1}^{N}$ and $\tilde{\mathbf{b}}_2^{\mathrm{l,c}} = [\tilde{b}_{2,k}^{\mathrm{l,c}}]_{k=1}^{N}$ are entered into the turbo decoder, which comprises two convolutional decoders, as shown in Figure 4.1(b). The upper convolutional decoder of Figure 4.1(b) employs the received frames $\tilde{\mathbf{b}}_1^{\mathrm{u,c}}$, $\tilde{\mathbf{b}}_2^{\mathrm{u,c}}$ and the frame of *a priori* soft bits $\tilde{\mathbf{b}}_1^{\mathrm{u,a}}$ provided by the lower convolutional decoder, in order to provide the frame of *extrinsic* soft bits $\tilde{\mathbf{b}}_1^{\mathrm{u,e}}$. Following this, the *extrinsic* soft bits $\tilde{\mathbf{b}}_1^{\mathrm{u,e}}$ are interleaved and passed to the lower convolutional decoder as the frame of *a priori* soft bits $\tilde{\mathbf{b}}_1^{\mathrm{l,a}}$. The lower convolutional decoder employs the received frame $\tilde{\mathbf{b}}_2^{\mathrm{l,c}}$ and the interleaved received frame $\tilde{\mathbf{b}}_1^{\mathrm{l,c}}$ to provide the *extrinsic* soft bits $\tilde{\mathbf{b}}_1^{\mathrm{l,e}}$, which are de-interleaved in the block $\prod^{-1}$ and passed as the frame of *a priori* soft bits $\tilde{\mathbf{b}}_1^{\mathrm{u,a}}$ to the upper convolutional decoder. These soft bits express not only *what* the most likely value of the corresponding bits are, but also *how* likely these bit values are. More specifically, each soft bit $\tilde{b}_{1,k}^{\mathrm{u,c}}$ expresses the two probabilities $P^c(b_{1,k}^{\mathrm{u,c}} = 0)$ and $P^c(b_{1,k}^{\mathrm{u,c}} = 1)$, where the subscripts and superscripts may be replaced for the case of the other soft bits in Figure 4.1(b).

The convolutional decoders are iteratively operated on the basis of the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [63], which comprises Equations (4.1) to (4.6). The

$$\gamma_k(s',s) = P^{\mathrm{a}}(b_{1,k} = b_1(s',s))P^{\mathrm{c}}(b_{1,k} = b_1(s',s))P^{\mathrm{c}}(b_{2,k} = b_2(s',s)) \tag{4.1}$$

$$\gamma_k^{\mathrm{e}}(s',s) = P^{\mathrm{c}}(b_{2,k} = b_2(s',s)) \tag{4.2}$$

$$\alpha_k(s) = \frac{\sum\limits_{\text{all } s'} \gamma(s',s)\alpha_{k-1}(s')}{\sum\limits_{\text{all } (s',s)} \gamma(s',s)\alpha_{k-1}(s')} \tag{4.3}$$

$$\beta_{k-1}(s') = \frac{\sum\limits_{\text{all } s} \gamma(s',s)\beta_k(s)}{\sum\limits_{\text{all } (s',s)} \gamma(s',s)\beta_k(s)} \tag{4.4}$$

$$P^{\mathrm{e}}(b_{1,k} = j) = \frac{\sum\limits_{\text{all}(s',s)\to(b_1(s',s)=j)} \gamma_k^{\mathrm{e}}(s',s)\alpha_{k-1}(s')\beta_k(s)}{\sum\limits_{\text{all } (s',s)} \gamma_k^{\mathrm{e}}(s',s)\alpha_{k-1}(s')\beta_k(s)} \tag{4.5}$$

$$\hat{b}_{1,k} = \underset{j\in\{0,1\}}{\arg\max} \sum_{\text{all } (s',s)\to b_1(s',s)=j} \gamma_k(s',s)\alpha_{k-1}(s')\beta_k(s) \tag{4.6}$$

BCJR algorithm employs Equation 4.1 for calculating a branch metric $\gamma_k(s',s)$ for each transition of Figure 4.1(c) from a previous state s' into the next state s. Here, $P^{\mathrm{a}}(b_{1,k} = b_1(s',s))$ is the probabilities that are expressed by the *a priori* soft bit $\tilde{b}_{1,k}^{\mathrm{a}}$ provided by the other convolutional decoder. Note that at the start of the iterative decoding process, $P^{\mathrm{a}}(b_{1,k} = b_1(s',s)) = 0.5$ is assumed. The *extrinsic* branch metrics $\gamma_k^{\mathrm{e}}(s',s)$ of Equation 4.2 correspond to the received probability of the parity bit $b_{2,k}$ provided by the other convolutional encoder. Following this, the state metrics $\alpha_k(s)$ of Equation 4.3 and $\beta_k(s')$ of Equation 4.4 are calculated for quantifying the probabilities associated with each of the 8 possible previous and next states of Figure 4.1(c). Note that $\alpha_{k-1}(s') = 1/8$ and $\beta_k(s) = 1/8$ is assumed for all s' and s at the start of the iterative decoding process. If tailbiting [81] is employed, then $\alpha_0(s') = \alpha_N(s')$ and $\beta_N(s) = \beta_0(s)$ may be employed. Furthermore, Equation 4.5 is employed for determining the probabilities $P^{\mathrm{e}}(b_{1,k} = 0)$ and $P^{\mathrm{e}}(b_{1,k} = 1)$, which are expressed by the *extrinsic* soft bit $\tilde{b}_{1,k}^{\mathrm{e}}$. Finally, Equation 4.6 is employed for determining the *a posteriori* hard decision $\hat{b}_{1,k}$, pertaining to the bit $b_{1,k}$. This iterative process is repeated until an accurate estimation of the decoded frame $\hat{\mathbf{b}}_1$ can be obtained or until the maximum affordable number of iterations has been reached.

4.2 Stochastic Implementation of Turbo Decoders

This section reviews the hardware implementation requirements of the STD, which is briefly summarized in [39] and is detailed in [79], although the latter is written in French. Therefore, this thesis offers the first detailed treatment of the STD in English. In the fully-parallel stochastic decoding of turbo codes [39, 79], the block diagram in the blue box of Figure 4.2 is replicated for each bit decoded by each convolutional decoder of

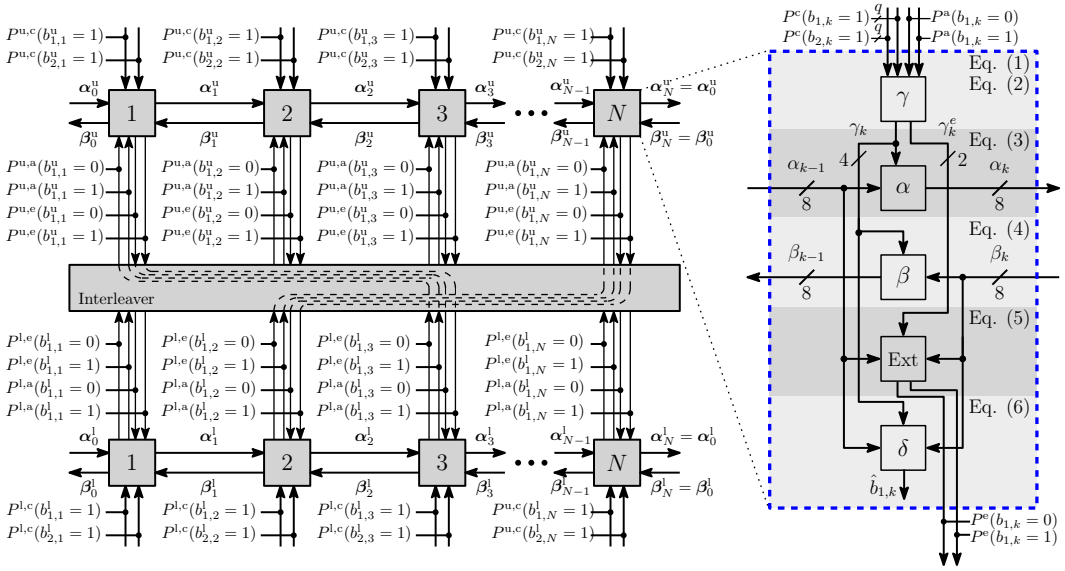


Figure 4.2: Block diagram of the fully-parallel STD.

Figure 4.1(b), with the two convolutional decoders being separated by a hard-wired interleaver, as shown in the left part of Figure 4.2.

The STD of [39, 79] adopts tailbiting [81] as described in Section 4.1. This guarantees that the initial and final states of the trellis of each convolutional decoder are identical. Owing to this, the state metrics $\alpha_N(s)$ output by the N^{th} block are provided as the inputs $\alpha_0(s')$ of the first block in each convolutional decoder, as shown in Figure 4.2. Likewise, the state metrics $\beta_0(s')$ output by the first block are provided as the inputs $\beta_N(s)$ of the N^{th} block. The incoming edges α_{k-1}, β_k, $P^{\mathrm{a}}(b_{1,k}=0)$ and $P^{\mathrm{a}}(b_{1,k}=1)$ of Figure 4.2 corresponds to one bit of a Bernoulli Sequence (BS) from a neighboring block. In a similar manner, the outgoing edges α_k, β_{k-1}, $P^{\mathrm{e}}(b_{1,k}=0)$ and $P^{\mathrm{e}}(b_{1,k}=1)$ corresponds to an outgoing bit of a BS. By contrast, the incoming edges $P^{\mathrm{c}}(b_{1,k}=1)$ and $P^{\mathrm{c}}(b_{2,k}=1)$ correspond to q-bit Fixed-Point (FX) probabilities provided by the channel. The block diagram of Figure 4.2 processes and exchanges one bit of each BS in each Decoding Cycle (DC). Furthermore, each block of Figure 4.2 corresponds to the stochastic implementation of Equations (4.1) to (4.6). Building on this, the following sections present the stochastic hardware implementation requirements of Equations (4.1) to (4.6).

4.2.1 Branch metrics

The module γ of Figure 4.2 performs the conversion of the q-bit FX representations of the received channel probabilities $P^c(b_{1,k}=1)$ and $P^c(b_{2,k}=1)$ into BSs. In addition to this, this module generates BSs representing the branch metrics $\gamma_k(s',s)$ and the *extrinsic* branch metrics $\gamma_k^e(s',s)$ of Equation 4.1 and Equation 4.2, respectively. The conversion of the received channel probabilities into BSs is achieved with the aid of two

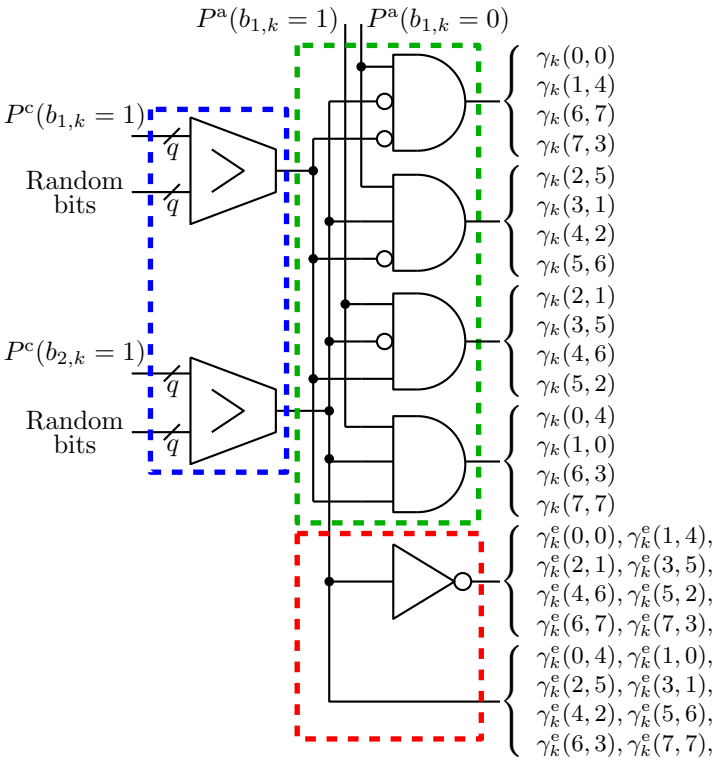


Figure 4.3: Stochastic realization of $\gamma_k(s', s)$ and γ_k^{e} [79, Fig. 2.5].

q-bit FX comparators and two q-bit pseudo-random numbers, as shown in the blue box of Figure 4.3. Here, q represents the number of quantization bits employed for representing the received channel probabilities and its value can be chosen based on the best trade-off between the advisable BER performance and the hardware requirements imposed, where $q = 7$ in the STD of [39, 79], for example. In this structure, $P^c(b_{1,k} = 1)$ and $P^c(b_{2,k} = 1)$ remain constant throughout the decoding process and the pseudo-random numbers change in every DC. The outgoing bit of each comparator is set to 1, if the corresponding probability is greater than the pseudo-random number and 0 otherwise. Following their conversion to BSs, the received channel probabilities are employed for obtaining the branch metrics $\gamma_k(s', s)$ of Equation 4.1. This is achieved with the aid of four 3-input AND gates, as shown in the green box of Figure 4.3, corresponding to the four possible combinations of the systematic and parity bits $b_1(s', s)$ and $b_2(s', s)$ shown in Figure 4.1(c). In addition to this, the *extrinsic* branch metrics $\gamma_k^e(s', s)$ of Equation 4.2 express the probabilities $P^c(b_{2,k} = 0)$ and $P^c(b_{2,k} = 1)$, pertaining to the parity bit $b_{2,k}$. As a result of this, the stochastic realization of $\gamma_k^e(s', s)$ can be implemented with two BSs for representing $P^c(b_{2,k} = 0)$ and $P^c(b_{2,k} = 1)$, as shown in the red box of Figure 4.3. The BSs representing the *a priori* probabilities $P^a(b_{1,k} = 0)$ and $P^a(b_{1,k} = 1)$ are provided by the other convolutional decoder, but this will not yet have generated any output during the first DC. Therefore, the first bits of the BSs can be initialized with random binary values during the first DC.

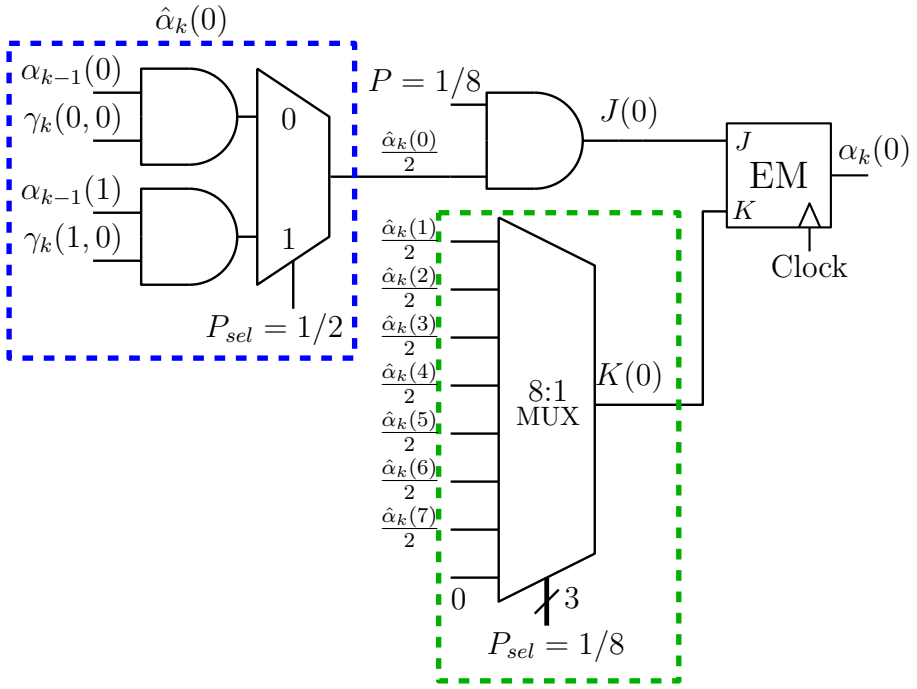


Figure 4.4: Stochastic realization of $\alpha_k(0)$ [79, Fig. 2.9].

4.2.2 State metrics

The modules α and β of Figure 4.2 compute the forward recursion of Equation 4.3 and the backward recursion of Equation 4.4, respectively. This is achieved with the aid of AND gates, Multiplexers (MUXs) and Edge Memories (EMs) for the multiplication, addition and normalization of BSs, respectively, as shown in Figure 4.4. In the following discussion, the stochastic implementation of the state metrics is described for the forward recursion state metrics $\alpha_k(s)$ of Equation 4.3, for the case where $s = 0$. The implementation of Equation 4.3 for all other states $s \in [1, 7]$ and of the backward recursion of Equation 4.4 can be performed following the same principles. In the LTE turbo decoder of [39, 79], Equation 4.3 can be modified as

$$\alpha_k(s) = \frac{\sum\limits_{\text{all } s'} \gamma(s', s)\alpha_{k-1}(s')}{\sum\limits_{\text{all } (s',s)} \gamma(s', s)\alpha_{k-1}(s')} = \frac{\hat{\alpha}_k(s)}{\sum\limits_{s=0}^{7} \hat{\alpha}_k(s)}, \tag{4.7}$$

where $\hat{\alpha}_k(s) = [\sum_{s'=0}^{7} \gamma(s', s)\alpha_{k-1}(s')]/2$ represents the non-normalized forward recursion. According to the state transition diagram of Figure 4.1(c), the state $s = 0$ can only be reached from the previous states $s' = 0$ and $s' = 1$. Owing to this, the term $\hat{\alpha}_k(0)$ of Equation 4.7, is simply $\hat{\alpha}_k(0) = [\gamma(0, 0)\alpha_{k-1}(0) + \gamma(1, 0)\alpha_{k-1}(1)]/2$. Therefore, the stochastic implementation of $\hat{\alpha}_k(0)$ can be performed by the circuit presented in the blue box of Figure 4.4, using a pair of 2-input AND gates and a 2-input MUX, as described in Section 2.3.1. JK-Type Flip-Flops (JKFFs) may be employed for performing the division required for normalizing probabilities, although they are susceptible to

the latching problem, as described in Section 2.3.2. Owing to this, the STD of [39, 79] employs the 32-bit Shift Register (SR)-based EMs of Figure 2.3, as shown in Figure 4.4. Hence, Equation 4.7 can be expressed as

$$\alpha_k(s) = \frac{\hat{\alpha}_k(s)}{\hat{\alpha}_k(s) + \sum\limits_{\bar{s}\in[0,7]\setminus\bar{s}=s} \hat{\alpha}_k(\bar{s})} = \frac{J(s)}{J(s)+K(s)}, \tag{4.8}$$

where $J(s) = [\hat{\alpha}_k(s)]/8$, $K(s) = [\sum_{\bar{s}\in[0,7]\setminus\bar{s}=s} \hat{\alpha}_k(\bar{s})]/8$ and $\bar{s} \in [0,7] \setminus \bar{s} = s$ denotes the exclusion of the state $\bar{s} = s$ from the set of states $\bar{s} \in [0,7]$. Here, $K(s)$ can be implemented with the aid of stochastic computing using an 8-input MUX with 3 pseudo-random selector bits representing a probability of $P_{sel} = 1/8$ and with one of the MUX inputs connected to logic 0, as shown in the green box of Figure 4.4. In addition to this, the factor of 8 division in the computation of $J(s)$ can be performed using an AND gate to multiply $\hat{\alpha}(s)$ with a BS representing the probability of 1/8. In the first DC, the inputs pertaining the BSs of $\alpha_{k-1}(s')$ can be initialized based on the best trade-off between BER performance and latency. To elaborate further, our simulations of Section 4.5 suggest that the decoding latency can be reduced when the bits of the BSs of the state metrics $\alpha_{k-1}(s')$ and $\beta_k(s)$ are initialized with random bit values. Moreover, the contents of the EM can be initialized as described in Section 2.3.2 during the first 32 DCs with a BS representing the probability $P = 0.5$. The resulting stochastic implementation of Equation 4.3, for the case of $s = 0$ is shown in Figure 4.4. Here, the inputs of the 8-input MUX are provided by structures similar to that shown in the blue box but corresponding to the other states, $s \in [1,7]$.

4.2.3 Extrinsic Probabilities

The calculation of the *extrinsic* probabilities of Equation 4.5 is performed by the module Ext of Figure 4.2. The stochastic implementation of Equation 4.5 can be implemented using the circuit of Figure 4.5.

In the following discussion, the stochastic implementation of the *extrinsic* probabilities of Equation 4.5 is described for the specific case where $b_{1,k}(s',s) = 0$, as represented by the blue box in Figure 4.5. The implementation of Equation 4.5 for the case where $b_{1,k}(s',s) = 1$ is represented by the green box in Figure 4.5 and can be performed following the same principles. In analogy to the discussion presented in Section 4.2.2, Equation 4.5 can be modified for the case where $b_{1,k}(s',s) = 0$ as

$$P^{\mathrm{e}}(b_{1,k}=0) = \frac{J(b_{1,k}=0)}{J(b_{1,k}=0)+J(b_{1,k}=1)}, \tag{4.9}$$

where

$$J(b_{1,k}=0) = \frac{\sum\limits_{\mathrm{all}(s',s)\to(b_1(s',s)=0)} \gamma_k^{\mathrm{e}}(s',s)\alpha_{k-1}(s')\beta_k(s)}{8}$$

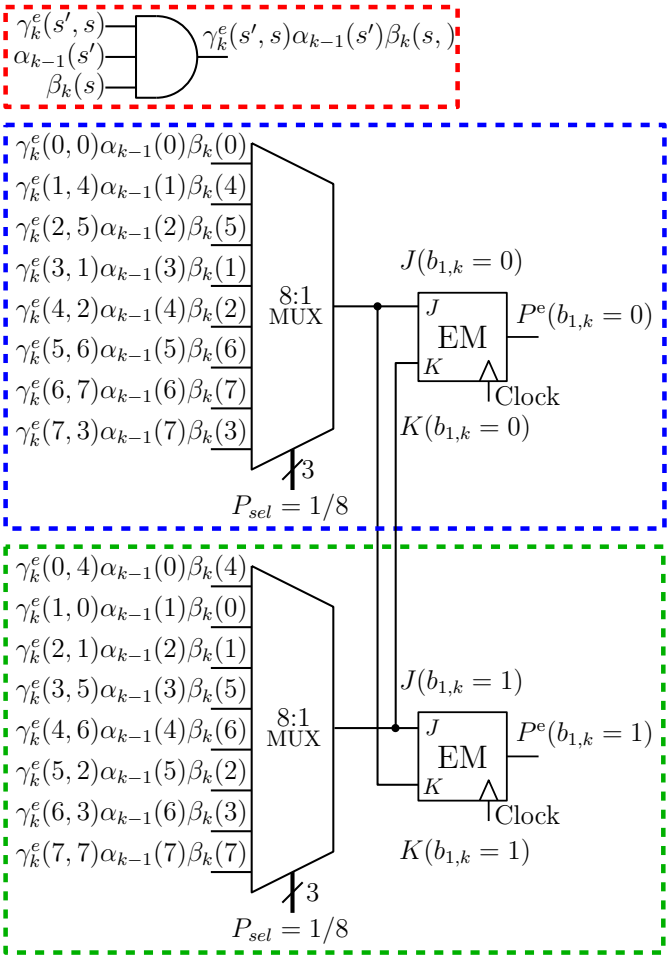


Figure 4.5: Stochastic realization of the calculation of the *extrinsic* probabilities in Equation 4.5 [79, Fig. 2.11].

and $J(b_{1,k}=1) = K(b_{1,k}=0)$ correspond to the set of state transitions engendered by the input bits $b_{1,k}=0$ and $b_{1,k}=1$, respectively. According to the state transition diagram of Figure 4.1(c), the input bit $b_{1,k}=0$ triggers the set of transitions $(s',s) =$ {(0,0), (1,4), (2,5), (3,1), (4,2), (5,6), (6,7), (7,3) }. As a result of this, $J(b_{1,k}=0)$ can be expressed as

$$\begin{aligned} J(b_{1,k}=0) = [&\gamma_k^e(0,0)\alpha_{k-1}(0)\beta_k(0)+ \\ &\gamma_k^e(1,4)\alpha_{k-1}(1)\beta_k(4)+ \\ &\gamma_k^e(2,5)\alpha_{k-1}(2)\beta_k(5)+ \\ &\gamma_k^e(3,1)\alpha_{k-1}(3)\beta_k(1)+ \\ &\gamma_k^e(4,2)\alpha_{k-1}(4)\beta_k(2)+ \\ &\gamma_k^e(5,6)\alpha_{k-1}(5)\beta_k(6)+ \\ &\gamma_k^e(6,7)\alpha_{k-1}(6)\beta_k(7)+ \\ &\gamma_k^e(7,3)\alpha_{k-1}(7)\beta_k(3)]/8. \end{aligned} \tag{4.10}$$

Here, $J(b_{1,k}=0)$ can be implemented with the aid of stochastic computing using a set of 8 3-input AND gates, one for each of the 8 multiplications of Equation 4.10, and one 8-input MUX for the averaging of the 8 product terms of Equation 4.10, as shown in the red and the blue box of Figure 4.5, respectively. In addition to this, the normalization of Equation 4.10 can be performed using the EM structure of Figure 2.3 with the input bits $J = J(b_{1,k}=0)$ and $K = J(b_{1,k}=1)$. Similar to the initialization of the EMs of the state metrics, the contents of the EM in this module can be initialized during the first 32 DCs with a BS representing the probability $P = 0.5$.

4.2.4 A Posteriori Probability

The estimation of the decoded bit $\hat{b}_{1,k}$ of Equation 4.6 is performed by the module δ of Figure 4.2. This is achieved in stochastic computing with the aid of AND gates and 8-input MUXs, as shown in Figure 4.6. The circuit of Figure 4.6 calculates the term $\hat{P}(b_{1,k}=0)$, which is proportional to the probability of the bit having the value 0, as shown in the blue box of Figure 4.6. This can be obtained by analyzing Equation 4.6 for the case where $b_{1,k}(s',s)=0$ and with the aid of the state transition diagram of Figure 4.1(c). In this way, $\hat{P}(b_{1,k}=0)$ can be expressed as

$$\begin{aligned}\hat{P}(b_{1,k}=0) = [&\gamma_k(0,0)\alpha_{k-1}(0)\beta_k(0)+\\ &\gamma_k(1,4)\alpha_{k-1}(1)\beta_k(4)+\\ &\gamma_k(2,5)\alpha_{k-1}(2)\beta_k(5)+\\ &\gamma_k(3,1)\alpha_{k-1}(3)\beta_k(1)+\\ &\gamma_k(4,2)\alpha_{k-1}(4)\beta_k(2)+\\ &\gamma_k(5,6)\alpha_{k-1}(5)\beta_k(6)+\\ &\gamma_k(6,7)\alpha_{k-1}(6)\beta_k(7)+\\ &\gamma_k(7,3)\alpha_{k-1}(7)\beta_k(3)]/8,\end{aligned} \tag{4.11}$$

The stochastic implementation of Equation 4.11 can be performed with the aid of 8 3-input AND gates, one for each of the 8 multiplications of Equation 4.11, and one 8-input MUX for the addition of the 8 product terms of Equation 4.11, as shown in the red and the blue box of Figure 4.6, respectively. The same principle can be applied for the stochastic implementation of $\hat{P}(b_{1,k}=1)$, which is proportional to the probability of the decoded bit being 1, as shown in the green box of Figure 4.6. Additionally, the estimation of the decoded bit $\hat{b}_{1,k}$ can be performed with a signed up/down saturated counter, as shown in Figure 4.6. The up/down counter increments its value if the BS representing $\hat{P}(b_{1,k}=0)$ takes the value of 1 and decreases its value if the BS representing $\hat{P}(b_{1,k}=1)$ takes the value of 1. The counter will not change its value if $\hat{P}(b_{1,k}=0)=\hat{P}(b_{1,k}=1)$. Lastly, the counter saturates its value if either the maximum or the minimum count value has been reached. In the STD of [39, 79], a 4-bit up/down counter with a maximum

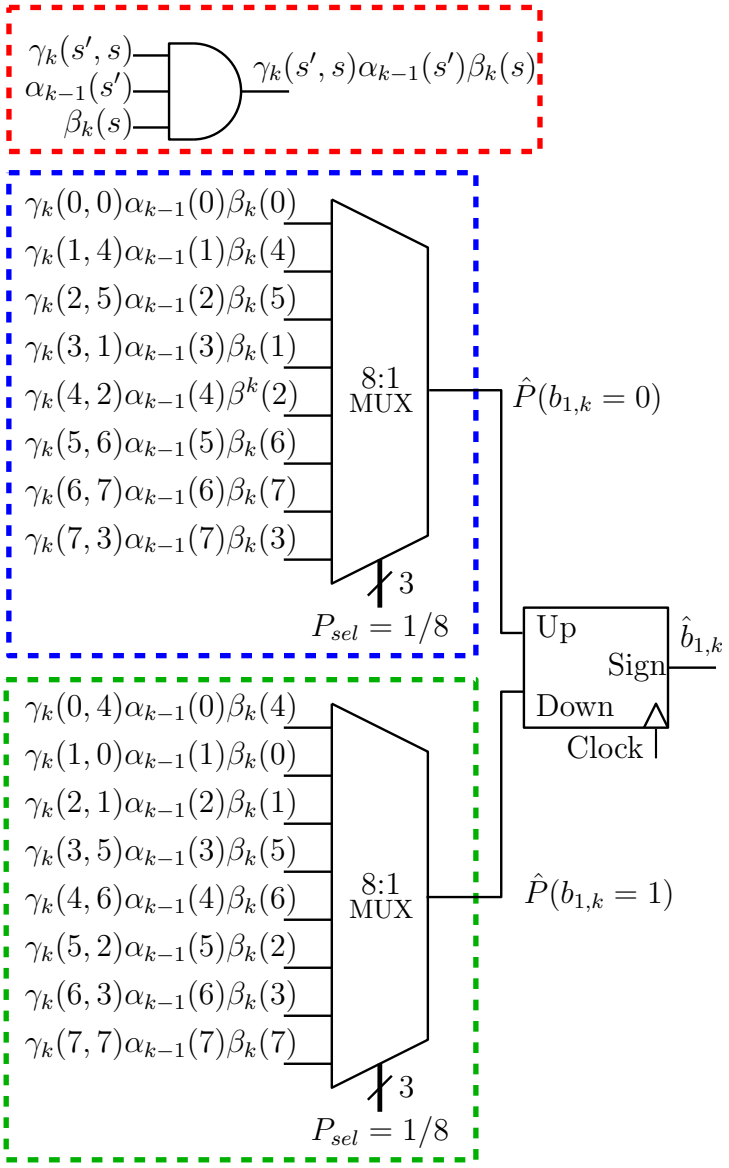


Figure 4.6: Stochastic realization of the calculation of the APP [79, Fig. 2.12].

value of +7 and a minimum value of -8 is employed. The estimation of the decoded bit $\hat{b}_{1,k}$ is performed in each DC by considering the sign bit of the saturated counter. In this way, if the value of the counter is ≥ 0, the estimated decoded bit is $\hat{b}_{1,k} = 0$ and 1 otherwise.

4.3 Timing-Error Tolerant Stochastic Turbo Decoder

Timing errors in synchronous systems occur when the clock period is not sufficiently long for all the signals to propagate from the output of DFFs, through the combinational logic and to the input of other DFFs. This occurs when techniques such as voltage-scaling or clock-scaling are employed for reducing the energy consumption or increasing the

throughput of the system. Furthermore, a reduction in the power supply, owing to voltage scaling or to power supply noise, increases the likelihood of the occurrence of processing errors, as a result of the quadratic dependency of propagation delays on the power supply. Whenever a timing error occurs, there is a chance that the affected DFF might enter into a metastable state, in which the digital signals have an indeterminate value that does not correspond to either a logic 0 or 1. However, given sufficient time, the metastable state will randomly evolve to a stable but unpredictable logic value of 0 or 1 [80]. This effectively imposes an additional propagation delay on the affected signal, hence increasing the likelihood of timing errors and metastability occurring at the next DFF. In this way, a single metastable event can trigger subsequent metastability occurrences in successive DFFs, causing catastrophic propagation of metastability that destroys the operation of the entire circuit. Moreover, metastability might cause undesired glitches, logic inconsistency and late transitions [80], which may result in the corruption of the bits stored in the EMs in the context of stochastic decoders, hence severely degrading the error correction capability of the decoder, as Chapter 3 demonstrated for the case of SLDPCDs. This motivates modifications to the STD of [39, 79] in order to enhance its tolerance to timing errors by preventing the catastrophic propagation of metastability. In the following sections, we present several novel enhancements to the STD of [39, 79], which not only improve its tolerance to timing errors, but also significantly improves its latency, throughput, energy efficiency and error correction capabilities in the presence of timing errors caused by power supply variations. In addition to this, we describe how each of these enhancements affects each of the design trade-offs presented in Figure 1.1. Each of these enhancements is detailed in the following subsections, which will show that they may be implemented by replacing Figure 2.3 with Figure 4.7 and Figure 4.6 with Figure 4.9.

4.3.1 Output Synchronizers for Mitigating the Catastrophic Propagation of Metastability

As mentioned above, variations in the power supply increase the likelihood of timing errors and metastability, which may catastrophically propagate through the system and destroy the entire operation of the STD. In order to reduce the probability of metastability cascading through the STD of [39, 79] owing to timing errors caused by power supply variations, we propose the replacement of the single output DFFs of the STD's α, β and Ext blocks shown in Figures 4.2, 4.4 and 4.5, so that they employ the synchronizer circuits of Figure 4.7, which comprise two DFFs [82], Wherever combinational logic is present between two DFFs, the occurrence of metastability in the first DFF increases the likelihood of metastability occurring at the second DFF, potentially causing the catastrophic propagation of metastability, as described above. However, since there is no combinational logic in the path between DFF1 and DFF2 of the synchronizers of Figure 4.7, the propagation delay associated with this path is negligible, when compared

to the clock period. As a benefit of this, if DFF1 enters into a metastable state due to a timing error, the time available for its metastability to be resolved is maximized and the probability of metastability cascading to DFF2 is significantly reduced [82].

The employment of synchronizer circuits is a commonly used technique for preventing the propagation of metastability during the transfer of data between different clock domains in asynchronous systems [82]. However, in our timing analysis presented in Section 4.5.1, we demonstrate that timing errors will frequently occur if the nominal operating conditions of the STD are reduced below the recommended safety margins for the sake of improving either the throughput or the energy consumption. Timing errors may also occur due to power supply variations. In addition to this, in the stochastic decoding of turbo codes presented in [39, 79], up to 250×10^3 DCs are needed before a reliable final decision can be obtained. More specifically, our analysis suggests that thousands of metastability events occur during the decoding of each frame, when the STD of [39, 79] is operated continuously in the presence of power supply variations. This results in a high likelihood of timing errors and metastability occurring and cascading through the circuit, destroying the decoding process and severely affecting the error correction capabilities of the STD, unless synchronizers circuits are employed.

The introduction of the synchronizer circuits enhances the decoding capabilities of the STD in the presence of timing errors caused by power supply variations, as we will demonstrate in Section 4.5. However, this modification increases the chip area, latency and energy consumption of the STD and reduces its throughput, owing to the increased number of clock cycles required for exchanging each bit of the BSs, as we will show in Section 4.4. In order to mitigate this throughput reduction, Section 4.3.2 describes how the STD can be modified to decode two frames concurrently.

4.3.2 Decoding of Two Frames Concurrently

The employment of the synchronizers described in Section 4.3.1 mitigates the probability of a single metastability event destroying the entire operation of the STD. However, this modification increases the chip area and increases the number of clock cycles required for exchanging each bit of the BSs, hence increasing both the latency as well as the energy consumption and reducing the throughput of the STD. Nonetheless, these additional clock cycles can be exploited for the concurrent decoding of a second received frame, in order to eliminate the throughput reduction. With this objective in mind, the STD can be modified to process information pertaining to alternate received frames in alternate clock cycles. The simultaneous decoding of two received frames has been previously proposed for the case of Low-Density Parity-Check (LDPC) codes only [83]. However, this thesis presents a novel technique for the simultaneous decoding of two received frames using STDs for the first time. This is achieved by modifying the EMs in the α, β and Ext blocks shown in Figures 4.2, 4.4 and 4.5 for simultaneously storing information

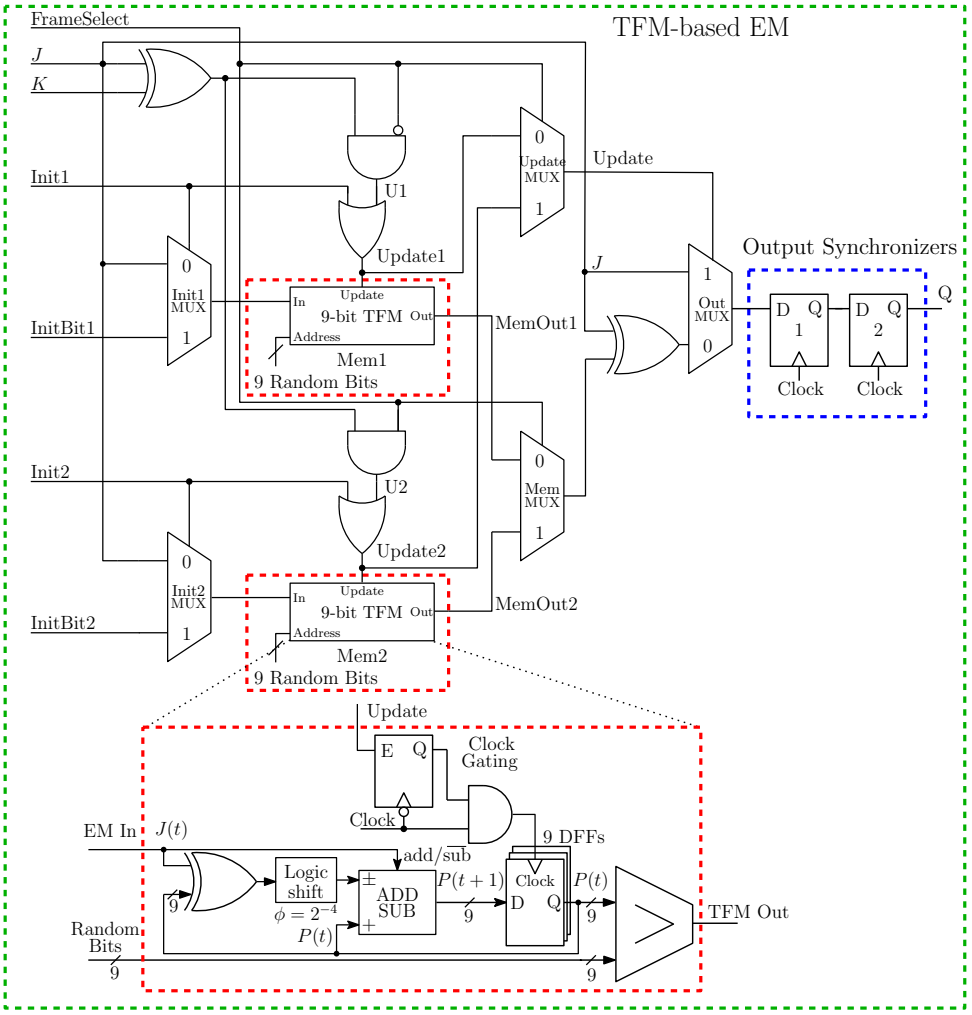


Figure 4.7: Modified TFM-based EM with output synchronizers, relying on two TFMs for concurrently decoding two frames and clock gating.

pertaining to each of the two received frames. Note that while this eliminates the above-mentioned throughput reduction that is introduced by the synchronizers, this does not eliminate the above-mentioned latency, chip area and energy consumption increases, as we will show in Section 4.4. Instead, these disadvantages of introducing synchronizers will be eliminated using the techniques described in

The green box of Figure 4.7 shows how the EM structures of Figures 2.3, 4.4 and 4.5 can be modified for the simultaneous decoding of two frames, by including two independent sets of memories, labeled Mem1 and Mem2, as shown in the red boxes. In this configuration, each set of memories will store information related to a particular decoded frame. Along with the additional memory, we introduce a FrameSelect signal and the logic required for controlling which specific set of DFFs to use in each clock cycle. The FrameSelect signal alternately adopts values of 0 and 1 in alternate clock cycles, alternating the activation of Mem1 and Mem2 for their update and read operations, according to their control signals. More specifically, the contents of Mem1 and Mem2

can be updated or read according to the value of the control signals U1 and Update1, for the case of Mem1, and U2 and Update2 for the case of Mem2. When the FrameSelect signal has a logic value of 0 and $J \neq K$, the signals U1 and Update1 adopt the value of 1, whereupon the contents of Mem1 are updated according to the regenerative bit J. In this scenario, the Update signal adopts the value of Update1=1, owing to the UpdateMUX, whereupon the regenerative bit J is passed to the first DFF of the synchronizers through OutMUX. By contrast, when the FrameSelect signal has a logic value of 0 and $J = K$, the U1 and Update1 signals adopt the value of 0. In this case, the Update signal adopts the value of 0 and a randomly selected bit from Mem1 is XORed with J and passed to the first DFF of the synchronizers through MemMUX and OutMUX. The operation of Mem2, for the case when FrameSelect=1, follows the same principle described above. Note that the initialization of each memory can be performed independently from each other and from the value adopted by FrameSelect. This is achieved using the signals Init1 and Init2, which directly dictate the value of Update1 and Update2 and control the initialization multiplexers Init1MUX and Init2MUX, corresponding to Mem1 and Mem2, respectively. Moreover, a memory can be initialized whilst the other memory is being employed during a decoding process. This allows one of the memories to be reset so that it can begin decoding a new frame, even if the other memory is still being used to decode a different frame. As described in Sections 4.2.2 and 4.2.3, each memory is initialized before the beginning of each decoding process with a BS corresponding to a probability of 0.5.

Along with the inclusion of the additional memories and control logic presented in Figure 4.7, the STD alternates between providing the corresponding systematic and the parity probabilities related to each of the two received frames. For the purpose of our investigation, we consider these sets of probabilities to be stored in a memory external to the decoder, which are selected according to the value of the FrameSelect signal, as shown in Figure 4.8. In this configuration, the set of MUXs labeled SysMUX in Figure 4.8 selects between the systematic probabilities $P^c(b_{1,k} = 1)$ of the received frames Frame1 and Frame2, when FrameSelect adopts the value of 0 and 1, respectively. Similarly, the received frame of parity probabilities $P^c(b_{2,k} = 1)$ of Frame1 and Frame2 is selected with the aid of the ParMUX, when FrameSelect adopts the value of 0 and 1, respectively. In each clock cycle, the selected probabilities $P^c(b_{1,k} = 1)$ and $P^c(b_{2,k} = 1)$ are provided to the γ module of the corresponding section of the STD.

As described above, the synchronizers and additional EMs increase not only the chip area, but also the energy consumption of the STD. To overcome these problems, we recommend the employment of low-power design techniques. In our investigation, we employ Clock Gating (CG) for reducing the dynamic energy consumption of the Application Specific Integrated Circuit (ASIC) implementation of the STD, as we will demonstrate in Section 4.4. More specifically, the dynamic energy consumption of the STD may be reduced by only enabling the clock signal of those specific DFFs, whose contents have

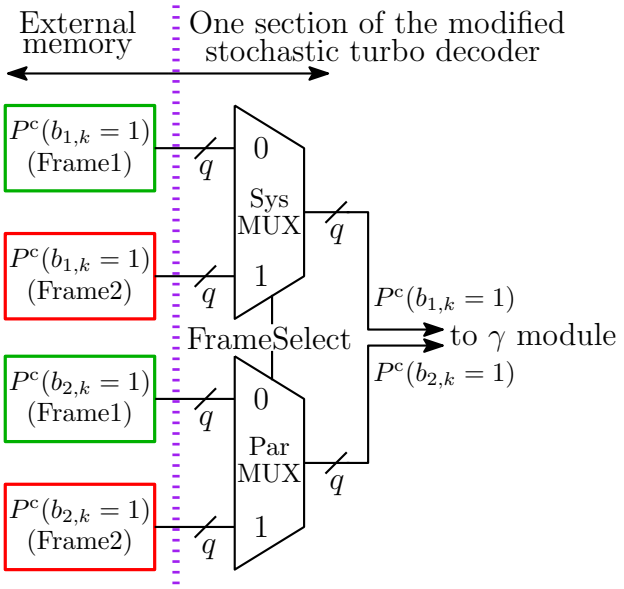


Figure 4.8: External set of systematic and parity probabilities for concurrently decoding two frames.

to be updated in a particular clock cycle, as shown in Figure 4.7. More specifically, the clock signal is AND gated with the output of an active-low latch, which is driven by the Update signal, as shown in Figure 4.7. Moreover, the area used by the clock-gated SR-based EMs will be significantly reduced, as we will show in Section 4.4, since the feedback MUX used for each DFF in the SR of Figure 2.3 will no longer be required. Instead, the DFFs will update their contents only when their clock signal is enabled, maintaining their current value otherwise. Furthermore, the associated propagation delays of the EMs are reduced, owing to the reduced fanout load of the Update signals, as detailed in Chapter 3 for the case of the SLDPCD. More specifically, the Update signal in the EM of Figure 2.3 has a fanout load of 32 Multiplexers, compared to a fanout load of a single latch in the clock-gated EMs of Figure 4.7.

When decoding two frames concurrently, the STD is required to estimate two sets of decoded bits $b_{1,k}$. Owing to this, an additional up/down counter is introduced in the δ module of Figures 4.2 and 4.6 for providing two independent decoded bits pertaining to the two independent decoding process of Frame1 and Frame2, as shown in the blue boxes of Figure 4.9. In this configuration, two clock-gating latches are employed for updating the counters labeled Counter1 and Counter2. More specifically, the contents of Counter1 determines the decoded bit $b_{1,k}$ of the decoded frame Frame1 and is only updated when FrameSelect=1. By contrast, the decoded bit $b_{1,k}$ for the case of the decoded frame Frame2 is determined when FrameSelect=0. As part of the trade-off analysis of the hardware implementation of the STD presented in Section 4.4, we will demonstrate that the chip area, latency, throughput and energy consumption of the STD is not significantly affected by the introduction of the additional counter.

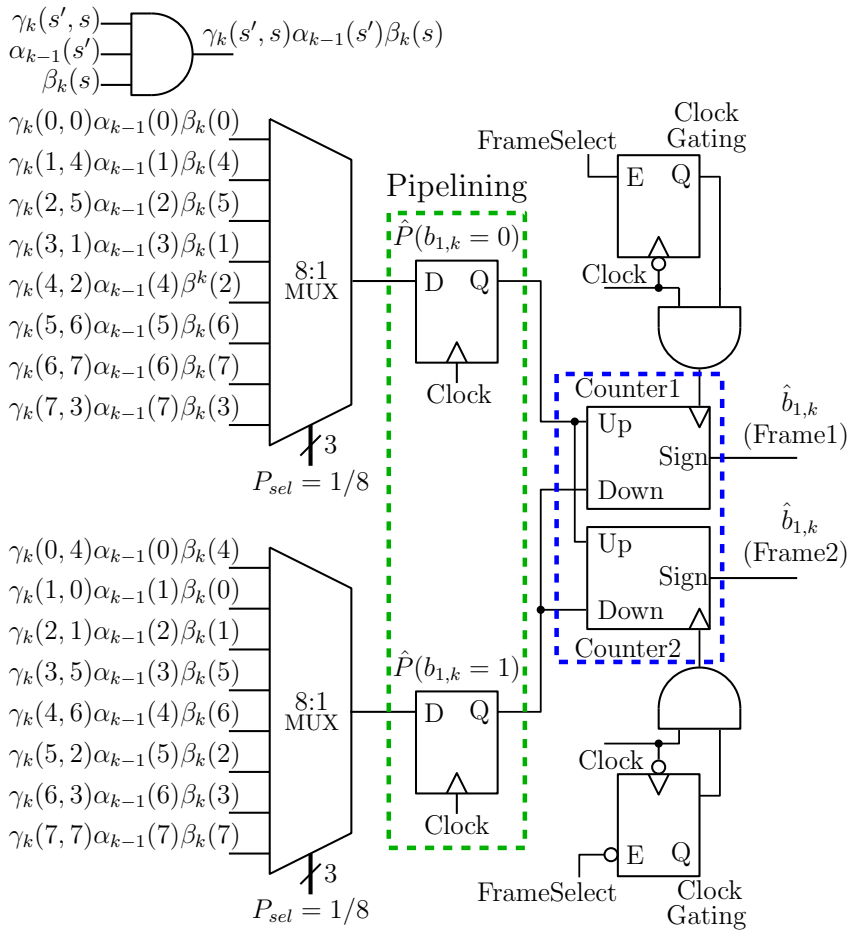


Figure 4.9: Modified stochastic realization of the calculation of the APP for concurrently decoding two frames.

4.3.3 Tracking Forecast Memory-based Edge Memories in Stochastic Turbo Decoders

As described in Section 4.3.2, the introduction of the additional EMs for the concurrent decoding of two received frames increases the chip area, latency and energy consumption of the STD. These problems can be overcome by the employment of TFM-based EMs, whilst enhancing the error correction capability of the STD in the presence of timing errors, as we will demonstrate in Sections 4.4 and 4.5. The employment of SR-based [27, 29] and TFM-based [30, 33] EMs has been proposed in order to overcome the latching problem in stochastic LDPC decoders. However, in the STD of [39, 79], only SR-based EMs have been previously employed. In this work, we demonstrate for the first time that the employment of TFM-based EMs, as shown in the lower part of Figure 4.7, is also beneficial in stochastic turbo decoding.

The TFM-based EM of Figure 4.10 was proposed in [30]. Here, the m-bit SR of Figure 2.3 is replaced by an n-bit TFM, which stores a FX binary number to quantify a moving average probability of the regenerative bit J assuming the value 1. This is achieved by

considering the previous bits of BSs, but placing special emphasis on the most recent bits, as detailed in [30]. More specifically, TFMs employ a decaying mechanism to ensure that only the most recent regenerative bits are considered for the calculation of the stored probability $P(t+1)$, which is calculated with the aid of the relaxation parameter ϕ for determining the significance given to the most recent regenerative bit J.

As in the SR-based EM, when $J \neq K$ in Figure 4.10, the FX probability $P(t+1) \in [0,1]$ of the TFM in time $t+1$ is updated according to:

$$P(t+1) = (1-\phi)P(t) + \phi J(t), \tag{4.12}$$

where $\phi \in (0,1)$ is the relaxation parameter, which can be chosen to optimize the BER performance of the stochastic LDPC decoder, and $J(t) \in \{0,1\}$ is the regenerative bit. By contrast, when $J = K$, the output of the TFM-based EM is determined by comparing the n-bit probability $P(t)$ to an n-bit pseudo-random number. If the probability stored by the TFM is larger than or equal to the random number, the outgoing bit is set to 1, otherwise it is set to 0.

Considering that J can only take the value of 0 or 1 at any given time, Equation 4.12 can be modified as follows:

$$\begin{aligned} P(t+1) =& (1-\phi)P(t) + \phi J(t) \\ =& P(t) - \phi P(t) + \phi J(t) \\ =& P(t) + \phi[J(t) - P(t)], \end{aligned}$$

which can be further simplified as

$$P(t+1) = \begin{cases} P(t) - \phi P(t) & J(t) = 0 \\ P(t) + \phi \bar{P}(t) & J(t) = 1, \end{cases} \tag{4.13}$$

where $\bar{P}(t) = 1 - P(t)$ is the complementary probability of $P(t)$. As a benefit of this, the two adders of the TFM of Figure 4.10 are substituted by a single adder/subtractor, where the complementary probability $\bar{P}(t)$ can be obtained using XOR gates, if $P(t)$ is stored as an unsigned FX number. Meanwhile, the multiplication $\phi \cdot P(t)$ can be readily implemented using a hard-wired logical shift, if the relaxation parameter ϕ is chosen as a negative power of 2, as shown in the lower red box of Figure 4.7. According to Equation 4.13, the complementary probability $\bar{P}(t)$ is only necessary if $J(t) = 1$. This functionality can be implemented with the aid MUXs and NOT gates, with the selector bits of the MUXs adopting the value of J, as described in [33]. Alternatively, the same functionality can be obtained with 2-input XOR gates, with one of the inputs adopting the value of J and the other input connected to the individual bits of $P(t)$, as shown in the lower part of Figure 4.7. Additionally, the add/$\overline{\text{sub}}$ signal determines whether an addition or a subtraction will be performed by the ADD/SUB block, when $J = 1$ and $J = 0$, respectively. The n-bit FX comparator is employed for determining the outgoing

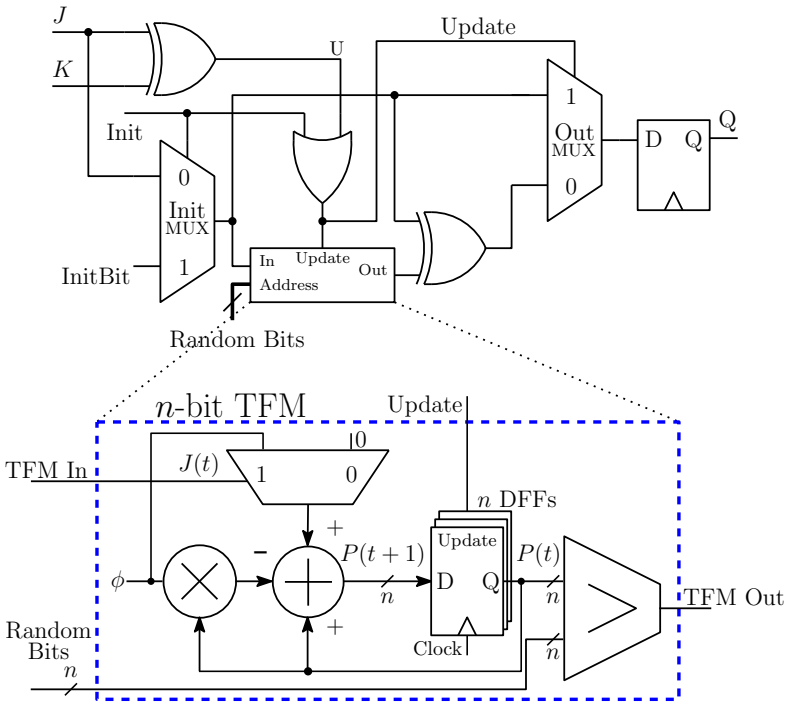


Figure 4.10: Architecture of TFM-based EMs, as proposed in [30].

bit when $J \neq K$, with TFMOut adopting the value of 1 if the probability stored in the TFM is larger than or equal to the n-bit pseudo-random number. Finally, at the start of the STD decoding process, each TFM is initialized to store the probability $P = 0.5$, which can be achieved in a single clock cycle by setting the Most Significant Bit (MSB) of each TFM to logic 1 and the rest of the bits to logic 0. This single clock cycle compares favorably to the 32 clock cycles required to initialize the SR Figure 2.3 with a given BS. The employment of TFMs in EMs effectively reduces the chip area requirements and energy consumption of the STDs, when compared to the employment of SRs in EMs in the STD of [39, 79]. More specifically, for the proposed modifications, we recommend the employment of 9-bit TFMs, with a relaxation parameter $\phi = 2^{-4}$. As a result of this, the proposed TFM-based EMs can be realized by using only 9 DFFs, compared to the 32 DFFs required in the SR-based EMs of [39, 79]. Moreover, the SR's 32 to 1 MUX of Figure 2.3 can be replaced with 9-bit FX comparators.

In addition to the advantages described above, TFMs are capable of tracking changes in the regenerative bit's probability more accurately than SRs, as detailed in [33]. This effectively reduces the number of DCs required for successfully decoding a frame when TFMs are employed, hence eliminating the potential latency increase resulting from the employment of the synchronizers of Section 4.3.1 Similarly, TFM-based EMs facilitate the use of lower clock periods than SR-based EMs, as Section 4.5 will show. This may be attributed to the relatively low complexity of the TFMs, as well as to the relatively low fanout loads imposed on their Update signal. Owing to this, the proposed TFMs-based EMs offer a desirable trade-off between chip area, energy efficiency, latency, throughput

and decoding performance, as Sections 4.4 and 4.5 will show. Figure 4.7 illustrates the resultant TFM-based EM circuit, relying on synchronizers, two sets of EMs and CG.

4.3.4 Pipelining

The role of the δ module of Figure 4.2 is to compute the APP and to make the final decision for the decoded bit $\hat{b}_k$ output by the STD. Owing to this, the occurrence of a timing error within this module, caused by power supply variations, might lead to an incorrect decision for a decoded bit, hence severely affecting the error correction capability of the STD. In order to overcome this problem, we propose the employment of a pipelining stage consisting of two DFFs placed in parallel, as shown in the green box of Figure 4.9. The additional DFFs break the combinational path that ends at the saturated counter. This reduces the time required for signals to propagate, hence reducing the occurrence of timing errors within the computation of the APP and improving the STD's error correction capability, as we will show in Section 4.5.2. Moreover, the employment of a pipeline stage does not impose a significant hardware overhead, as we will show in Section 4.4, since only two DFFs per decoded bit are employed. Similarly, the throughput and energy consumption of the STD are only slightly degraded, since the decision for the decoded bit $\hat{b}_{1,k}$ is only delayed by one DC.

The data flow of the modified STD is illustrated in Figure 4.11, where each set of light-shaded and dark-shaded blocks and DFFs store information pertaining to the decoding frames Frame1 and Frame2, respectively. In the upper part of Figure 4.11, the first set of memories Mem1 can represent either SR-based or TFM-based EMs. These EMs and the first DFF of the synchronizers will update their contents according to the bit of the BS resulting from the decoding process of Frame1 at time t. In time $t+1$, the contents of the second DFFs of the synchronizers will contain the stochastic bit generated at time t and pertaining to Frame1, owing to the synchronizers shifting their contents to the following DFF, as shown in the dark-shaded DFF of the lower part of Figure 4.11. In this same manner, the contents of Mem2 and the first DFFs of the synchronizers will update their value according to the decoding process of Frame2 in time $t+1$, as shown in the light-shaded DFFs of the lower part of Figure 4.11. Note that Counter1 is enabled one time instant after Mem1 is enabled, owing to the introduction of the pipelining stage. This is represented with a green solid line in the lower part of Figure 4.11. Alternatively, a red dashed line in Figure 4.11 represents either Counter1 or Counter2 in idle mode, in which their contents are not updated and no new estimation of $\hat{b}_{1,k}$ is made for that particular frame.

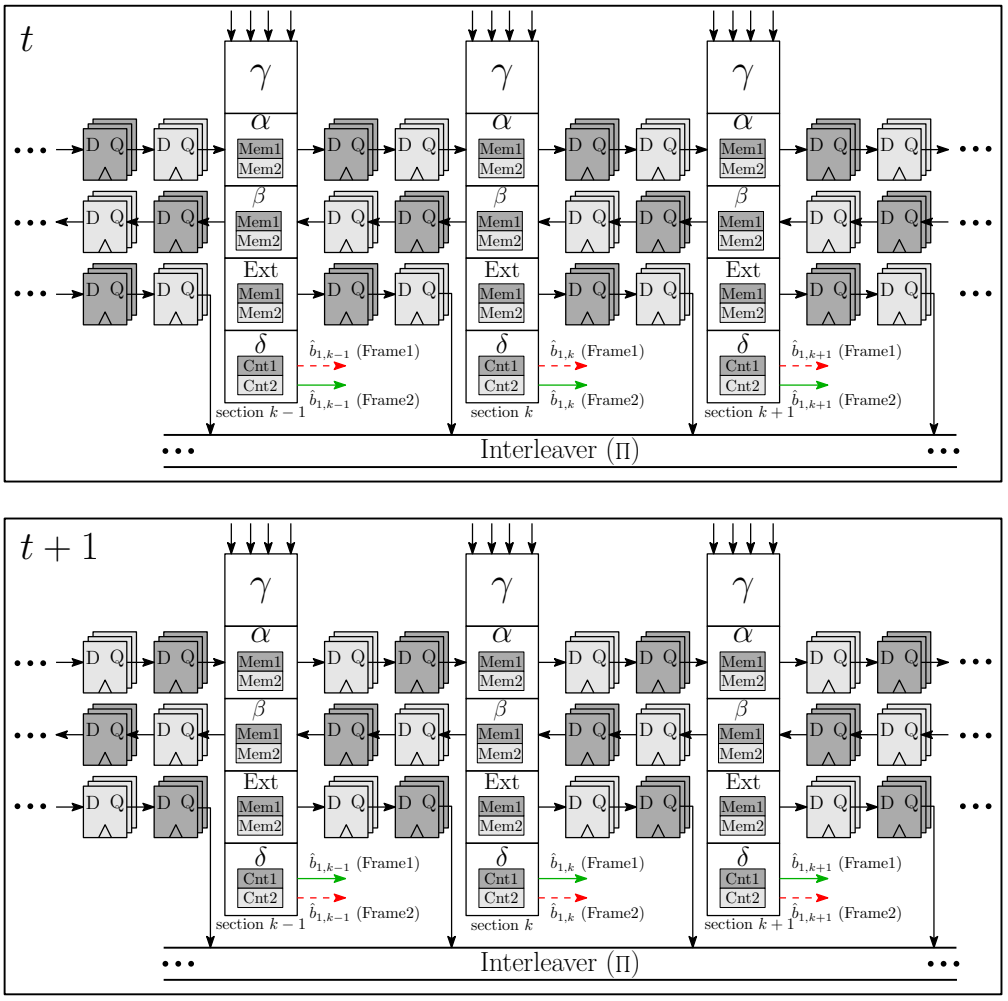


Figure 4.11: Data scheduling of the modified STD.

4.4 Trade-off Analysis of the Hardware Implementations of the Timing-Error Tolerant Stochasic Turbo Decoder

This section presents the different hardware trade-offs of various STD implementations. Table 4.1 characterizes the STD improvements of Section 4.3 in terms of diverse characteristics, including: the chip area per decoded bit, number of equivalent NAND gates, average number of DCs required for successfully decoding a frame, when employing early stopping, minimum clock period, latency, throughput and processing energy consumption per decoded bit, when using Taiwan Semiconductor (TSMC) 90 nm technology. These results are obtained for the implementation of an 8-state, 50-bit, rate 1/3, tailbiting STD using an S-Random interleaver and the state-transition diagram of Figure 4.1(c), as presented in [39]. Additionally, we employ Noise-Dependent Scaling (NDS) with $\eta = 1$ and $\psi = 2$, as described in Section 2.3.2. Here, we limit the implementation of the different STDs to the case of a 50-bit frame length, in order to reduce the complexity of our simulations and of the physical synthesis using automatic place and route. However, our simulations suggest that the hardware implementation results scale

Table 4.1: Hardware implementation results of the STD when operated at $E_b/N_0 = 3.0$ dB.

Scheme	Chip area per decoded bit	Average DCs	V_{DD} (V)	T_{clk} (ns)	Latency (μs)	Throughput (Kbps)	Energy (nJ/bit)
SR-1 (Benchmark)	0.0260 mm^2 8617 gates	14×10^3	1.20	4.0	56	892	335
			0.84	8.0	112	446	164
SR-2 (Synch.)	0.0260 mm^2 8744 gates	17×10^3	1.20	4.0	68	735	337
			0.84	8.0	136	367	165
SR-3 (Extra EMs)	0.0495 mm^2 15505 gates	14×10^3	1.20	4.5	126	793	643
			0.84	8.6	240	415	315
SR-4 (CG)	0.0397 mm^2 12853 gates	14×10^3	1.20	4.0	112	892	620
			0.84	7.6	212	469	303
SR-5 (Pipeline)	0.0397 mm^2 12933 gates	13×10^3	1.20	3.6	93	1068	621
			0.84	6.7	174	574	304
TFM-1 (TFM-based)	0.0129 mm^2 4596 gates	10×10^3	1.20	2.2	22	2272	55
			0.84	4.1	41	1219	27
TFM-2 (Synch.)	0.0129 mm^2 4724 gates	13×10^3	1.20	2.2	28	1748	66
			0.84	4.1	53	938	32
TFM-3 (Extra EMs)	0.0250 mm^2 8817 gates	10×10^3	1.20	2.3	46	2173	88
			0.84	4.2	84	1190	45
TFM-4 (CG)	0.0250 mm^2 8516 gates	10×10^3	1.20	2.2	44	2272	77
			0.84	4.1	82	1219	37
TFM-5 (Pipeline)	0.0250 mm^2 8531 gates	10×10^3	1.20	2.2	44	2272	77
			0.84	4.1	82	1219	37

approximately linearly with the frame length. Each scheme presented in Table 4.1 and in Figure 4.12 corresponds to the successive inclusion of each of the improvements described in Section 4.3, with SR-1 being the state-of-the-art STD of [39]. More specifically, SR-2 corresponds to the introduction of output synchronizers into SR-1; SR-3 refers to the introduction of the additional set of EMs into SR-2 to enable the concurrent decoding of two frames; SR-4 corresponds to the introduction of CG into SR-3 and finally, SR-5 refers to the introduction of the pipelining stage into SR-4. The schemes TFM-1 to TFM-5 present the TFM-based STDs counterparts of SR-1 to SR-5, with TFM-5 being the modified STD employing all enhancements proposed in Section 4.3.

The results of Table 4.1 were obtained from the physical layout generated by the automatic place and route of the above mentioned STDs using Cadence SoC Encounter [72]. The results of chip area per decoded bit were obtained from the layout of the γ, α, β, Ext and δ modules of Figure 4.2, where the γ module includes q=7-bit FX comparators for converting the probabilities provided by the channel into BSs. Additionally, the results of Table 4.1 were obtained for the cases, where the supply voltage of the STDs is set to either 1.20 V or 0.84 V, in the absence of power supply variations. Both the critical clock period and the energy consumption of the STDs are obtained from Synopsys PrimeTime [73]. The average number of DCs, latency, throughput and energy efficiency results were obtained from post-layout gate-level simulations, with extracted parasitics and annotated delays without timing errors, when allowing a maximum of 10^5 DCs and using early stopping. More specifically, we assume that a fully-parallel Cyclic Redundancy Check (CRC) [84] is employed in each DC for determining whether the

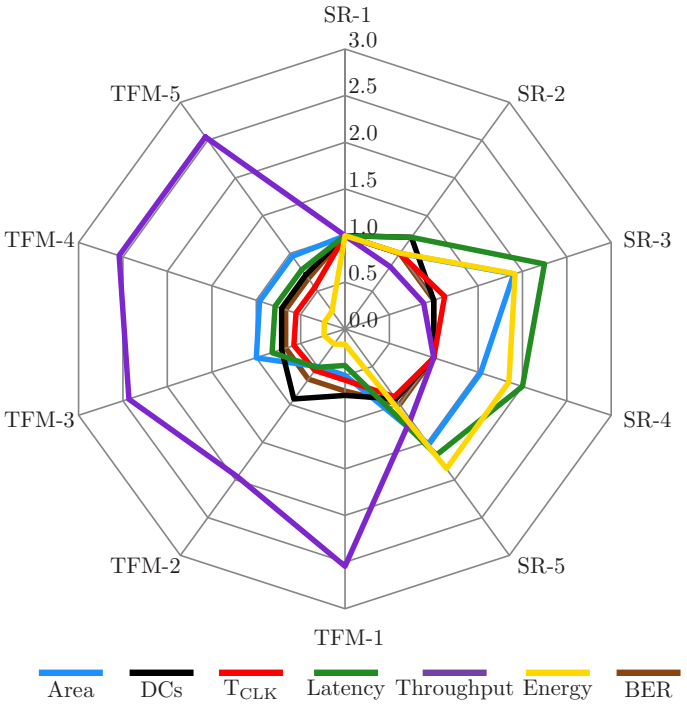


Figure 4.12: Hardware implementation results of the modified STD schemes. The presented results are normalized relative to SR-1, when $V_{DD} = 1.2$ V and $E_b/N_0 = 3.0$ dB.

frame of estimated decoded bits provided by the STDs contains any errors. Although the corresponding hardware characteristics are not considered in Table 4.1, they may be considered to have only a negligible effect, as detailed in [85]. We also assume that the different STDs operate at their critical clock period, for the case of Binary Phase Shift Keying (BPSK) communication over an Additive White Gaussian Noise (AWGN) channel and a Signal to Noise Ratio (SNR) per bit of $E_b/N_0 = 3.0$ dB. The notation (Scheme, V_{DD}, T_{clk}) will be used in the following sections in order to simplify the discussion of the results. In addition to Table 4.1, Figure 4.12 presents the hardware implementation trade-offs associated with each scheme of Table 4.1, relative to the benchmark scheme SR-1, when the various different STDs operate at $V_{DD} = 1.2$ V. Note that similar trends were found for the case when $V_{DD} = 0.84$ V.

We begin our discussion by analyzing the impact of each enhancement in the hardware implementation of the STD. This is followed by a comparison of the different hardware implementation trade-offs of SR-based and TFM-based STDs.

As discussed in Section 4.3.1 as well as confirmed in Table 4.1 and Figure 4.12, the employment of synchronizers increases the average number of DCs, latency and energy consumption and reduces the throughput of the STDs. This can be observed when comparing the SR-2 and TFM-2 schemes to the SR-1 and TFM-1 arrangements of Table 4.1 and Figure 4.12, respectively. More specifically, observe in the fourth column of Table 4.1 that SR-2 requires an average of 17×10^3 DCs for successfully decoding

a frame, compared to the 14×10^3 required in SR-1, when these schemes are operated at $E_b/N_0 = 3.0$ dB and achieve a BER of 4.6×10^{-4}, as we will show in Section 4.5.2. Likewise, TFM-2 requires an average of 13×10^3 DCs, compared to 10×10^3 DCs required by TFM-1 to achieve a BER of 2.8×10^{-4} at $E_b/N_0 = 3.0$ dB, which is the same BER that is achieved by the ideal Floating-Point (FP) Logarithmic BCJR (Log-BCJR) implementation employing 8 iterations. These differences are explicitly visualized in Figure 4.12.

The inclusion of the additional EMs for concurrently decoding two frames increases the chip area, extends the latency of the STDs, increases the energy consumption, but facilitates an increased throughput, as observed for the schemes SR-3 and TFM-3 of Table 4.1 and Figure 4.12, when compared to SR-2 and TFM-2, respectively. As seen in Table 4.1, the average numbers of DCs required by schemes SR-3 and TFM-3 are identical to those of SR-1 and TFM-1, respectively. However, the latency of SR-3 and TFM-3 is increased, when compared to SR-2 and TFM-2, respectively. This may be attributed to the decoding process of SR-3 and TFM-3 being completed in alternate clock cycles. Despite the increased latency, SR-3 and TFM-3 exhibit an increased throughput, when compared to SR-2 and TFM-2, respectively, owing to the simultaneous decoding of two frames, as detailed in Section 4.3.2. Similarly, SR-3 and TFM-3 exhibit an increased energy consumption owing to the additional EMs, when compared to SR-2 and TFM-2, respectively.

As detailed in Section 4.3.2 as well as observed in Table 4.1 and Figure 4.12, the employment of clock gating in SR-4 and TFM-4, reduces the chip area, the latency and the energy consumption and increases the throughput of the STDs, when compared to SR-3 and TFM-3, respectively. Particularly, the chip area reduction of SR-4 may be attributed to the elimination of the 32 MUXs needed in the SRs, as described in Section 4.3.2. By contrast, only the equivalent number of NAND gates of TFM-4 is slightly reduced, when compared to TFM-3. Additionally, as explained in Section 4.3.4, the inclusion of the pipelining stage does not significantly increase the area requirements of the STDs and does not extend the latency. As a result of this, the employment of clock gating and the pipelining stage facilitates lower clock periods, which is reflected in the improved latency, throughput and energy consumption of SR-4 and SR-5, when compared to SR-3, as well as of TFM-4 and TFM-5, when compared to TFM-3.

Let us now compare the hardware implementation trade-offs of SR-based and TFM-based STDs. When comparing the chip area requirements of SR-based and TFM-based STDs, Table 4.1 and Figure 4.12 show that TFM-based schemes exhibit lower area requirements than their SR-based counterparts. More specifically, TFM-1, TFM-2 and TFM-3 present area requirements that are about 0.50 times the area of SR-1, SR-2 and SR-3, respectively. Additionally, the implementation of TFM-4 and TFM-5 requires 0.62 times the chip area of SR-4 and SR-5, respectively. Furthermore, TFM-5, which presents all the proposed enhancements, requires 0.96 times the area of the benchmark

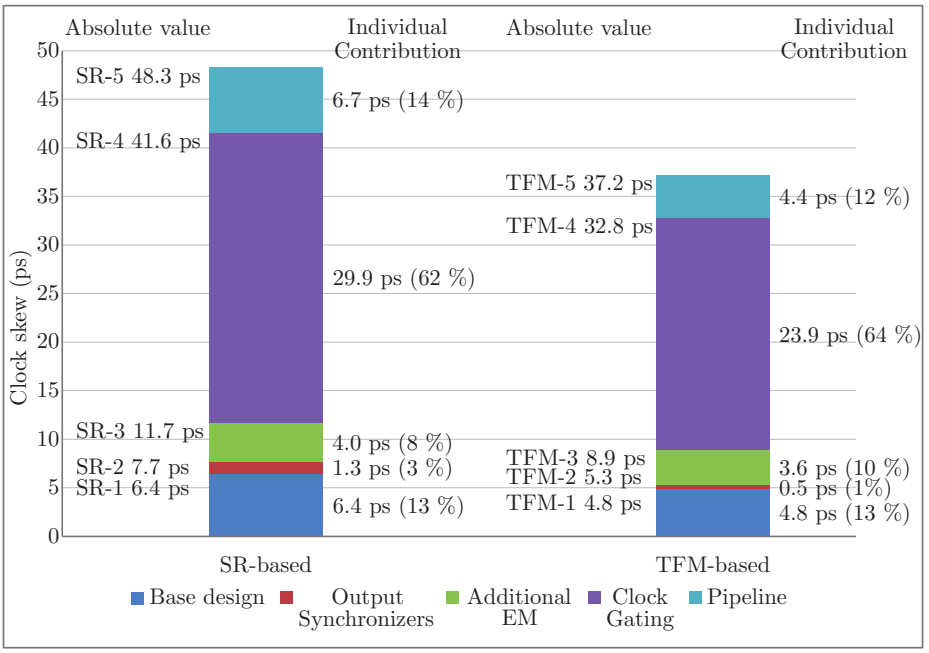


Figure 4.13: Clock skew of the different STD implementations.

SR-1. This lower area requirement of TFM-based schemes may be attributed to the TFM's employment of only 9 DFFs, instead of the 32 DFFs needed in a SR, as explained in Section 4.3.2. As explained in Section 4.3.3, TFM-based STDs require fewer DCs for achieving iterative decoding convergence, when compared to SR-based STDs. This, in addition to the reduced clock periods offered by TFM-based STDs, facilitate reduced latencies, reduced energy consumptions and increased throughputs, when compared to their SR-based STDs counterparts. As an example of this, the latency, throughput an energy consumption of (TFM-5, 1.20 V, 2.2 ns) are 0.78, 2.55 and 0.23 times those of (SR-1, 1.20 V, 4.0 ns), respectively. Furthermore, when the STDs operate at a supply voltage of 0.84 V, (TFM-5, 0.84 V, 4.1 ns) presents a latency of 82 μs, an energy consumption of 37 nJ/bit and a throughput of 1219 Kbps. These results suggest that (TFM-5, 0.84 V, 4.1 ns) may increase the throughput by a factor of 1.36, while consuming only 0.11 times the energy of that of (SR-1, 1.20 V, 4.0 ns), without increasing the chip area, albeit at the cost of increasing the latency by a factor of 1.46.

The significantly improved dynamic energy efficiency of TFM-5 relative to SR-1 may be mainly attributed to the use of TFM-based EMs, instead of SR-based EMs. More specifically, the energy consumption per TFM per DC in active- and standby-mode is only 6 nJ and 0.2 nJ, respectively, compared to 88 nJ and 4.37 nJ for an SR, when the supply voltage is set to 1.20 V. As detailed in Section 4.3.2, the static energy consumption may be expected to be higher in the SR-3 to SR-5 and TFM-3 to TFM-5 schemes than in the SR-1 to SR-2 and TFM-1 to TFM-2 arrangements, respectively. However, the static energy consumption of TFM-5 may be expected to be similar to that of SR-1, since these designs have similar chip areas and gate counts.

Figure 4.13 quantifies the effect of each proposed enhancement of Section 4.3 on the clock network of the STD. Each successive enhancement increases the clock skew of the STD, since each enhancement is achieved at the cost of increasing the number of DFFs in the STD, with the main source of clock skew in the proposed designs being the insertion of clock gating for the sake of reducing the dynamic energy consumption of the STDs. The increased clock skew of the proposed designs increases the likelihood of timing errors occurring owing to power supply variations. As part of our design methodology, these clock skew results are considered, when performing the timing analysis of the different STDs implementations in Section 4.5.1 for determining the critical clock period of each design.

4.5 Error Correction Capabilities of the Timing-Error Tolerant Stochastic Turbo Decoders

In this section, we characterize the decoding performance of the STD in the presence of timing errors, when BPSK modulation is used for communication over an AWGN channel.

4.5.1 Timing Error Model

Supply voltage variations in an ASIC may be caused by effects such as IR-drop, $L \cdot d_i/d_t$ noise, crosstalk, electrostatic discharges, particle strikes, switching noise and fabrication process variations [12, 13, 14, 15, 16], among other causes. In accordance with [77], we model these effects by representing the supply voltage of the ASIC with a Gaussian distribution having a mean of μ, which represents the nominal supply voltage, and a standard deviation of σ, which represents the power supply variations. In our analysis, the clock period T_{clk}, the nominal supply voltage μ and the degree of power supply variations σ are fixed for the whole operation of the decoder [77]. However, a different random value of V_{DD} is selected from the Gaussian distribution for each clock cycle, which is then used for all gates in the STD during the current clock cycle. The selected values of μ are 1.20 and 0.84, corresponding to the nominal supply voltage of TSMC 90 nm technology and this value scaled down 30%, respectively. The value of σ is selected to obtain particular values of $3\sigma/\mu$, namely 0.03, 0.05 and 0.07 which correspond to the three standard deviation variation fraction of the selected supply voltage, as detailed in [78]. Similarly, the selected values of T_{clk} correspond to the critical clock periods of each scheme presented in Section 4.4. More specifically, we considered the clock skews of Section 4.4 and a short-path timing analysis for creating feasible timing budgets and for determining the corresponding critical clock period of each design.

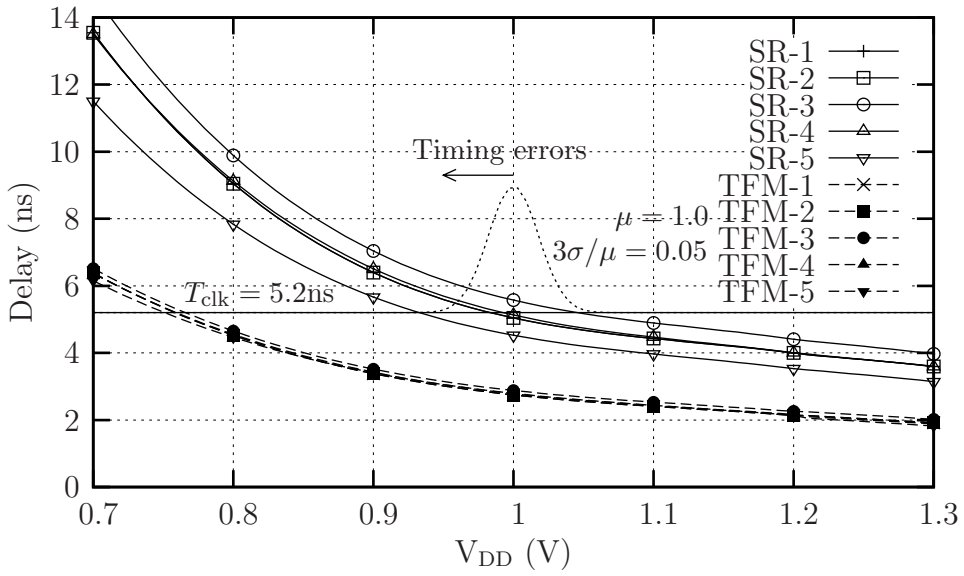


Figure 4.14: Delays of critical paths of different implementations of the STD.

In order to characterize the presence of timing errors in the STD, we performed a post-layout timing analysis with extracted parasitics and annotated delays of the different TSMC 90 nm implementations of the STDs for a range of supply voltages. Figure 4.14 presents only the largest propagation delays of the various STDs under different values of the supply voltage. However, for the purpose of our timing error model, we consider the critical path delay of every DFF in each of the different STDs separately. In each clock cycle, the chosen value of V_{DD}, T_{clk} and Figure 4.14 are used for determining the delay encountered by each signal propagating to each DFF. Moreover, Figure 4.14 reveals that in the SR-1 STD, timing errors are more likely to occur in the Ext and δ modules of Figure 4.2, owing to their long critical paths. However, these critical paths have been significantly reduced by the inclusion of the pipeline stage in SR-5, as shown in Figure 4.14.

Since the STDs operate at their critical clock period, timing errors will be encountered if the selected value V_{DD} is smaller than the nominal supply voltage μ. Similarly, we assume that timing errors occur, whenever the delays are larger than the fixed T_{clk}. As a result of this, some paths will experience timing errors in some clock cycles, but not in others. Similarly, a large overall number of timing errors will occur in some clock cycles and only a small number of timing errors will be encountered in others. If a timing error is indeed encountered, our error model assumes that a random bit value will be clocked into the affected DFFs and the same value will be propagated through the circuit in the subsequent clock cycles. This random value responds to unpredictable glitches and late transitions caused by timing errors, as shown in Figure 4.15, as well as resolved metastable states owing to the use of synchronizers [80]. For illustrative purposes, Figure 4.14 includes a Gaussian distribution associated with $\mu = 1.0$, $3\sigma/\mu = 0.05$ (nominal V_{DD} of 1.0 V and three-sigma standard deviation power supply variation of

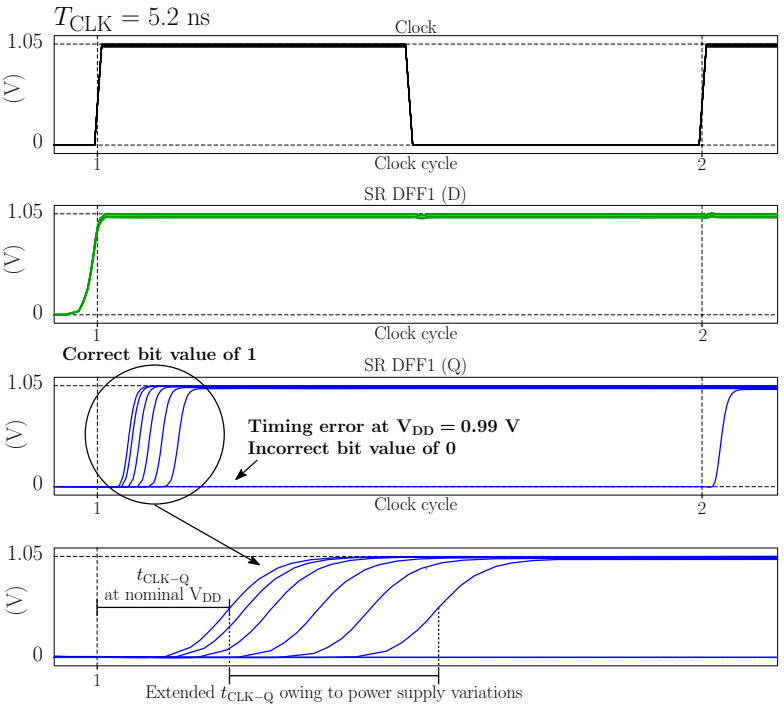


Figure 4.15: SPICE simulation of a critical path under different supply voltages of $V_{DD} \in \{0.99, 1.00, 1.01, 1.02, 1.03, 1.04, 1.05\}$ V.

5%) and a clock period of $T_{clk} = 5.2$ ns. In this particular example, SR-3 will be subject to timing errors if the selected value of V_{DD} drops below 1.05 V, since the critical path delay exceeds the critical clock period in this condition, as shown in Figure 4.15. More specifically, the waveforms of Figure 4.15 were obtained using SPICE simulations of the critical path in the SR-3 scheme, when using ST 90 nm technology operated at different supply voltages and at a fixed critical clock period of $T_{CLK} = 5.2$ ns. Owing to this, the signal output by the critical path arrives at the D input of the first DFF in the SR-based EM at the same time, as the first clock edge shown in Figure 4.15. Here, power supply voltages of 1.00 V $\leq V_{DD} \leq$ 1.05 V are sufficiently high to avoid generating a timing error in the DFF. However, if the supply voltage is reduced to $V_{DD} = 0.99$ V, a timing error is observed. In this scenario, a bit value of 0 is erroneously stored in the SR DFF and presented at its Q output after the first clock edge of Figure 4.15. In addition to this, Figure 4.15 demonstrates that the propagation delay t_{CLK-Q} of the DFF is increased, when the supply voltage is reduced, increasing the likelihood of timing errors and metastability occurring in the subsequent DFFs, as detailed in Section 4.3. By contrast, the TFM-based STDs will not experience any timing errors in this configuration, since their critical paths have lower critical clock periods.

4.5.2 Decoding Performance in the Presence of Timing Errors

The notation (Scheme, V_{DD}, T_{clk}, %) will be used in this section to refer to a specific STD implementation operated at the given supply voltage, clock period and three standard

deviation percentage of power supply noise $3\sigma/\mu$, respectively.

Figure 4.16 presents the BER and hardware performance of the SR- and TFM-based STD, when operated at V_{DD} = 1.20 V. Figure 4.16(a) presents the BER performance for a FP implementation of the conventional Log-BCJR turbo decoder using 8 decoding iterations as a benchmark. We also present BER plots of the SR- and TFM-based STD, when allowing a maximum of 10^5 DCs in the absence of timing errors, which confirm that STDs are capable of achieving similar BER performances as the ideal FP turbo decoder. Note that SR-1 to SR-5 offer identical BER performance in the absence of timing errors, since the proposed modifications only impact the BER in the presence of timing errors. Likewise, TFM-1 to TFM-5 offer identical BER performance in the absence of timing errors. As mentioned in Section 4.3.1, our timing analysis of Section 4.5.1 suggests that thousands of metastability events occur when the different STDs are operated continuously in the presence of power supply variations. Since SR-1 and TFM-1 do not consider the employment of synchronizers for preventing the catastrophic propagation of metastability, we do not plot their BER in the presence of timing errors, since it would be very poor. Figure 4.16(a) presents the BER of the modified schemes SR-4, SR-5, TFM-4, TFM-5 when operated at V_{DD} = 1.20 V. Note that the BER performances of SR-2, SR-3, TFM-2 and TFM-3 are not included in Figure 4.16(a), since SR-2 and SR-3 offer similar BER performance to that of SR-4. Similarly, TFM-2 and TFM-3 offer similar BER performance to that of TFM-4. However, schemes SR-2 and SR-3 and schemes TFM-2 and TFM-3 offer different chip area, latency, throughput and energy consumption, when compared to SR-4 and TFM-4, respectively, as described above in Section 4.4.

Figure 4.16(a) demonstrates that the proposed STDs offer an enhanced tolerance to timing errors. As an example of this, Figure 4.16(a) shows that (SR-4, 1.20 V, 4.0 ns, 3%) exhibits an E_b/N_0 degradation of about 0.14 dB respect to the ideal BER performance of SR-1. Similarly, (TFM-4, 1.20 V, 4.0 ns, 5%) offers similar BER performance of that of the TFM-based STD in the absence of timing errors. However, schemes SR-4 and TFM-4 exhibit an E_b/N_0 degradation of about 0.75 dB and 0.5 dB, when the three standard deviation percentage of power supply noise is increased to 5% and to 7%, as shown in Figure 4.16(a) in (SR-4, 1.20 V, 4.0 ns, 5%) and (TFM-4, 1.20 V, 4.0 ns, 7%), respectively,. Moreover, Figure 4.16(a) demonstrates that the inclusion of the pipeline stage described in Section 4.3.4 improves the BER performance of the STDs by preventing the occurrence of timing errors during the estimation of the decoded bit $b_{1,k}$. This is shown in the BER performance of (SR-5, 1.20 V, 4.0 ns, 5%), which exhibits similar BER performance of SR-1. Likewise (TFM-5, 1.20 V, 4.0 ns, 7%) presents similar error correction capabilities to that of (SR-1, 1.20 V, 4.0 ns, 0%), which corresponds to the state-of-the-art STD of [39] with no timing errors. This corresponds to an E_b/N_0 degradation of about 0.1 dB, when compared to TFM-1 in the absence of timing errors.

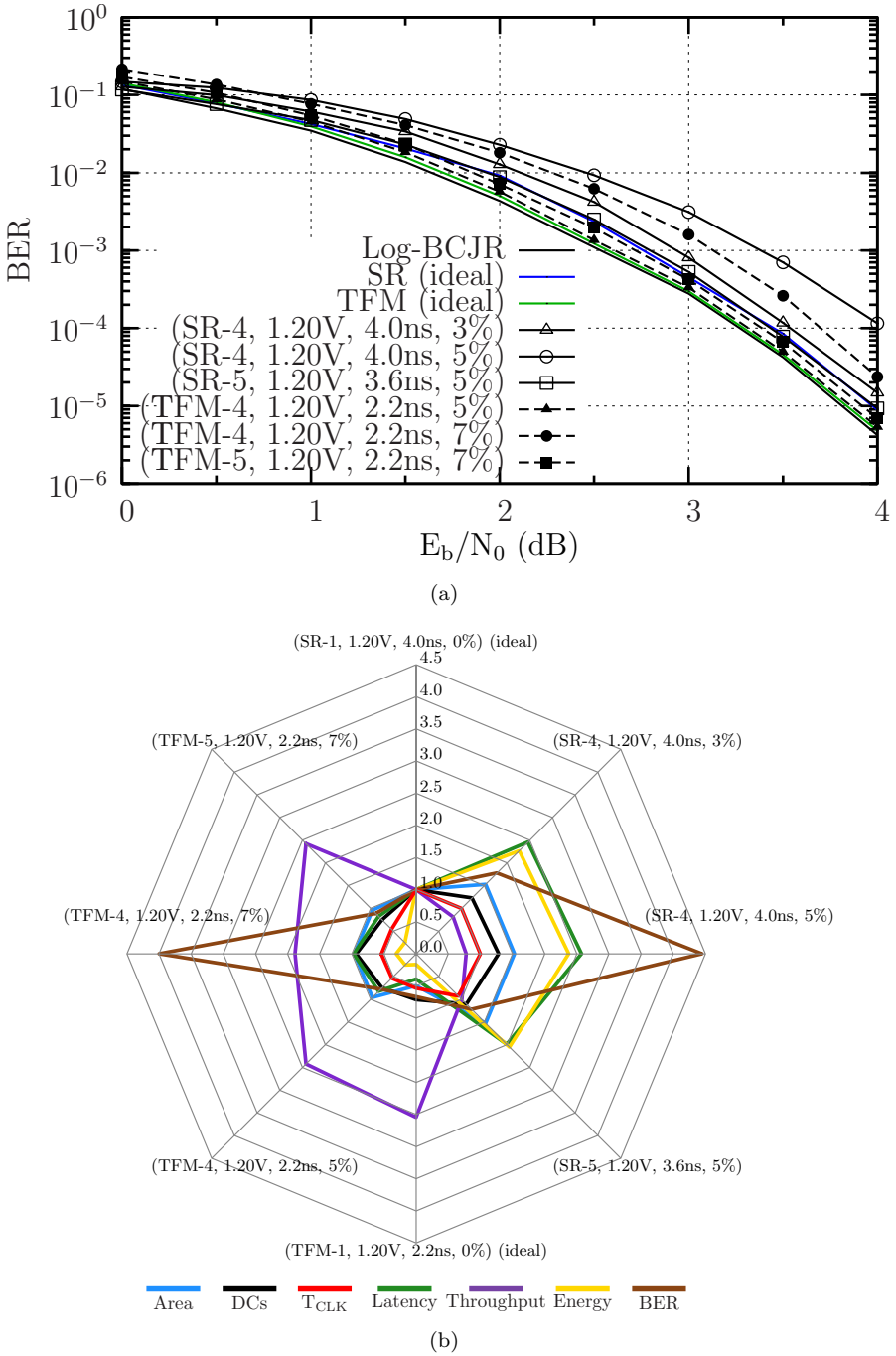
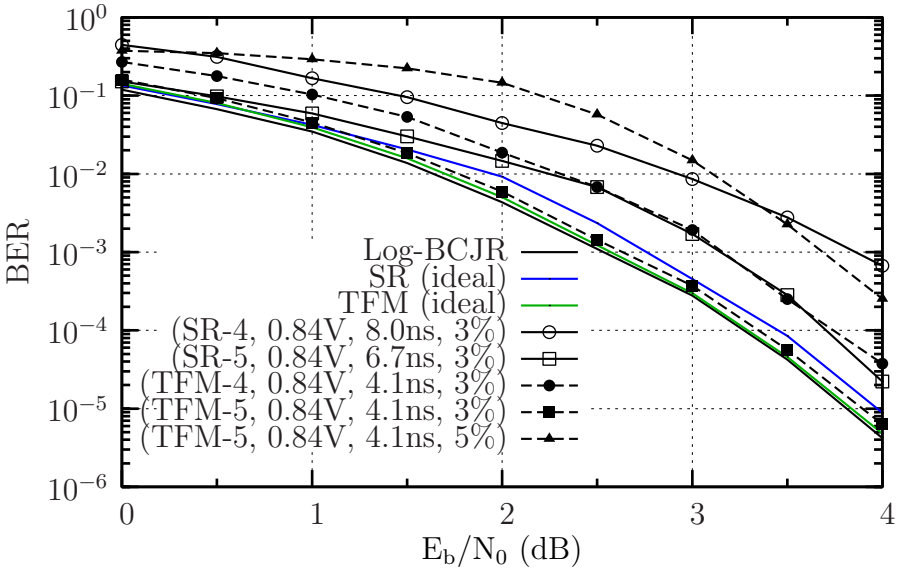
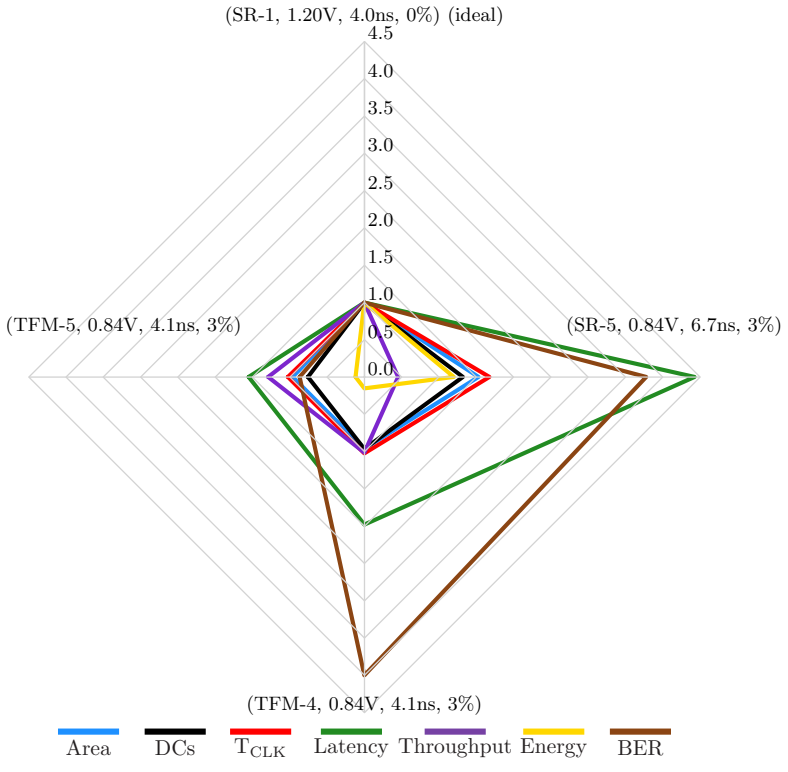


Figure 4.16: BER and hardware and performance of the modified STDs in the presence of timing errors and power supply variations when V_{DD} = 1.20 V: (a) BER performance of the modified STDs operated at V_{DD} = 1.20 V. (b) Hardware performance of the modified STD operated at V_{DD} = 1.20 V and E_b/N_0 = 3.0 dB.

In addition to the BER performance, Figure 4.16(b) shows the chip area, average number of DCs, latency, throughput and energy consumption per decoded bit of the proposed STDs, normalized relative to the benchmark SR-1 in the absence of timing errors, when the different STDs operate at V_{DD} = 1.20 V and E_b/N_0 = 3.0 dB. We also present



(a)



(b)

Figure 4.17: BER and hardware and performance of the modified STDs in the presence of timing errors and power supply variations when V_{DD} = 0.84 V: (a) BER performance of the modified STDs operated at V_{DD} = 0.84 V. (b) Hardware performance of the modified STDs operated at V_{DD} = 0.84 V and E_b/N_0 = 3.0 dB.

the hardware performance of TFM-1 in the absence of timing errors as a benchmark. Figure 4.16(b) shows that the average number of DCs required by (SR-4, 1.20 V, 4.0 ns, 3%), (SR-4, 1.20 V, 4.0 ns, 5%) and (SR-5, 1.20 V, 3.6 ns, 5%) is 1.23, 1.29 and 1.10

times that of SR-1 in the absence of timing errors, respectively. This, in addition to the decoding of two frames in alternate clock cycles, is reflected in increased latencies and energy consumptions. However, (SR-5, 1.20 V, 3.6 ns, 5%) exhibits a similar throughput to that of SR-1 in the absence of timing errors, albeit at the cost of an increased BER. Figure 4.16(b) shows that TFM-based decoders in the presence of timing errors offer a reduced number of DCs and latencies, which facilitate increased throughputs and reduced energy consumptions, when compared to the benchmark SR-1 in the absence of timing errors. More specifically, the latency, throughput and energy consumption of (TFM-5, 1.20 V, 2.2 ns, 7%) are 0.83, 2.42 and 0.25 times those of SR-1 in the absence of timing errors, respectively.

Figure 4.17(a) demonstrates that the BER performance of the different STDs is severely affected when they are operated at V_{DD} = 0.84 V and in the presence of power supply variations, owing to the quadratic dependency of delays on the power supply. More specifically, Figure 4.17(a) shows that (SR-4, 0.84 V, 8.0 ns, 3%) exhibits an E_b/N_0 degradation of about 1.1 dB, compared to the ideal BER performance of the SR-1. The introduction of the pipeline stage into SR-4 reduces the E_b/N_0 degradation to 0.3 dB, which can be observed in the BER performance of (SR-5, 0.84 V, 6.7 ns, 3%). A similar trend is encountered in the TFM-based STDs, where (TFM-4, 0.84 V, 4.1 ns, 3%) exhibits an E_b/N_0 degradation of about 0.5 dB respect to the ideal BER performance of TFM-1. The introduction of the pipeline stage into TFM-5 enhances the BER performance, as shown in the BER plot of (TFM-5, 0.84 V, 4.1 ns, 3%), which exhibits an E_b/N_0 degradation of only about 0.1 dB. However, when the percentage of power supply variation is increased to 5% (TFM-5, 0.84 V, 4.1 ns, 5%) exhibits a E_b/N_0 degradation of about 1.0 dB. Figure 4.17(b) presents the hardware performance of the modified STDs when they operate at V_{DD} = 0.84 V and E_b/N_0 = 3.0 dB. Here, we present hardware performance results for SR-1 operated at V_{DD} = 1.20 V and in the absence of timing errors as benchmark. Note that schemes (SR-4, 0.84 V, 8.0 ns, 3%) and (TFM-5, 0.84 V, 4.1 ns, 5%) are not considered in Figure 4.17(b), owing to their increased BER. Figure 4.17(b) shows that (SR-5, 0.84 V, 6.7 ns, 3%) increases the number of DCs, latency, energy consumption and BERs performance, when compared to (SR-5, 1.20 V, 4.0 ns, 0%). Scheme (TFM-4, 0.84 V, 4.1 ns, 3%) exhibits a throughput similar to that of SR-1 in the absence of timing errors. However, this modified scheme presents an increased latency and BER. By contrast, the average number of DCs, latency, throughput and energy consumption of (TFM-5, 0.84 V, 4.1 ns, 3%) are 0.75, 1.55, 1.28 and 0.12 times those of (SR-1, 1.20 V, 4.0 ns, 0%), respectively.

Owing to the hardware implementation results presented in Table 4.1, as well as in Figures 4.16(b) and 4.17(b) and to the BER performance of Figures 4.16(a) and 4.17(a), we recommend the employment of TFM-5 in the presence of timing errors.

4.6 Chapter Conclusions

In this chapter, we have presented modifications to the state-of-the-art STD of [39], which significantly improve its timing error tolerance. This has been achieved by considering the close relationship between the different trade-offs involved in the hardware implementation of the STD, as listed in Figure 1.1. To elaborate further, the implementation of iterative decoders is typically oriented towards the optimization of a particular design objective. For example, a particular design may focus on achieving a low chip area, to the detriment of all other design objectives. More specifically, only the design constraints and parameters that affect the overall chip area of the design may be considered and optimized during the design process. However, this approach fails to consider other characteristics of the hardware implementation, such as its energy efficiency, latency, throughput, error correction capability or timing error tolerance. Hence, the resultant implementation may not achieve the desired hardware specifications or BER performance. Motivated by this trend, we conceived the design approach of Figure 4.18 for improving the tolerance of STDs to timing errors.

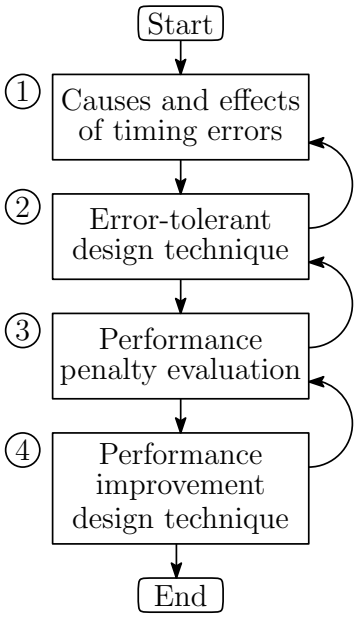


Figure 4.18: Design flow of error-tolerant iterative decoders.

In this case, the design guidelines of Figure 4.18 may be interpreted as follows.

1) As described in Section 4.3 and observed in Figure 4.15, power supply variations were identified as the most detrimental cause of both timing errors and metastability in the STD. Hence we have to conceive measures to mitigate the catastrophic propagation of metastability through the decoder.

2) The employment of synchronizers is recommended as the error-tolerant design technique for mitigating the catastrophic propagation of metastability, as detailed in Section 4.3.1.

3a) However, as portrayed in Table 4.1 and Figure 4.12, the employment of synchronizers increased the chip area, the latency as well as the energy consumption and reduced the throughput.

4a) Therefore, an additional set of EMs may be advocated for enabling the simultaneous decoding of two received frames for the sake of improving the throughput of the STD, as described in Section 4.3.2.

3b) However, observe in Table 4.1 and Figure 4.12 in Section 4.4 that the additional set of EMs increased the chip area, the latency and the energy consumption of the design.

4b) Therefore clock gating is proposed in Section 4.3.2 for reducing the chip area, latency and energy consumption of the STD.

4c) Additionally, in Section 4.3.3, TFM-based EMs were proposed for reducing the chip area, the latency as well as the energy consumption of the design and for improving both the throughput as well as the BER performance, as portrayed in Table 4.1 and Figure 4.12.

4d) Finally, in Section 4.3.4, pipelining was recommended for further improving the BER performance of the STD in the presence of timing errors, as observed in Figures 4.16 and 4.17.

Building on this, we have characterized the trade-offs among the chip area, energy efficiency, latency, throughput and error correction capabilities of different timing-error tolerant STDs, when they operate at two different nominal supply voltages. Our simulations in Figures 4.16 and 4.17 show that the proposed STD (TFM-5, 1.20 V, 2.2 ns, 7%) offers the same BER performance as the state-of-the-art STD (SR-1, 1.20 V, 4.0 ns, 0%) design of [39], despite suffering from power supply variations, while increasing the throughput by a factor of 2.42, reducing the latency by a factor of 0.83 and consuming only 0.25 times the energy of that of (SR-1, 1.20 V, 4.0 ns), without increasing the chip area. Furthermore, this trade-off analysis technique may be applied to the design of other timing-error-tolerant iterative decoder implementations in order to determine the most desirable configuration.

Chapter 5

Reduced-Latency Stochastic Turbo Decoders

The different Stochastic Turbo Decoder (STD) implementations presented in Chapter 4 require thousands of Decoding Cycles (DCs) for successfully decoding a frame, hence resulting in relatively poor processing latencies, throughputs and energy efficiencies. Owing to these impediments, stochastic decoders have been deemed unsuitable for practical low-latency next-generation Mission-Critical Machine-Type Communication (MCMTC) systems [43]. To elaborate further, MCMTC systems require reliable machine-human and machine-machine communication in order to achieve real-time control of dynamic processes, such as those required by industrial process automation and manufacturing, energy distribution, remote surgery and vehicular traffic safety, for example. In these applications, short emergency and control messages constituted by a low number of bits must be reliably transmitted with ultra-low latency. Moreover, MCMTC systems are expected to rely on a large number of energy-constrained wireless sensors, actuators and programmable controllers. As a result, the processing energy consumption must be minimized. In the context of iterative decoders, this may be achieved by reducing the number of iterations of the decoding algorithm, albeit at the cost of degrading the error correction capabilities of the implementation. These scenarios motivate the employment of short-frame-lengths error correction decoders having ultra-low processing latencies on the order of microseconds [43].

Substantial research efforts have been made in order to reduce the number of DCs required by STDs for achieving iterative decoding convergence. As an example of this, the authors of [39] proposed exponential transformations [86] for the implementation of

This chapter is partially based on the following publications.

I. Perez-Andrade, S. Zhong, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Stochastic computing improves the timing-error tolerance and latency of turbo decoders: Design guidelines and trade-offs," in *IEEE Access,* vol. 4, pp. 1008–1038, Feb 2016.

STDs. This technique uses stochastic computing to perform the Taylor's expansion of the exponential function of Bernoulli Sequences (BSs). In this way, the additions of BSs are transformed into multiplications of exponentially transformed BSs, which can be performed with the aid of AND gates, as detailed in Section 2.3.1. However, the result of the multiplication of the exponentially transformed BSs must be converted back into the conventional BS representation, by employing a logarithmic transformation. The order of the Taylor's series determines not only the accuracy of the exponential transformation, but also the hardware complexity of its implementation, as detailed in [79]. As an example of this, a NOT gate may be employed for the first order approximation of both exponential and logarithmic functions. By contrast, the second order approximation requires one D-type Flip Flop (DFF) and two NAND gates for the exponential function, as well as one DFF, two AND gates and a 2-input Multiplexer (MUX) for the logarithmic function. The error correction capability of the STD is improved by using higher order approximations, but the hardware complexity grows rapidly. Motivated by this, the authors of [39] applied the second order exponential transformation to a 200-bit STD. This technique reduced the maximum number of DCs from 250×10^3 to 32×10^3, without degrading the error correction performance of the STD, albeit at the cost of increasing the hardware complexity of the design. In order to further reduce the number of DCs, the STD of [40] employed the multiple-stream decoding technique, which was originally introduced for the stochastic decoding of cortex codes in [37]. This technique increases the degree of parallelism for the STD by representing each probability with $\rho \geq 2$ BSs. The exponential transform-based multiple-stream decoding of the 200-bit STDs in conjunction with $\rho = 32$ allowed the reduction of the number of DCs from 32×10^3 to 1×10^3*, albeit at the cost of increasing the hardware complexity by a factor of $\rho = 32$. Hence, this technique may be deemed unsuitable for practical STD implementations.

The authors of [41, 42] proposed modified BS representations for the implementation of fully-parallel Logarithmic BCJR (Log-BCJR) decoders. More specifically, these contributions proposed sign-magnitude BSs for representing Logarithmic Likelihood Ratios (LLRs). Here, each LLR is represented using one BS for determining its sign and one or more BSs for determining its magnitude. As an example of this, [41] employs 2-bit BS-based LLRs in the range of $[-1, +1]$. In [42], the authors proposed a sliding-window method for converting BSs into bit-serial sign-magnitude LLRs. In each DC, the three most-recent bits of a BS are combined to provide an LLR comprising one sign bit and two magnitude bits. Therefore, each LLR is represented in the range of $[-3, +3]$. Additionally, this implementation relies on 3-bit Fixed-Point (FX) adders and 4-bit FX comparators for the addition and max operations of the sign-magnitude BSs, respectively. These techniques significantly reduce the number of DCs required for successfully decoding a frame and hence yield substantial throughput gains. However, these

*The number of DCs is reported as 1×10^3 in [40]. However, in the more detailed description of the multiple-stream STD given in the PhD thesis of [79], the same architecture and implementation is reported to employ 8×10^3 DCs.

designs may be considered to be half-stochastic STDs or low-precision serially-operated FX Log-BCJR decoders, rather than true-stochastic decoders. Owing to its reliance on FX numbers, the half-stochastic STD of [42] does not benefit from the inherent tolerance of true STDs to timing errors.

Against this background, in this chapter we propose a Reduced-Latency STD (RLSTD) design, which reduces the number of clock cycles required for achieving near-optimal error correction performance by an order of magnitude, without increasing the chip area. This is achieved by employing:

1) OR *gates for performing approximate stochastic additions.*

2) A reduced-complexity Tracking Forecast Memory (TFM) design for overcoming the latching problem.

3) A single DFF for estimating each decoded bit.

Moreover, we analyze the different trade-offs presented in Figure 1.1 for both of the improved STD designs.

The rest of this chapter is structured as follows. Section 5.1 details how to reduce the number of clock cycles required by the STD. Section 5.2 quantifies its error correction performance and Section 5.3 characterizes its hardware efficiency.

5.1 Proposed Reduced-Latency Stochastic Turbo Decoder

The following sections detail the proposed modifications to the STD of [39], which significantly reduce the number of DCs required for achieving iterative decoding convergence. Each of these enhancements is detailed in the following subsections.

5.1.1 Approximate stochastic adders

As described in Section 4.1, the turbo decoding algorithm requires the addition of probabilities, as shown in Equations (4.3) and (4.6). In stochastic computing, BSs can only represent probabilities in the range of [0,1]. However, the addition of M probabilities may not give a result falling in the closed interval [0,1]. To overcome this problem, the addition of M probabilities may be performed by scaling the operands by a factor of M, so that we have $P_{add} = \sum_{i=1}^{M}[P_i/M]$, which represents the mean of the M probabilities. In stochastic computing, this may be performed using an M-input MUX, as described in Section 2.3.1. In this configuration, the MUX outputs the value of one of the M input BSs, which is randomly selected in each DC. However, this means that the other $(M-1)$ BSs do not directly contribute to the output BS of the MUX. As a result of

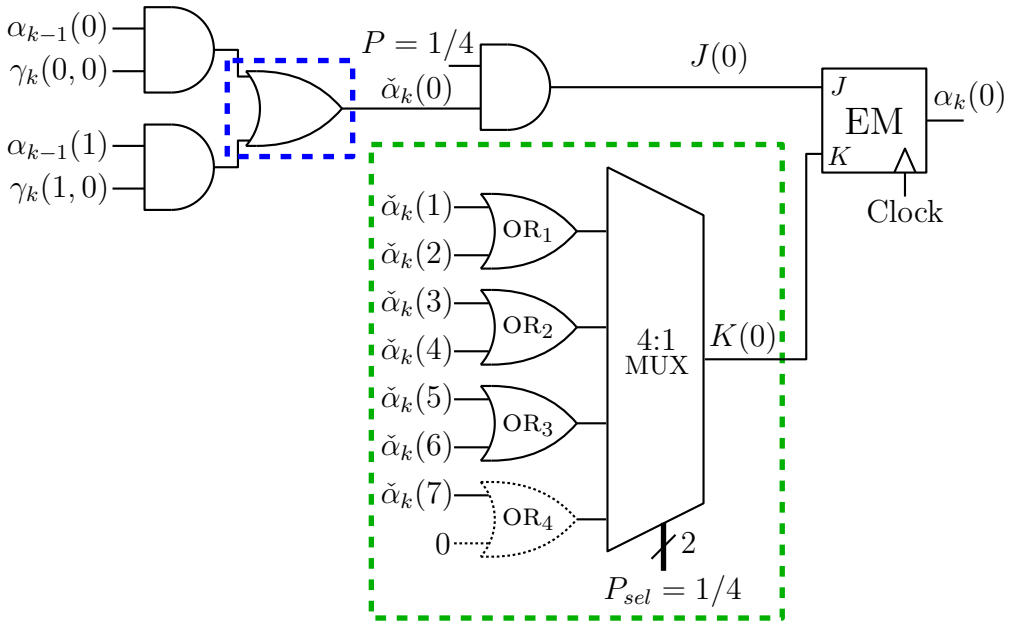


Figure 5.1: Stochastic realization of $\alpha_k(0)$ employing OR gates as approximate adders.

this, the length of the outgoing BS P_{add} is required to be M times longer than that of the input BSs, in order for P_{add} to achieve the same precision as the input BSs [40].

As an alternative to the employment of MUXs as stochastic adders, OR gates may be employed for performing approximate additions, as detailed in [18]. This has the advantage of granting all M input BSs influence over the output BS, hence reducing the length required for the output to achieve the same precision as the inputs. However, the probability P_{OR} represented by the BS output by a two-input OR gate is given by $P_{\text{OR}} = P_A + P_B - P_A \cdot P_B$, where P_A and P_B correspond to the probabilities represented by the two input BSs S_A and S_B, respectively. Therefore, the addition P_{OR} exhibits an error that is proportional to the product $P_A \cdot P_B$, although this will become negligible, if either P_A or P_B has a very small value.

Figure 5.1 shows how OR gates may be employed to perform approximate additions in an STD. More specifically, Figure 4.4 exemplifies the stochastic implementation of the forward recursion $\alpha_k(s)$ of Equation 4.3 for the case where $s = 0$, as it was previously shown in Figure 4.4. The blue box of Figure 5.1 at the top left corner shows the employment of a 2-input OR gate to approximate the two-term addition $\breve{\alpha}_k(0) = [\alpha_{k-1}(0)\gamma_k(0,0) + \alpha_{k-1}(1)\gamma_k(1,0)]$. Similarly, the green box at the bottom right corner of Figure 5.1 shows the employment of OR gates for the approximate addition of 8 BSs. Note that an M-input OR gate outputs the value of 1 if any of its M inputs adopts this value. As a result of this, the employment of an 8-input OR gate required for providing $K(0)$ in Figure 5.1 may result in this BS becoming stuck at 1. Owing to this, the green box of Figure 5.1 implements the addition $K(0) = [\sum_{i=1}^{7}[\breve{\alpha}_k(i)] + 0]/4$ using four 2-input OR gates and a single 4:1 MUX. The division by 4 in $K(0)$

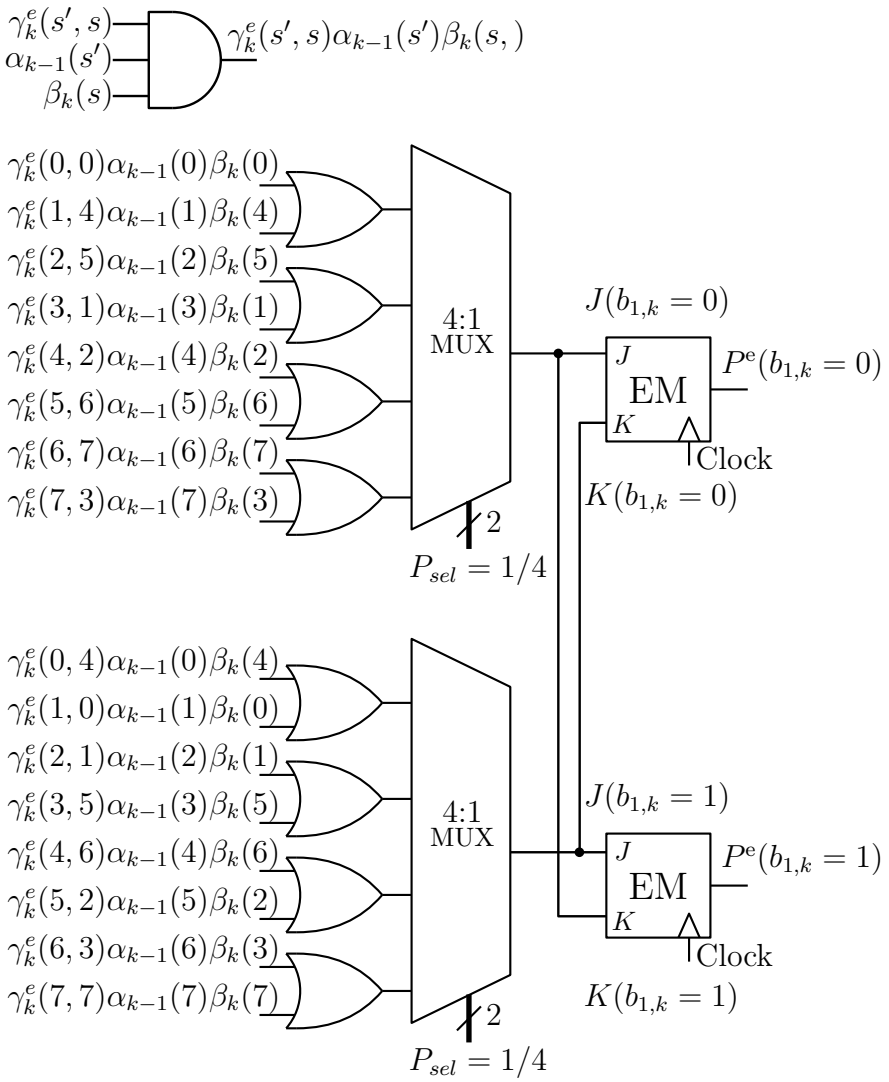


Figure 5.2: Estimation of the *extrinsic* probabilities in the RLSTD.

requires the corresponding division by 4 to be employed in $J(0) = \breve{\alpha}_k(0)/4$, in order to preserve $\alpha_k(0) = J(0)/[J(0) + K(0)]$. This may be achieved using an AND gate for multiplying $\breve{\alpha}_k(0)$ with a BS representing the probability of 1/4, as shown in Figure 5.1. Similar structures to that of Figure 5.1 may be employed for the implementation of the forward recursion $\alpha_k(s)$ for all other states $s \in [1, 7]$, as well as for the backward recursion $\beta_{k-1}(s')$ of Equation 4.4. Note that the OR gate labeled as OR_4 in Figure 5.1 is drawn with dotted lines for indicating that this gate may be eliminated, since one of its inputs has the constant logical value of 0. However, this OR gate is indeed required for the approximate additions that may be used for the calculation of the *extrinsic* probabilities of Equation 4.5 and for the calculation of the A Posteriori Probability (APP) of Equation 4.6, as shown in Figures 5.2 and 5.3, respectively.

As we will demonstrate in Section 5.2, the employment of the approximate adders imposes only an imperceptible Bit Error Ratio (BER) performance degradation on the

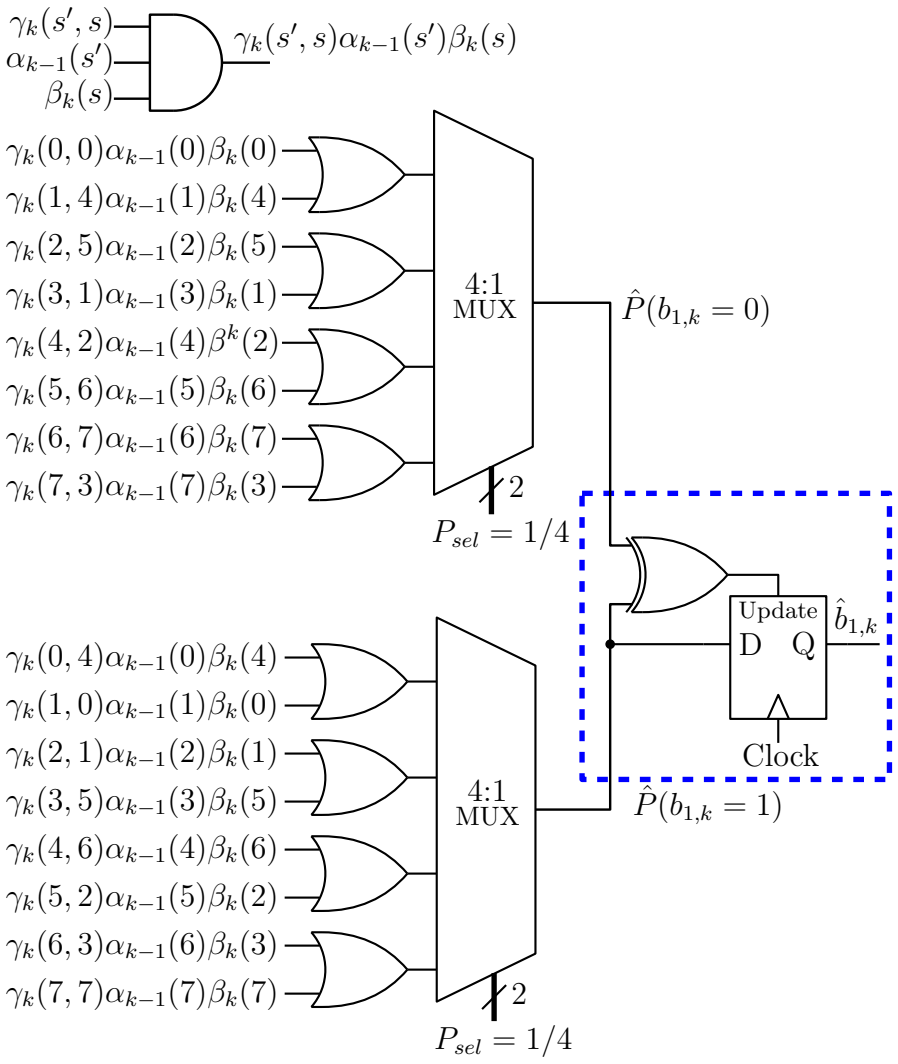


Figure 5.3: Estimation of the hard-decision bit $\hat{b}_{1,k}$ in the RLSTD.

RLSTD. By contrast, the approximate stochastic adders reduce the chip area, the average number of DCs required, the latency and the energy consumption, while increasing the throughput of the RLSTD.

5.1.2 Reduced-complexity tracking-forecast memory

As detailed in Section 4.4, TFM-based Edge Memories (EMs) offer significant improvements in both the BER and the hardware implementation performance of STDs. This may be attributed to the TFM's enhanced capability for tracking changes in the regenerative bit's probability and to the relatively low hardware complexity of TFMs. As described in Section 4.3.3, TFMs employ Equation 4.13 for quantifying the moving average probability $P(t+1)$ of the regenerative bit $J(t)$ having the value of 1 according to

$$P(t+1) = \begin{cases} P(t) - \phi P(t) & J(t) = 0 \\ P(t) + \phi \bar{P}(t) & J(t) = 1. \end{cases}$$

The relaxation parameter ϕ determines the significance given to the regenerative bit and its value can be chosen by obtaining a compelling trade-off between the BER and hardware efficiency, where we use $\phi = 2^{-4}$ in the TFM-based STDs presented in Chapter 4. However, in this section, we propose the use of $\phi = 2^{-1}$, since this facilitates a significant further reduction in the hardware implementation complexity of TFMs. In this case, the probability of the TFM can be expressed as

$$P(t+1) = \begin{cases} P(t) - \frac{P(t)}{2} & = \frac{P(t)}{2} & J(t) = 0 \\ P(t) + \frac{\bar{P}(t)}{2} & = P(t) + \frac{1-P(t)}{2} & \\ & = \frac{P(t)}{2} + \frac{1}{2} & J(t) = 1. \end{cases} \tag{5.1}$$

We refer to the resultant scheme as Reduced-Complexity Tracking Forecast Memory (RCTFM), which may be implemented using the arrangement shown in Figure 5.4(b). Here, the FX TFM represents $P(t)$ using a 9-bit unsigned FX number, where the Most Significant Bit (MSB) has a significance of 2^{-1} and the Least Significant Bit (LSB) has a significance of 2^{-9}. The multiplication $P(t)/2$ can be realized by shifting the contents of the FX TFM one position to the right. Moreover, the conditional addition of the constant $1/2$ can be realized by updating its MSB with the incoming regenerative bit $J(t)$.

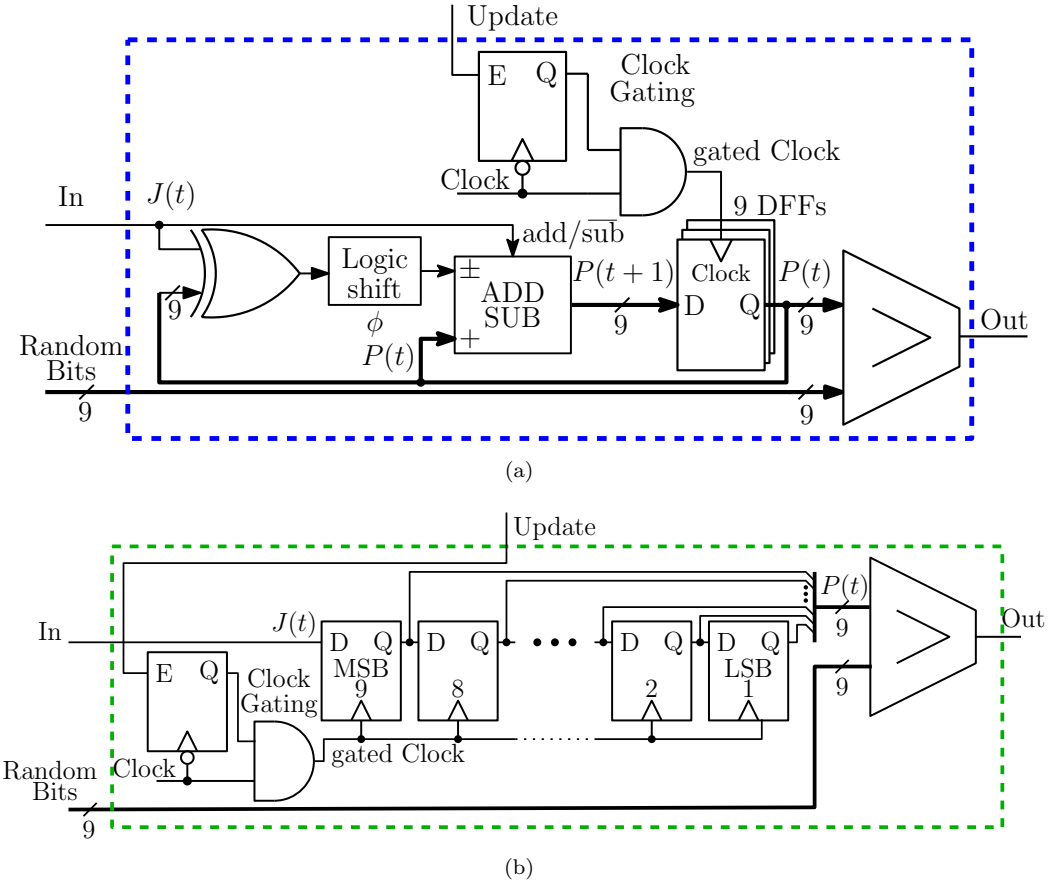


Figure 5.4: Comparison of TFM structures: (a) Conventional TFM employed in TFM-based STDs of Chapter 4. (b) Proposed RCTFM associated with $\phi = 2^{-1}$.

Figure 5.4 compares the hardware complexity of the proposed RCTFM to those of the TFMs employed in schemes TFM-1 to TFM-5 of Chapter 4. In particular, the hardware implementation complexity of the RCTFM is reduced by avoiding the employment of the FX adder/subtractor and the set of XOR gates employed by the TFM shown in Figure 5.4(a).

The operation of the RCTFM-based EM follows the same principles as the Shift Register (SR)-based and TFM-based EMs described in Sections 2.3.2 and 4.3.3, respectively. Moreover, the RCTFM operates as a combination of an SR and a TFM. To elaborate further, the RCTFM adopts the functionality of an SR, when $J \neq K$. In this situation, the DFFs of the RCTFM will shift their contents one position and the incoming regenerative bit $J(t)$ will be stored in the first DFF, in analogy to the behavior of an SR-based EM. Here, the clock-gating latch and the AND gate are employed to enable the clock signal of the DFFs only when the signal Update is asserted, owing to $J \neq K$. By contrast, if $J = K$, the unsigned FX probability $P(t)$ stored in the RCTFM is compared to a pseudo-random number, in order to determine the outgoing bit of the RCTFM, in analogy to the operation of a TFM-based EM. Here, the outgoing bit assumes the value of 1 if the probability stored in the RCTFM is larger than or equal to the 9-bit pseudo-random number. The contents of the RCTFM-based EM can be initialized prior to the beginning of the decoding process, in order to improve the attainable BER performance of the STD. In our investigations detailed in the following sections, the RCTFMs are initialized to store the probability $P = 0.5$, which can be achieved in a single clock cycle by setting the MSB of each RCTFM to logic 1 and the rest of the bits to logic 0.

5.1.3 Output decision

The STDs presented in Chapter 4 employs a 4-bit saturated up/down counter for the estimation of the decoded bit $\hat{b}_{1,k}$. The saturated counter stores a binary value that behaves similarly to a FX representation of an LLR [29], with its MSB determining the hard-decision bit $\hat{b}_{1,k}$, as described in Section 4.2.4. The width of the saturated counter determines both its capability to track changes in the represented LLR and the precision of its representation of this LLR. In this way, a wider counter provides a higher precision for the represented LLR, as well as a more robust mechanism against rapid variations in the value of this LLR. This is particularly useful at low channel Signal to Noise Ratio (SNR) values, where the LLR represented may be expected to fluctuate between consecutive DCs, owing to the low reliability of the decoding process in this SNR region. By contrast, smaller counter widths offer a lower precision for the represented LLR and are more susceptible to changes within the BSs. Despite this, low-precision counters may be employed at high SNR values, owing to the high reliability of the decoding process, as detailed in [29]. In spite of these trade-offs, we propose the employment of only a single DFF for determining the hard-decision bit $\hat{b}_{1,k}$ of the RLSTD, as shown in Figure 5.3.

Here, the output DFF takes the value of 0 if $\hat{P}(b_{1,k} = 0) = 1$ and $\hat{P}(b_{1,k} = 1) = 0$. Similarly, the output DFF takes the value of 1 if $\hat{P}(b_{1,k} = 0) = 0$ and $\hat{P}(b_{1,k} = 1) = 1$. This is achieved by updating the output of the DFF with the bit of the BS representing $\hat{P}(b_{1,k} = 1)$, when $\hat{P}(b_{1,k} = 0) \neq \hat{P}(b_{1,k} = 1)$, as shown in the blue box of Figure 5.3. By contrast, the DFF will not update its contents, if $\hat{P}(b_{1,k} = 0) = \hat{P}(b_{1,k} = 1)$.

Section 5.2 will demonstrate that the error correction capability of the RLSTD is not degraded by having only a single output DFF. Moreover, Section 5.3 will demonstrate that the single output DFF reduces the hardware complexity of the STD.

5.2 Error Correction Capabilities of the Reduced-Latency Stochastic Turbo Decoder

In this section, we characterize the error correction capability of the proposed RLSTD of Section 5.1 and compare it to that of various benchmarks. Figure 5.5 presents the BER performance achieved for different STDs employing 50-bit frames, a coding rate of 1/3, 8 states, tailbiting, S-Random interleavers, the state-transition diagram of Figure 4.1(c) and a Noise-Dependent Scaling (NDS) associated with $\eta = 1$ and $\psi = 2$, as presented in Chapter 4. Moreover, owing to the reduced the complexity of the RLSTD, Figure 5.5 presents BER plots when employing 200-bit frames, in order to allow a direct comparison with the results of [39, 42]. All results assume Binary Phase Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel. Figure 5.5 presents BER plots for the RLSTD described in Section 5.1, when allowing a maximum of 10^4 DCs and when employing early stopping of the decoder's iterations upon achieving convergence. We also present BER plots for four benchmarks, namely the Floating-Point (FP) Log-BCJR and Max-Log-BCJR turbo decoders, when allowing a maximum of 8 iterations, as well as for the schemes SR-1 and TFM-1 described in Chapter 4, when allowing a maximum of 10^5 DCs. Figure 5.5 demonstrates that the RLSTD exhibits a similar BER performance to that of the TFM-1 scheme, which confirms that the proposed modifications do not degrade the attainable error correction performance of the RLSTD. More specifically, the 50-bit RLSTD exhibits near-optimal decoding performance, when compared to the FP Log-BCJR decoder. However, the 200-bit RLSTD exhibits an E_b/N_0 degradation of up to 0.2 dB and 0.5 dB, when compared to the sub-optimal Max-Log-BCJR and to the Log-BCJR turbo decoders, respectively. Note however that similar trends may be observed for the 200-bit TFM-1 STD described in Section 4.3.

Figure 5.6 presents the average number of DCs required by the 50-bit and the 200-bit STDs for successfully decoding a frame, when operated at different E_b/N_0 values and when compared to the benchmarks of SR-1 and TFM-1 presented in Chapter 4. Figure 5.6 demonstrates that the proposed RLSTD reduces the average number of DCs

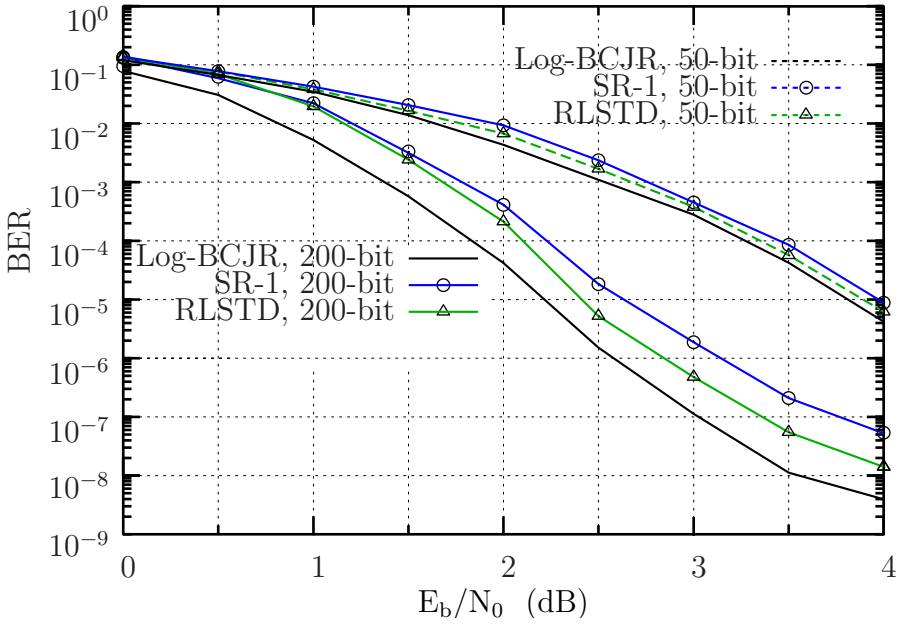


Figure 5.5: BER performance of the RLSTD, as well as of various benchmarks.

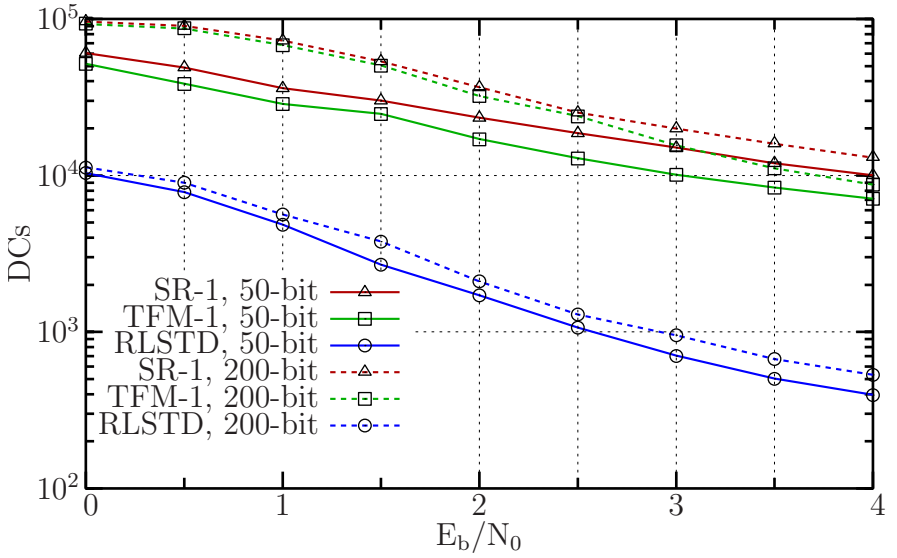


Figure 5.6: Average number of DCs for successfully decoding a frame for the RLSTD.

by an order of magnitude, when compared to both the SR-1 and to the TFM-1 schemes. This may be attributed to the increased switching activity owing to the employment of OR gates as approximate adders and to the employment of a single DFF for providing the decision of the decoded bit. Figures 5.5 and 5.6 demonstrate that the proposed RLSTD significantly reduces the number of DCs required for successfully decoding a frame, without degrading its error correction capability. As a benefit of this, the latency, the throughput and the energy efficiency of the STD are significantly enhanced, as we will demonstrate in Section 5.3.

5.3 Hardware Implementation of the Reduced-Latency Stochastic Turbo Decoder

Table 5.1: Hardware complexity comparison of the SR-1, TFM-1 and RLSTD schemes.

Module	Scheme	AND2	OR2	XOR2	MUX2	MUX4	MUX8	DFFs
γ	SR-1	8	0	0	0	0	0	0
	TFM-1	8	0	0	0	0	0	0
	RLSTD	8	0	0	0	0	0	0
α	SR-1	40	8	16	280	0	8	264
	TFM-1	40	8	16	24	0	8	73
	RLSTD	32	40	16	24	8	0	73
β	SR-1	40	8	16	280	0	8	264
	TFM-1	40	8	16	24	0	8	73
	RLSTD	32	40	16	24	8	0	73
Ext	SR-1	32	2	16	68	0	2	66
	TFM-1	32	2	16	2	0	2	20
	RLSTD	32	10	16	2	2	0	20
δ	SR-1	32	0	0	0	0	2	4
	TFM-1	32	0	0	0	0	2	4
	RLSTD	32	8	1	0	2	0	1
TOTAL	SR-1	**152**	**18**	**48**	**628**	**0**	**20**	**598**
	TFM-1	**152**	**18**	**48**	**50**	**0**	**20**	**170**
	RLSTD	**136**	**98**	**49**	**50**	**20**	**0**	**167**

Table 5.1 presents the hardware complexity of the RLSTD in terms of the number of basic logic gates and DFFs employed per decoded bit, when compared to the hardware complexity of the SR-1 and TFM-1 schemes of Table 4.1 used as benchmarks. Table 5.1 demonstrates that the number of 2-input MUX gates employed in the TFM-1 and RLSTD schemes is significantly reduced, when compared to those of SR-1 described in Section 4.2. This may be attributed to SR-1 employing 32 MUXs for the update operation of each of the 18 SR-based EMs and to the employment of 2-input MUXs for the addition of BSs. Similarly, the number of DFFs employed in the TFM-1 scheme detailed in Section 4.3 and in the RLSTD scheme described in Section 5.1 is significantly reduced, owing to the employment of 9-bit TFMs. By contrast, the RLSTD scheme employs a higher number of 2-input OR gates, when compared to both SR-1 and TFM-1, owing to the employment of these logic gates for the approximate addition of BSs. However, Table 5.2 demonstrates that the increased number of OR gates does not increase the chip area or the overall gate count of the RLSTD.

Table 5.2 compares the hardware efficiency of different STDs, when using Taiwan Semiconductor (TSMC) 90 nm technology. We present results for the rate 1/3, 8-state, tailbiting RLSTD, TFM-1 and SR-1 schemes, using S-Random interleavers and the state-transition diagram of Figure 4.1(c), in order to allow direct comparison with the

results of Table 4.1 in Section 4.4. Table 5.2 characterizes the various STDs in terms of their diverse characteristics, including the chip area per decoded bit, clock period, number of equivalent NAND gates, number of DCs, latency, throughput, as well as area and energy efficiency, when using TSMC 90 nm technology. Additionally, we employ NDS associated with $\eta = 1$ and $\psi = 2$ for the proposed RLSTD as well as for the SR-1 and TFM-1 schemes, as described in Section 2.3.2. As described in Section 4.4, the results of Table 5.2 were obtained from the physical layout generated by the automatic place and route of the RLSTD, TFM-1 and SR-1 STDs using Cadence SoC Encounter [72]. These results were obtained for the cases, where the supply voltage of the STDs is set to 1.20 V in the absence of power supply variations. Both the critical clock period and the energy consumption are obtained from Synopsys PrimeTime [73]. The average number of DCs, latency, throughput and energy efficiency were obtained from post-layout gate-level simulations, with the extracted parasitics and annotated delays without timing errors, when using early stopping and allowing a maximum of 10^5 DCs for both SR-1 and TFM-1, as described in Section 4.4, as well as a maximum of 10^4 DCs for the proposed RLSTD described in Section 5.1. We assume that the different STDs operate at their critical clock period, for BPSK transmission over an AWGN channel having an SNR, where a BER of 10^{-5} is achieved. We also present the hardware efficiency of the half-stochastic STD, as reported in [42]. Note that the hardware results of [42] correspond to the synthesis of a 2048-bit fully-parallel decoder using TSMC 90 nm technology. However, the authors of [42] did not quantify the area or energy consumption of this half-stochastic STD implementation, hence the corresponding characteristics cannot be shown in Table 5.2.

Table 5.2 demonstrates that the reduced number of DCs offered by the RLSTDs facilitate reduced latencies, increased throughputs as well as improved area and energy efficiencies, when compared to the benchmarks SR-1 and TFM-1. As shown in Table 5.2, the proposed RLSTD has the lowest gate count and area per bit among all the STDs considered. However, the proposed RLSTD presents a larger gate count than that of the half-stochastic decoder of [42]. The proposed RLSTD enables the highest clock frequency and the lowest average number of DCs, when the frame length is 50 bits, resulting in the lowest latency among all schemes considered. This is particularly attractive in MCMTCs, where emergency or control messages comprising as low as tens of bits must be reliably communicated with ultra-low latency.

Our simulations suggest that the number of DCs required by the proposed RLSTD scales approximately linearly with the frame length. We performed BER simulations of the RLSTD having a frame length of 2048 and an SNR of $E_b/N_0 = 1.25$ dB, which we found to result in the desired BER of 10^{-5}. Here, the average number of DCs required by the RLSTD is 6×10^3. Owing to this, the throughput and area efficiency of the RLSTD having a frame length of 2048 bits are significantly lower than those of the half-stochastic decoder of [42], as shown in Table 5.2. However, the half-stochastic decoder employs a fixed number of 280 DCs. Therefore, the latency of this design remains

Table 5.2: Hardware efficiency of different STDs.

	RLSTD			TFM-1		SR-1 [39]		[42]
Algorithm	Stochastic			Stochastic		Stochastic		Half-stoch.
Area per bit (mm^2)	0.0110			0.0129		0.026		-
Gate count per bit	4.4 K			4.6 K		8.6 K		3.4 K
T_{clk} Frequency	1.7 ns 588 MHz			2.2 ns 454 MHz		4.0 ns 250 MHz		1.8 ns 550 MHz
Frame length (bits)	50	200	2048	50	200	50	200	2048
BER @ E_b/N_0	10^{-5} 3.8 dB	10^{-5} 2.6 dB	10^{-5} 1.25 dB	10^{-5} 3.8 dB	10^{-5} 2.55 dB	10^{-5} 3.95 dB	10^{-5} 2.75 dB	10^{-5} 1.25 dB
Average DCs	200	800	6 K	7.5 K	19 K	10 K	20 K	280
Latency (μs)	0.34	1.36	10.2	16.5	41.8	40	80	0.504
Throughput (Mbps)	147	147	200	3.0	4.7	1.2	2.5	4000
Area eff. (bps/gates)	669	167	22	13	5.1	2.8	1.5	574
Area eff. (Mbps/mm^2)	267	66	8.8	4.65	1.82	0.92	0.48	-
Energy eff. (nJ/bit)	0.76	3.04	31.13	41.25	104	240	480	-

constant regardless of the frame length, as opposed to the RLSTD implementation. As a result, for a frame length of 50 bits, the latency of the half-stochastic decoder is $1.8\text{ ns} \times 280 = 0.504\ \mu$s and its throughput is $50/0.504\ \mu\text{s} = 99$ Mbps, which are inferior to those of the proposed RLSTD. Based on these results, we recommend the employment of the proposed RLSTD for short-frame-length, ultra-low-latency applications, such as MCMTCs.

The hardware efficiency of the RLSTD is further detailed in Figure 5.7, which compares the hardware implementation trade-offs associated with the 50-bit and 200-bit versions of the TFM-1 and RLSTD schemes, relative to the benchmark scheme SR-1, for the case where $E_b/N_0 = 3.0$ dB. Figure 5.7(a) shows that the average number of DCs required by the proposed RLSTD scheme is as low as 0.035 times the corresponding number required by SR-1. As a result of this, the latency, throughput and energy consumption of the modified RLSTD are 0.015, 65 and 0.005 times those of the benchmark scheme SR-1, respectively. Similar trends may be observed for the hardware implementation results of the 200-bit STDs, as shown in Figure 5.7(b). Here, the proposed RLSTD exhibits a latency that is just 0.013 times the latency of SR-1, which increases the throughput by 78 times and consumes 0.005 times the energy.

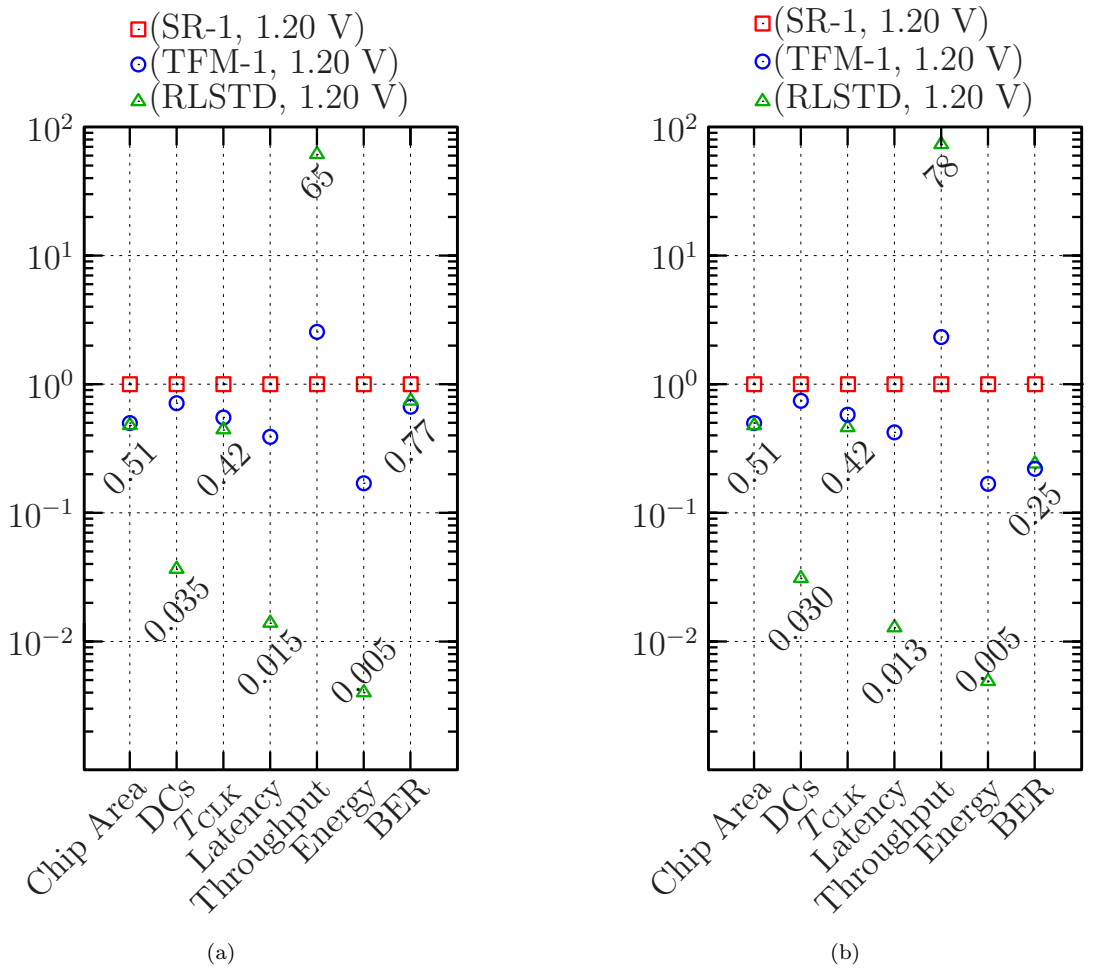


Figure 5.7: Hardware implementation results for different STDs, normalized relative to SR-1, when operated at $V_{DD} = 1.20$ V and when $E_b/N_0 = 3.0$ dB. (a) 50-bit STDs. (b) 200-bit STDs.

5.4 Chapter Conclusions

In this chapter, we have proposed modifications to the state-of-the-art STD that significantly reduce the average number of DCs required for successfully decoding a frame. This has been achieved with the employment of OR gates for the approximate stochastic addition of BSs, as well as a reduced-complexity TFM and a single output DFF for the estimation of the decoded bit. As a result of these modifications, the proposed RLSTD improves the latency, throughput and energy efficiency of the state-of-the-art STD by an order of magnitude, without imposing an area extension and without degrading the error correction capabilities of the STD. Moreover, we have characterized the hardware complexity and the trade-offs between the chip area, latency, throughput, energy efficiency and error correction capabilities of the proposed RLSTD, when compared to the schemes SR-1 and TFM-1 presented in Chapter 4. Our simulations show that the proposed 50-bit RLSTD exhibits an improved BER, requires 0.035 times the number of DCs, 0.015

times the latency and 0.005 times the energy of the state-of-the-art SR-1, while increasing the throughput 65 times and employing only 0.51 times the chip area. Similar trends were found for the proposed 200-bit RLSTD, which offers a throughput that is 78 times the throughput of SR-1. Based on the results presented in Section 5.3, we conclude that the proposed RLSTD is particularly suited for short-frame-length and low-latency communication systems, such as those required in next-generation MCMTCs.

Note that the different STDs presented in this chapter and in Chapter 4 exhibit very low chip area per bit, but the cost of this limits the processing throughput to the order of Mbps. However, next-generation communication standards [10, 11] are expected to require processing throughputs on the order of tens of Gbps. In order to fulfill this high throughput requirement, a Fully-Parallel Turbo Decoder (FPTD) algorithm has been recently proposed in [60]. This FPTD algorithm achieves throughputs in the order of tens of Gbps, albeit at the cost of a large computational complexity. Building on this, Chapter 6 reviews the hardware implementation of the FPTD algorithm of [60]. In addition to this, we present a novel Reduced-Critical-Path Fully-Parallel Turbo Decoder (RCP-FPTD) algorithm, which further enhances the attainable throughput of turbo decoder implementations. Moreover, we present a timing analysis for determining the causes and effects of timing errors in the FPTD and RCP-FPTD implementations. Finally, Chapter 6 details the employment of timing-error-tolerant design techniques in the FPTD and RCP-FPTD implementations for enhancing their BER performance and hardware efficiency in the presence of timing errors.

Chapter 6

Timing-Error-Tolerant Fully-Parallel Turbo Decoders

The various Stochastic Turbo Decoders (STDs) presented in the previous chapters require a large number of Decoding Cycles (DCs) for achieving iterative decoding convergence. As an example of this, the STDs of Chapter 4 typically require thousands of DCs to successfully decode each frame. This limits the attainable throughput of these STDs to only a few Mbps. This limitation is partially overcome by the Reduced-Latency STD (RLSTD) of Chapter 5, which requires only hundreds of DCs, achieving throughputs in the order of hundreds of Mbps, while maintaining low chip area requirements. However, the number of DCs required by the RLSTD scales almost linearly with the frame length of the turbo code. As a result of this, the proposed RLSTD may only be considered suitable for short-frame-length communication systems, such as as those required in the next-generation Mission-Critical Machine-Type Communication (MCMTC) [43].

In contrast to this, the next-generation of wireless communication standards [10, 11] are expected to require processing throughputs on the order of tens of Gbps. Significant efforts have been made along the way to fulfilling these throughput requirements. As an example of this, the state-of-the-art turbo decoder implementations achieve processing throughputs of 2.15 Gbps [57], 1.67 Gbps [58] and 1.28 Gbps [59]. These turbo decoder implementations rely on increasing the parallelism of the highly-serial Logarithmic BCJR (Log-BCJR) algorithm. Despite these efforts, the proposed solutions still require hundreds of clock periods to complete a decoding iteration, resulting in a requirement for thousands of clock periods for completing the iterative decoding process. This may be attributed to the inherently serial nature of the Log-BCJR algorithm. Moreover, the

This chapter is partially based on the following publications.

I. Perez-Andrade, S. Zhong, K. Li, A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Timing-error-tolerant VLSI implementation of fully-parallel turbo decoding," in *IEEE J. Solid-State Circuits,* [In preparation].

implementations of [57, 58, 59] do not consider fault tolerant design. As a result of this, the significant throughput gains offered by these implementations may be diminished by the occurrence of processing errors, if the operating conditions of the system fluctuate below the recommended safety margins.

In order to fulfill the high throughput requirements of next-generation wireless communication standards, a Fully-Parallel Turbo Decoder (FPTD) algorithm has been recently proposed in [60]. This FPTD algorithm facilitates throughputs in the order of tens of Gbps, since it disposes with the highly-serial data dependencies of the Log-BCJR algorithm. Owing to this, each FPTD decoding iteration may be completed in a single clock period, albeit at the cost of a large computational complexity.

Against this background, this chapter reviews the recently-proposed FPTD algorithm of [60]. Moreover, we present a novel Reduced-Critical-Path Fully-Parallel Turbo Decoder (RCP-FPTD) algorithm, which further improves the processing throughput of turbo decoders. In addition to this, we characterize the different trade-offs associated with the hardware implementation of the FPTD and the RCP-FPTD. Furthermore, we investigate the inherent tolerance to timing errors of the FPTD and RCP-FPTD implementations. This is achieved by performing a timing analysis to characterize the causes and effects of timing errors in FPTD and RCP-FPTD implementations. Finally, we employ the Better-Than-Worst-Case (BTWC) design approach [19] and Error Detection and Correction (EDAC) techniques [13, 20, 21, 22] for mitigating the effect of timing errors in the FPTD and RCP-FPTD implementations, which significantly enhance their error correction capabilities and the hardware efficiency in the presence of timing errors.

The rest of this chapter is structured as follows. Section 6.1 reviews the traditional Log-BCJR algorithm in order to allow its comparison with the recently-proposed FPTD algorithm of [60], which is presented in Section 6.2. Moreover, Section 6.2 reviews the hardware implementation requirements of the FPTD Section 6.3 presents the novel RCP-FPTD algorithm and its hardware implementation requirements. Following this, Section 6.4 presents the trade-off analysis of the FPTD and RCP-FPTD implementations, when compared to RLSTD presented in Chapter 5, as well as when compared to the state-of-the-art turbo decoder implementations of [57, 58, 59]. Section 6.5 investigates the inherent tolerance to timing errors of the FPTD and RCP-FPTD implementations. This is achieved by performing a timing analysis for determining the causes and effects of timing errors in the FPTD and RCP-FPTD implementations. Section 6.6 details the employment of BTWC and EDAC design techniques for mitigating the negative effects of timing errors in the FPTD and RCP-FPTD implementations. Furthermore, Section 6.6 characterizes the trade-off between the error correction performance and hardware efficiency of the various timing-error tolerant FPTD and RCP-FPTD implementations. Finally, Section 6.7 summarizes our main findings, while Section 6.8 presents our concluding remarks for this chapter.

6.1 Logarithmic BCJR Algorithm

This section reviews the concept of turbo encoding and turbo decoding, followed by a review of the Log-BCJR algorithm. However, the discussion of Section 4.1 regarding turbo encoding is repeated in this section for the benefit of the reader. A turbo code comprises the parallel concatenation of two convolutional codes separated by an interleaver, as shown in Figure 4.1(a) and repeated in Figure 6.1 for convenience.

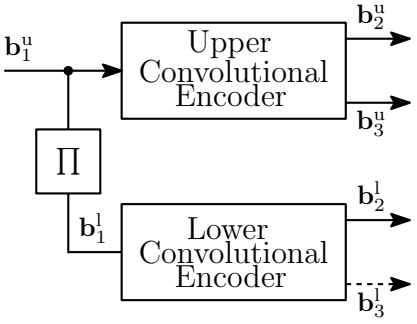


Figure 6.1: Simplified turbo encoder.

A message frame $\mathbf{b}_1^{\mathrm{u}} = [b_{1,k}^{\mathrm{u}}]_{k=1}^{N}$ comprising N bits is provided to the upper convolutional encoder, which outputs a parity frame $\mathbf{b}_2^{\mathrm{u}} = [b_{2,k}^{\mathrm{u}}]_{k=1}^{N}$ and a systematic frame $\mathbf{b}_3^{\mathrm{u}} = [b_{2,k}^{\mathrm{u}}]_{k=1}^{N}$, each comprising N bits. The message frame $\mathbf{b}_1^{\mathrm{u}}$ is also interleaved, in order to obtain the interleaved message frame $\mathbf{b}_1^{\mathrm{l}} = [b_{1,k}^{\mathrm{l}}]_{k=1}^{N}$, which is provided to the lower convolutional encoder so that it can generate the parity frame $\mathbf{b}_2^{\mathrm{l}} = [b_{2,k}^{\mathrm{l}}]_{k=1}^{N}$ and the systematic frame $\mathbf{b}_3^{\mathrm{l}} = [b_{3,k}^{\mathrm{l}}]_{k=1}^{N}$. Note that in a rate 1/3 turbo code, the systematic frame $\mathbf{b}_3^{\mathrm{l}}$ is not transmitted, as represented by the dotted line in Figure 6.1. Here, the superscripts 'u' and 'l' indicate relevance to the upper and lower convolutional encoders, respectively. However, these superscripts are only used throughout the rest of this chapter for explicitly distinguishing between the two convolutional encoders and are omitted when the discussion applies equally to both.

As mentioned in Section 4.1, the turbo decoder operates on the basis of the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [63], which is described by Equations (4.1) to (4.6). However, the BCJR algorithm relies on the multiplication and addition of probabilities in the range [0,1], which can lead to numerical stability issues, when the probabilities adopt very small values. In order to overcome this problem, the BCJR algorithm can be operated in the logarithmic domain. The resulting algorithm is referred to as Log-BCJR [87] and its operation is based on Logarithmic Likelihood Ratios (LLRs), as defined in Equation 3.7 and repeated here for convenience as

$$L_P = \log \frac{P(b=0)}{P(b=1)}.$$

$$\gamma_k(s',s) = b_1(s',s)\cdot b^{\mathrm{a}}_{1,k} + b_2(s',s)\cdot b^{\mathrm{a}}_{2,k} + b_3(s',s)\cdot b^{\mathrm{a}}_{3,k} \tag{6.1}$$

$$\alpha_k(s) = \max^{*}_{\text{all } s'}[\gamma_k(s',s) + \alpha_{k-1}(s')] \tag{6.2}$$

$$\beta_{k-1}(s') = \max^{*}_{\text{all } s}[\gamma_k(s',s) + \beta_k(s)] \tag{6.3}$$

$$\delta_k(s',s) = \gamma_k + \alpha_{k-1}(s') + \beta_k(s) \tag{6.4}$$

$$b^{\mathrm{e}}_{1,k} = \left[\max^{*}_{\{(s',s)|b_1(s',s)=1\}}[\delta_k(s',s)]\right] - \left[\max^{*}_{\{(s',s)|b_1(s',s)=0\}}[\delta_k(s',s)]\right] - b^{\mathrm{a}}_{1,k} - b^{\mathrm{a}}_{3,k} \tag{6.5}$$

$$\hat{b}_{1,k} = (b^{\mathrm{a}}_{1,k} + b^{\mathrm{e}}_{1,k} + b^{\mathrm{a}}_{3,k}) > 0 \tag{6.6}$$

The Log-BCJR operates on the basis of Equations (6.1) to (6.6), which can be obtained by taking the natural logarithm of Equations (4.1) to (4.6). Here, Equations (6.2), (6.3) and (6.5) employ the Jacobian logarithm, which is defined as

$$\max^{*}(a,b) = \max(a,b) + \log[1+\exp^{-|a+b|}], \tag{6.7}$$

for the case of two operands and may be extended to more operands by exploiting its associative property, where $\max^{*}(a,b,c) = \max^{*}(\max^{*}(a,b),c)$, for example. The complexity of the Log-BCJR algorithm can be reduced by using the approximation

$$\max^{*}(a,b) \approx \max(a,b), \tag{6.8}$$

at the cost of slightly degrading the Bit Error Ratio (BER) performance of the resultant Max-Log-BCJR algorithm, as detailed in [87].

Figure 6.2 illustrates the implementation of the Log-BCJR algorithm employing Equations (6.1) to (6.6), for the case of a turbo decoder having a frame length of N-bits. After their transmission over a wireless channel, the received frames of soft-valued LLRs $\mathbf{b}^{\mathrm{u,a}}_2 = [b^{\mathrm{u,a}}_{2,k}]^N_{k=1}$, $\mathbf{b}^{\mathrm{u,a}}_3 = [b^{\mathrm{u,a}}_{3,k}]^N_{k=1}$, $\mathbf{b}^{\mathrm{l,a}}_2 = [b^{\mathrm{l,a}}_{2,k}]^N_{k=1}$ and $\mathbf{b}^{\mathrm{l,a}}_3 = [b^{\mathrm{l,a}}_{3,k}]^N_{k=1}$ are provided to the turbo decoder, which is comprised of the upper and lower convolutional decoders separated by an interleaver, as shown in Figure 6.2. Note that the frame of lower systematic LLRs $\mathbf{b}^{\mathrm{l,a}}_3$ may be obtained by interleaving the upper systematic LLRs $\mathbf{b}^{\mathrm{u,a}}_3$, since $\mathbf{b}^{\mathrm{l}}_3$ is typically not transmitted in turbo codes having coding rate of 1/3.

In each iteration of the Log-BCJR algorithm, each convolutional decoder performs a forward recursion for calculating the state metrics $\alpha_k(s)$ of Equation 6.2, followed by a backward recursion for calculating $\beta_{k-1}(s')$ of Equation 6.3. More specifically, each iteration of the Log-BCJR algorithm comprises the following steps. The k^{th} block of the first row of the upper convolutional decoder combines the three *a priori* LLRs $b^{\mathrm{u,a}}_{1,k}$, $b^{\mathrm{u,a}}_{2,k}$ and $b^{\mathrm{u,a}}_{3,k}$, in order to obtain an *a priori* branch metric $\gamma^{\mathrm{u}}_k(s',s)$ for each transition of Figure 4.1(c) from a previous state s' into the next state s. Following this, the k^{th}

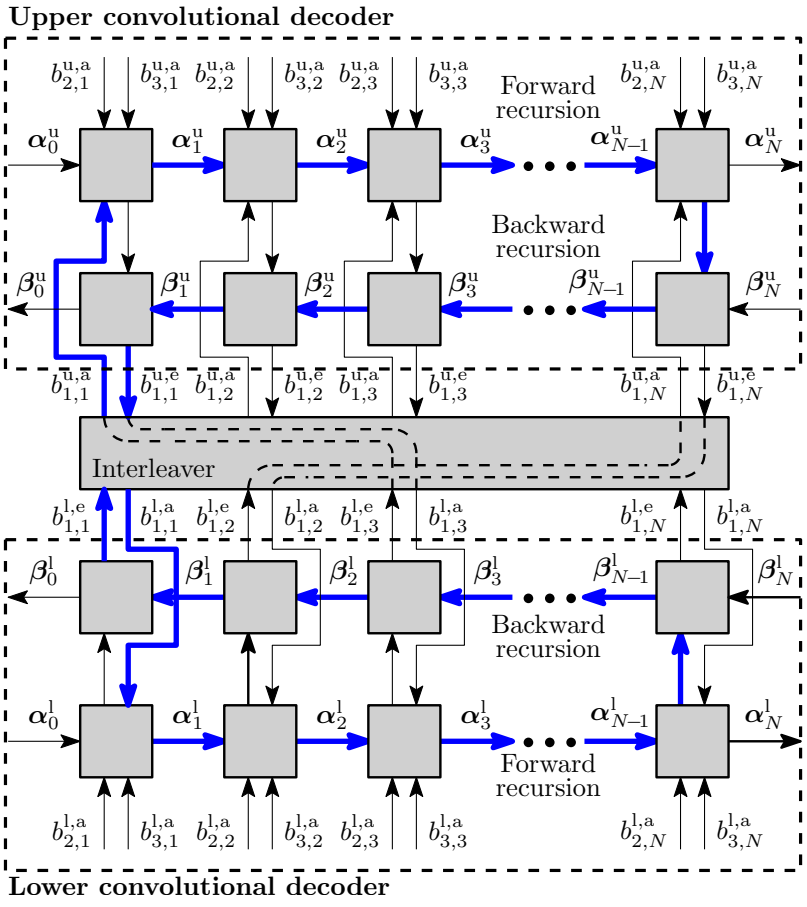


Figure 6.2: Data dependencies of the LogBCJR algorithm [60, Fig. 1(c)].

block of the first row of the upper convolutional decoder employs Equation 6.2 during the forward recursion for providing the vector of state metrics $\boldsymbol{\alpha}_k^{\mathrm{u}} = [\alpha_k^{\mathrm{u}}(s)]_{s=0}^{7}$. Note that the calculation of the state metrics $\boldsymbol{\alpha}_k^{\mathrm{u}}$ is a serial process, where the k^{th} block of the forward recursion may be operated only after the preceding $(k-1)^{th}$ block. The forward recursion continues until all N blocks have been processed in order. After this, the N blocks are processed in reverse order during the backward recursion, where the k^{th} block of the second row of the upper convolutional decoder employs Equation 6.3 for providing the state metrics $\boldsymbol{\beta}_{k-1}^{\mathrm{u}} = [\beta_{k-1}^{\mathrm{u}}(s')]_{s'=0}^{7}$. In analogy to the operation of the forward recursion, the backward recursion operates serially, where the k^{th} block may be operated only after the block $(k+1)^{th}$ block. Additionally, the k^{th} block of the second row of the upper convolutional decoder employs Equation 6.4 for proving the *a posteriori* branch metrics $\delta_k(s', s)$. Furthermore, the k^{th} block in the second row of the upper convolutional decoder employs Equation 6.5 for calculating the *extrinsic* LLR $b_{1,k}^{\mathrm{u,e}}$. Following this, the frame of *extrinsic* LLRs $\mathbf{b}_1^{\mathrm{u,e}} = [b_{1,k}^{\mathrm{u,e}}]_{k=1}^{N}$ is interleaved and passed to the lower convolutional decoder as the frame of *a priori* LLRs $\mathbf{b}_1^{\mathrm{u,a}} = [b_{1,k}^{\mathrm{u,a}}]_{k=1}^{N}$. In analogy to the operation of the upper decoder, the lower convolutional decoder employs Equations (6.1) to (6.5) to combine the received LLRs $\mathbf{b}_2^{\mathrm{l,a}}$ and $\mathbf{b}_3^{\mathrm{l,a}}$ and the *a priori* LLRs $\mathbf{b}_1^{\mathrm{l,a}}$ provided by the upper convolutional decoder, in order to provide the *extrinsic* LLRs

$\mathbf{b}_1^{\mathrm{l,e}} = [b_{1,k}^{\mathrm{l,e}}]_{k=1}^{N}$, which are de-interleaved and passed to the upper convolutional decoder as the frame of *a priori* LLRs $\mathbf{b}_1^{\mathrm{u,a}}$. Finally, each block of the upper convolutional decoder combines the *a priori* LLR $b_{1,k}^{\mathrm{u,a}}$ provided by the lower convolutional decoder with the *extrinsic* LLR $b_{1,k}^{\mathrm{u,e}}$ and the received systematic LLR $b_{3,k}^{\mathrm{u,a}}$ using Equation 6.6, in order to estimate the decoded bit $\hat{b}_{1,k}$. This iterative process is repeated until an accurate estimation of the decoded frame can be obtained, or until the maximum affordable number of iterations has been reached.

Each iteration of the Log-BCJR algorithm comprises the sequential operation of the $4N$ blocks in the order indicated in the blue arrows of Figure 6.2. Owing to this, one iteration of the Log-BCJR algorithm is completed in $T = 4N$ clock periods. As a result of this, Log-BCJR turbo decoders may exhibit a large processing latency, particularly when decoded large frame lengths. In order to overcome this problem, various techniques have been proposed for increasing the parallelism and reducing the number of clock periods required by the Log-BCJR algorithm. As an example of this, the authors of [57] proposed the employment of the Non-Sliding Window (NSW) technique for decomposing the rows of Figure 6.2 into 32 windows, each comprising an equal number of blocks. This technique mitigates the highly-serial data dependencies of the Log-BCJR algorithm by operating all 32 windows simultaneously, with each window's recursions employing the results provided by the adjacent windows in the previous iteration. Additionally, the authors of [57] employed the Radix-4 transform for merging two of the state-transition diagrams of Figure 4.1(c) into one, effectively halving the number of blocks of Figure 6.2, albeit at the cost of doubling the number of *a priori* LLRs employed by each block and more than doubling their computational complexity. By combining the NSW and the Radix-4 transform, the state-of-the-art Long Term Evolution (LTE) [9] turbo decoder of [57] requires $T = N/32$ consecutive clock periods to complete one decoding iteration of the Log-BCJR algorithm. When employing the longest frame length defined in the LTE standard [9], the state-of-the-art turbo decoder of [57] requires $T = 6144/32 = 192$ consecutive clock periods per decoding iteration, which is nearly two orders-of-magnitude higher than that of the novel FPTD algorithm recently proposed in [60]. In the FPTD algorithm of [60], all blocks of the upper and lower convolutional decoders may be operated concurrently, allowing the iterative decoding process to be completed in only tens of consecutive clock periods. The FPTD algorithm of [60] and its hardware implementation is described in the following section.

6.2 Fully-Parallel Turbo Decoder

The following sections describe the FPTD algorithm of [60] and its hardware implementation requirements, for the case of the turbo decoder specified in the LTE standard. In Section 6.2.1, we summarize the FPTD algorithm of [60], while Section 6.2.2 details its hardware implementation requirements.

6.2.1 Algorithm

The FPTD algorithm of [60] operates on the basis of Equations (6.9) to (6.13).

$$\gamma_k^t(s',s) = b_1(s',s)\cdot b_{1,k}^{t-1,\mathrm{a}} + b_2(s',s)\cdot b_{2,k}^{\mathrm{a}} + b_3(s',s)\cdot b_{3,k}^{\mathrm{a}} \tag{6.9}$$

$$\alpha_k^t(s) = \max_{\text{all } s'}{}^{*}[\gamma_k^t(s',s) + \alpha_{k-1}^{t-1}(s')] \tag{6.10}$$

$$\beta_{k-1}^t(s') = \max_{\text{all } s}{}^{*}[\gamma_k^t(s',s) + \beta_k^{t-1}(s)] \tag{6.11}$$

$$\begin{aligned} b_{1,k}^{t,\mathrm{e}} = &\left[\max_{\{(s',s)|b_1(s',s)=1\}}{}^{*}\left[\gamma_k^t(s',s) + \alpha_{k-1}^{t-1}(s') + \beta_k^{t-1}(s)\right]\right] - \\ &\left[\max_{\{(s',s)|b_1(s',s)=0\}}{}^{*}\left[\gamma_k^t(s',s) + \alpha_{k-1}^{t-1}(s') + \beta_k^{t-1}(s)\right]\right] - [b_{1,k}^{t-1,\mathrm{a}} + b_{3,k}^{\mathrm{a}}] \end{aligned} \tag{6.12}$$

$$\hat{b}_{1,k}^t = (b_{1,k}^{t-1,\mathrm{a}} + b_{1,k}^{t-1,\mathrm{e}} + b_{3,k}^{\mathrm{a}}) > 0 \tag{6.13}$$

Note that to aid our discussions, the notation t and $t-1$ is included in Equations (6.9) to (6.13), in order to explicitly indicate the clock period when each variable is calculated. After their transmission over a wireless channel, the received LLRs $\mathbf{b}_2^{\mathrm{u,a}} = [b_{2,k}^{\mathrm{u,a}}]_{k=1}^N$, $\mathbf{b}_3^{\mathrm{u,a}} = [b_{3,k}^{\mathrm{u,a}}]_{k=1}^N$, $\mathbf{b}_2^{\mathrm{l,a}} = [b_{2,k}^{\mathrm{l,a}}]_{k=1}^N$ and $\mathbf{b}_3^{\mathrm{l,a}} = [b_{3,k}^{\mathrm{l,a}}]_{k=1}^N$ are provided to the FPTD of Figure 6.3, where $\mathbf{b}_3^{\mathrm{l,a}}$ is obtained by interleaving $\mathbf{b}_3^{\mathrm{u,a}}$, as described in Section 6.1. Note that this notation does not include the period index t, since these LLRs remain constant throughout the iterative decoding process.

During the t^{th} clock period, the k^{th} block of Figure 6.3 combines the three *a priori* LLRs $b_{1,k}^{t-1,\mathrm{a}}$, $b_{2,k}^{\mathrm{a}}$, $b_{3,k}^{\mathrm{a}}$ using Equation 6.9, in order to obtain the branch metrics $\gamma_k^t(s',s)$. Similarly, in the same t^{th} clock period, the k^{th} block of Figure 6.3 employs Equations (6.10) and (6.11) to provide the vectors of *extrinsic* state metrics $\boldsymbol{\alpha}_k^t = [\alpha_k^t(s)]_{s=0}^7$ and $\boldsymbol{\beta}_{k-1}^t = [\beta_{k-1}^t(s')]_{s'=0}^7$, respectively. Still in the t^{th} clock period, the k^{th} block in the FPTD employs Equation 6.12 for providing the *extrinsic* LLR $b_{1,k}^{t,\mathrm{e}}$. The frame of *extrinsic* LLRs $\mathbf{b}_1^{t,\mathrm{e}} = [b_{1,k}^{t,\mathrm{e}}]_{k=1}^N$ is interleaved and provided to the other convolutional decoder as the *a priori* frame of LLRs $\mathbf{b}_1^{t,\mathrm{a}} = [b_{1,k}^{t,\mathrm{a}}]_{k=1}^N$. Finally, the FPTD employs Equation 6.13 for estimating the decoded bit $\hat{b}_{1,k}^t$. This iterative process is repeated until an accurate estimation of the decoded frame can be obtained, or until the maximum affordable number of iterations has been reached.

Note that in contrast to the operation of the Log-BCJR turbo decoder, the FPTD does not rely on forward and backward recursions in either the upper or lower convolutional decoders. More specifically, all $2N$ blocks of the FPTD of Figure 6.3 are operated simultaneously in the same t^{th} clock period. As a result of this, the FPTD requires only $T = 1$ clock period for completing each decoding iteration, compared to the $T = 4N$ clock periods required by the Log-BCJR algorithm. As an explicit benefit of this,

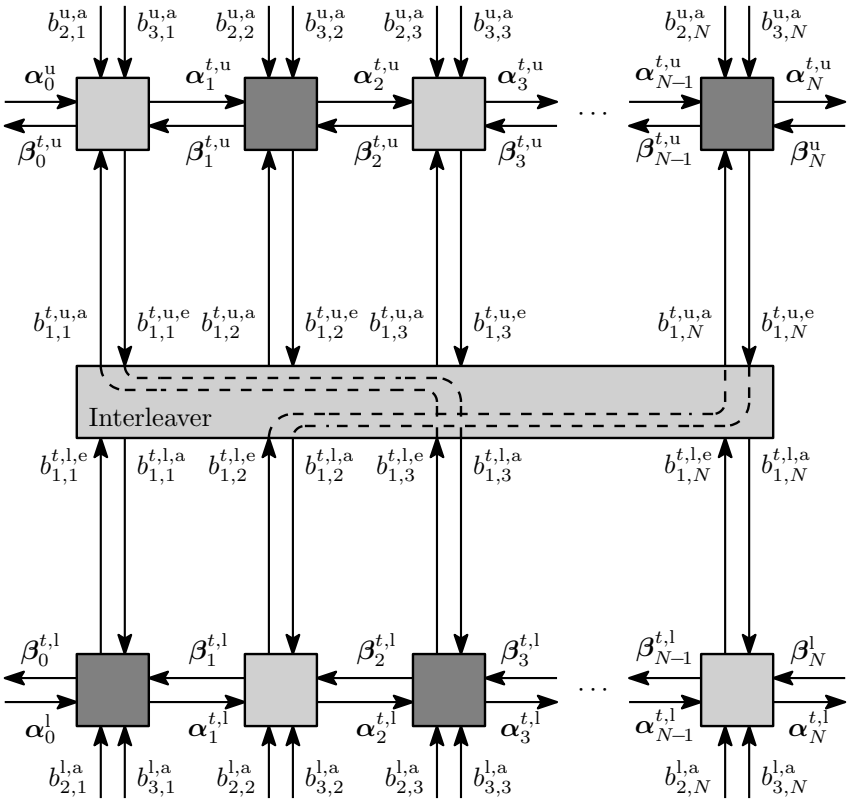


Figure 6.3: Block diagram of the FPTD algorithm [60, Fig. 1(b)].

the processing latency and throughput of the FPTD are significantly improved, when compared to those of the traditional Log-BCJR algorithm. However, this is achieved at the cost of an increased overall complexity, since more decoding iterations are required to achieve the same error correction performance, as detailed in [60].

When employed for the LTE turbo code [9], the FPTD algorithm benefits from the odd-even nature of the LTE interleaver [88]. More specifically, the LTE interleaver only connects odd-indexed blocks in the upper row of Figure 6.3 to odd-indexed blocks in the lower row. Similarly, even-indexed blocks in the upper row are only connected to even-indexed blocks in the lower row. As a result of this, none of the light-shaded blocks of Figure 6.3 are directly connected with each other, either within a row or via the interleaver. Likewise, none of the dark-shaded blocks of Figure 6.3 are directly connected with each other. This connection pattern allows the $2N$ blocks of Figure 6.3 to be grouped into two groups. The first group corresponds to the light-shaded blocks, comprising odd-indexed blocks in the upper row and even-indexed blocks in the lower row. Meanwhile, the second group corresponds to the dark-shaded blocks, comprising even-indexed blocks in the upper row and odd-indexed blocks in the lower row. Owing to this, the simultaneous operation in clock period t of the two groups of the FPTD may be considered as two independent iterative decoding process with no influence on each other, as detailed in [60]. Therefore, one of the two decoding processes may be considered to be redundant and may be discarded. This may be achieved by alternating

the operation of the light-shaded and dark-shaded groups in alternate clock periods t. In this configuration, one decoding iteration of the FPTD is completed in $T = 2$ consecutive clock periods. Note that this alternate operation of the light- and dark-shaded blocks of Figure 6.3 is naturally supported by Equations (6.9) to (6.13), since all variables having a time index of $(t-1)$ are provided by blocks having the opposite shading to that of the k^{th} block under consideration. Note that the odd-even operation of the FPTD achieves the same error correction capability using the same number of clock periods as the $T = 1$ approach. However, this is achieved with a 50% complexity reduction, owing to the operation of only half of the $2N$ blocks of Figure 6.3 in each clock period t.

In order to further reduce the complexity of the FPTD algorithm, the $\max^*$ operation of Equations (6.10) to (6.12) may be approximated by the max operation, at the cost of slightly degrading the BER performance of the decoder, in analogy with the Max-Log-BCJR algorithm, as described in Section 6.1. To overcome this BER performance degradation, it is possible to scale the *a priori* LLR $b_{1,k}^{t,\mathrm{a}}$ by a constant factor before it is used by the k^{th} block of the FPTD, as detailed in [89]. The scaling of $b_{1,k}^{t,\mathrm{a}}$ effectively reduces the confidence of the LLR, preventing its magnitude from rapidly growing in successive decoding iterations. Moreover, the scaling of the *a priori* LLRs is particularly useful in the hardware implementation of Fixed-Point (FX) decoders for the sake of improving their error correction capabilities, as we will demonstrate in Section 6.4

The LTE turbo encoder guarantees that the initial and final states of the transition diagram of Figure 4.1(c) are $s = 0$. This is achieved by employing termination bits at end of the encoding process of the N message bits. More specifically, each convolutional encoder of Figure 4.1(a) provides three parity termination bits $b_{1,N+1}$, $b_{1,N+2}$ and $b_{1,N+3}$, as well as three systematic termination bits $b_{2,N+1}$, $b_{2,N+2}$ and $b_{2,N+3}$. As a result of this, the received LLRs $b_{1,N+1}^{\mathrm{a}}$, $b_{1,N+2}^{\mathrm{a}}$, $b_{1,N+3}^{\mathrm{a}}$ and $b_{2,N+1}^{\mathrm{a}}$, $b_{2,N+2}^{\mathrm{a}}$, $b_{2,N+3}^{\mathrm{a}}$ are provided to three additional blocks positioned at the end of each row of the FPTD of Figure 6.3, as shown in Figure 6.4.

These three blocks perform a backward recursion before the FPTD iterative decoding process begins, using only Equations (6.9) and (6.11). Here $\beta_{N+3}(0) = 0$, while $\beta_{N+3}(s') = -\infty$ for all other $s' \in [1, 7]$. Likewise, $\alpha_0(0) = 0$ is used throughout the iterative decoding process, while $\alpha_0(s) = -\infty$ for all other $s \in [1, 7]$. The $\boldsymbol{\alpha}_k^0$ and $\boldsymbol{\beta}_{k-1}^0$ provided to all other blocks are initialized using zero-values before the start of the iterative decoding process. Similarly, the frame of *a priori* LLRs $\mathbf{b}_1^{0,\mathrm{a}}$ is initialized with zero-values.

In the general case where the interleaver pattern prevents the grouping of the $2N$ blocks of Figure 6.3 into two groups, the simultaneous operation in clock period t of all $2N$ blocks is recommended, as detailed in [60].

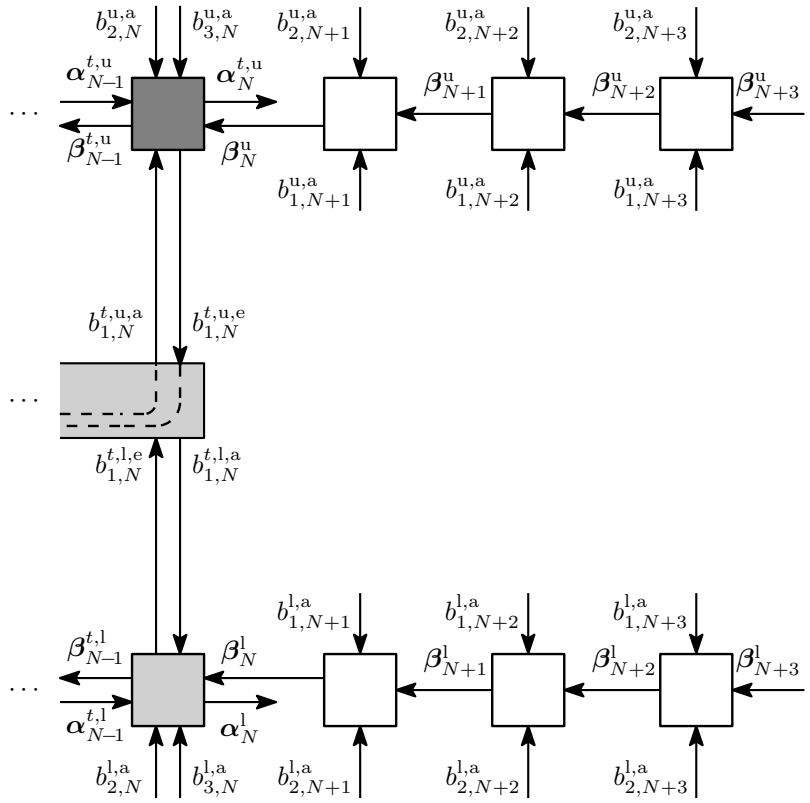


Figure 6.4: Schematic of the FPTD employing termination bits [60, Fig. 3].

6.2.2 Hardware Implementation

This section presents the hardware implementation requirements of the FPTD using FX two's complement arithmetic, for the case of implementing the LTE turbo decoder [9]. More specifically, we present the implementation of the 8-state FPTD using the state transition diagram of Figure 4.1(c) and odd-even interleavers [88] supporting a frame length in the range $N = [40, 6144]$, as specified in the LTE standard [9]. Figure 6.5 shows how Figure 6.3 can be modified to explicitly show the employment of D-type Flip Flops (DFFs) in the hardware implementation of the FPTD. Here, the blocks 'Reg' represent a group of DFFs for transferring LLRs between adjacent blocks of the FPTD. Each block of the FPTD employs the block diagram of the blue dashed box located in the bottom part of Figure 6.5 for the computation of Equations (6.9) to (6.13). In this configuration, we assume that the received LLRs provided by the demodulator are stored externally to the blocks of the FPTD. Note that the forward state metrics $\boldsymbol{\alpha}_N$, as well as the backward state metrics $\boldsymbol{\beta}_0$ are not passed on to any of the blocks of the FPTD. As a result of this, the blocks U_1 and L_1 may omit the implementation of the β module, as represented by the dotted lines in Figure 6.5. Similarly, the blocks U_N and L_N may omit the implementation of the α module.

The following sections detail the hardware implementation of Equations (6.9) to (6.13). We begin by describing how Noise-Dependent Scaling (NDS) may be employed in the

FPTD for overcoming the BER performance degradation associated with the employment of FX computations. Following this, we describe how the odd-even operation of the FPTD may be achieved. After this, we detail the number of quantization bits required by each LLR,state metric and intermediate variable in the FX representation of the FPTD algorithm. Finally, we detail the hardware requirements of Equations (6.9) to (6.13).

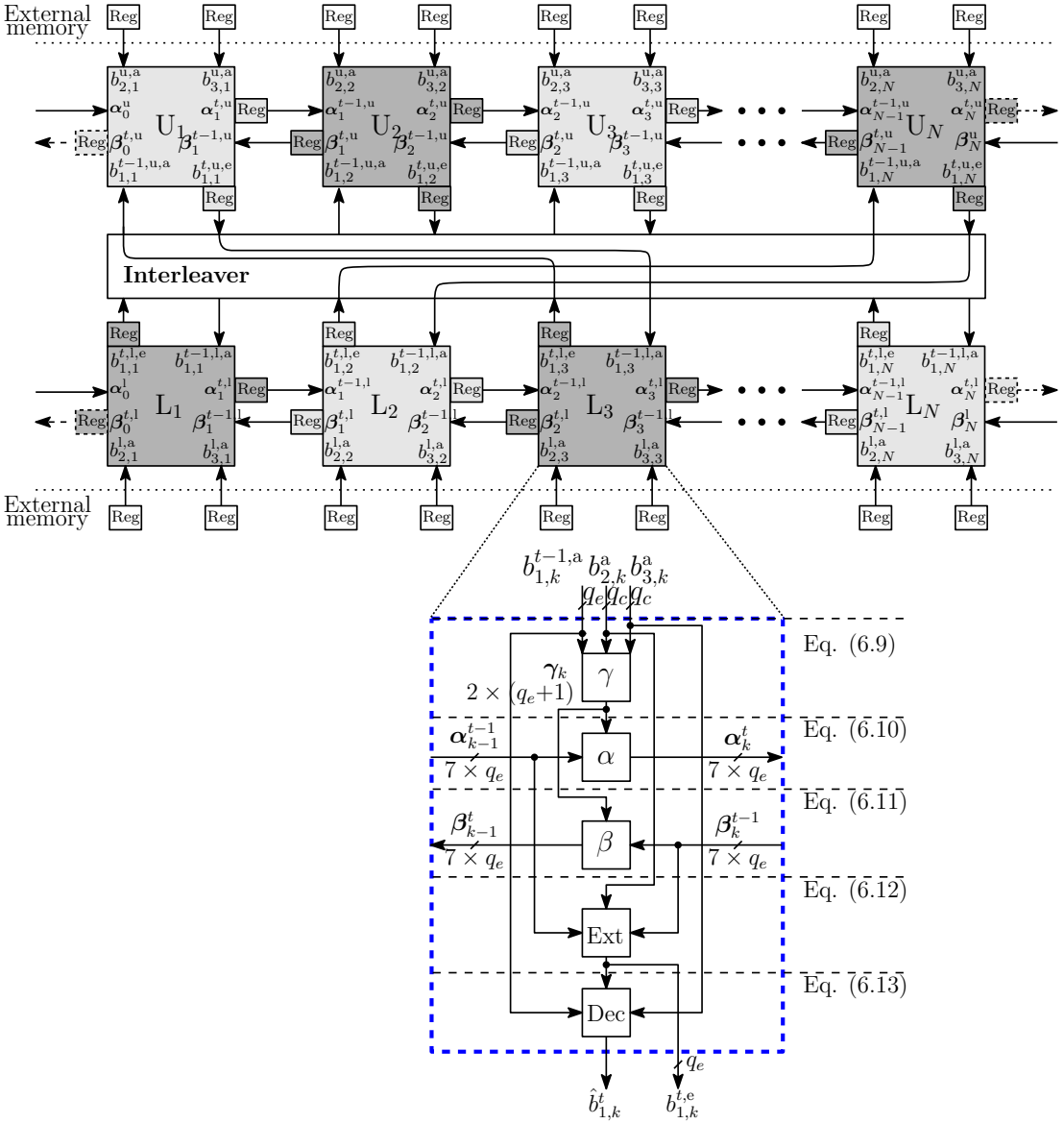


Figure 6.5: Block diagram of the FPTD showing the employment of registers.

6.2.2.1 Noise-Dependent-Scaling in FX FPTDs

As shown in Figure 6.5, the frames of parity LLRs $\mathbf{b}_2^{\mathrm{a}}$ and systematic LLRs $\mathbf{b}_3^{\mathrm{a}}$ are provided to each row of the FPTD. These LLRs are provided by the demodulator, which is assumed to employ NDS [29, 64] prior to converting the real-valued LLRs into

FX LLRs. NDS is particularly useful at high channel Signal to Noise Ratios (SNRs), since it effectively prevents the received LLRs from adopting either very large positive of very large negative values, which would saturate the FX number representation. In this way, NDS mitigates the BER performance degradation associated with the restricted dynamic range of FX computations [64]. In this method, the LLRs received from the channel are scaled depending on the channel's noise power spectral density N_0. Assuming a Binary Phase Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel, the k^{th} LLR $b^{a}_{i,k}$ is converted into a scaled LLR $b'^{a}_{i,k}$ according to

$$b'^{a}_{i,k} = \frac{\eta N_0}{\psi} \cdot b^{a}_{i,k}, \tag{6.14}$$

where η and ψ are parameters than can be chosen to optimize the BER performance of the decoder and $i \in \{2, 3\}$ for the case of the FPTD. Note that in order to facilitate our following discussions throughout the rest of this chapter, the scaled LLRs are simply referred to as $b^{a}_{i,k}$, since this does not modify the operation of the FPTD algorithm or its hardware implementation. Our experimental results of Section 6.4 show that different FPTD hardware implementations benefit from different NDS values, which are determined by the constants η and ψ. Moreover, since we assume that the demodulator performs the NDS of the received LLRs, its hardware implementation is not explored in this work.

6.2.2.2 Odd-Even scheduling

As mentioned in Section 6.2.1, the LTE FPTD benefits from the alternate operation between the light- and dark-shaded blocks of Figure 6.5. This behavior can be implemented in hardware by using clock gating for enabling and disabling the clock signals of each group of light- and dark-shaded Reg blocks of Figure 6.5, as shown in Figure 6.6. Here, two independent clock gating units, each comprising a D-type latch and an AND gate, are employed for generating the Clock_{light} and Clock_{dark} clock signals, respectively. This is achieved with the aid of the light/$\overline{\text{dark}}$ signal, which alternates its value in each clock cycle, as shown in Figure 6.6(b). Moreover, the light/$\overline{\text{dark}}$ signal may be generated by a simple clock divider. When light/$\overline{\text{dark}}$ takes the value of 1, the clock gating unit located in the top part of Figure 6.6(a) enables the Clock_{light} clock signal. In this condition, the clock gating unit located in the bottom part of Figure 6.6(a) disables the Clock_{dark} signal. By contrast, light/$\overline{\text{dark}}$ = 0 enables the clock signal Clock_{dark} and disables Clock_{light}. In this way, the two clock signals Clock_{light} and Clock_{dark} will never be enabled at the same time. Figure 6.6(b) exemplifies how these two clock signals may be employed by the FPTD for alternating the operation of the light- and dark-shaded blocks. Here, the output Q_{light} adopts the value of D_{light} only after the rising edge of Clock_{light} and retains its value otherwise. Similarly, the output Q_{dark} adopts the value of D_{dark} only after the rising edge of Clock_{dark} and retains its value otherwise.

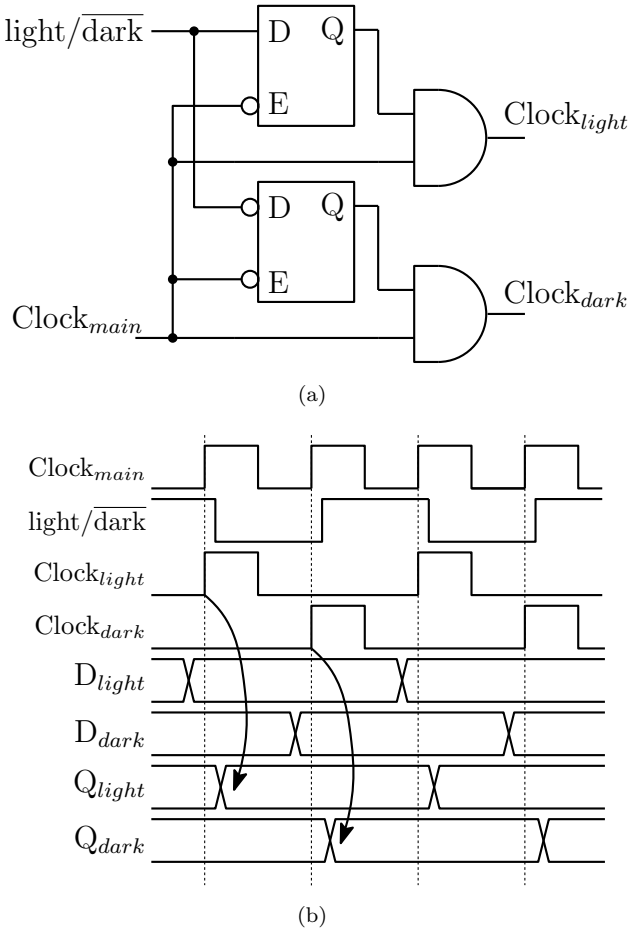


Figure 6.6: Clock signals for the light- and dark-shaded blocks of the FPTD. (a) Clock signals generation circuit. (b) Timing diagram.

Note that each of the three termination blocks of Figure 6.4 are operated only once before the beginning of the iterative decoding process, as described in Section 6.2.1. Owing to this, these termination blocks may employ the Clock$_{main}$ for their operation.

Alternatively to the generation of two independent clock signals, a single clock signal may be employed for updating all blocks of the FPTD. This may be achieved by operating each group of blocks on different edges of the clock, as detailed in [61]. In this configuration, the DFFs of the light-shaded blocks may update their contents only at the rising edge of the clock. Similarly, the DFFs of the dark-shaded blocks may update their contents only at the falling edge of the clock. This configuration requires the clock signal to have a duty cycle of 50%, in order to allow equal time for the operation of both groups of blocks. For the purpose of our investigations, we use the clock generation approach of Figure 6.6, since this configuration facilitates the employment of the error-tolerant design techniques that will be detailed in Section 6.6.

As mentioned in Section 6.2.1, when the interleaver pattern prevents the grouping of the $2N$ blocks of Figure 6.3 into two groups, all $2N$ blocks may be simultaneously operated

in each clock period t. As a result of this, all $2N$ blocks may employ the Clock_{main} signal for their operation and the clock generation circuits of Figure 6.6(a) may be omitted.

6.2.2.3 LLR quantization

As mentioned in Section 2.2, the number of quantization bits used in FX iterative decoders determines both their BER performance and their hardware complexity [64]. To elaborate further, the number of quantization bits q used in the two's complement number representation determines the dynamic range $[-2^{q-1}, 2^{q-1}-1]$ of the FX LLRs. Here, larger values of q result in larger dynamic ranges, which aid the iterative decoding process [64]. However, larger q values also results in larger hardware complexities, owing to the employment of larger arithmetic circuits. As a result of this, it is necessary to select a q value for the various *a priori* and *extrinsic* LLRs described in Section 6.2.1, in order to strike an attractive trade-off between BER performance and hardware complexity. For the purpose of our investigations, the received *a priori* parity and systematic LLRs $b^{\text{a}}_{2,k}$, $b^{\text{a}}_{3,k}$ are quantized using $q_c = 4$ bits. Similarly, the state metrics $\alpha_k(s)$, $\beta_{k-1}(s')$ and the *extrinsic* LLRs $b^{\text{e}}_{1,k}$ are quantized using $q_e = 6$ bits. Finally, the branch metrics $\gamma_k(s', s)$ are quantized using $q_e + 1 = 7$ bits. These values were chosen based on the best trade-off between the BER performance and hardware efficiency results, as will be presented in Section 6.4.

6.2.2.4 Branch metrics

The module γ of Figure 6.5 performs the scaling of the *a priori* LLR $b^{\text{t}-1,\text{a}}_{1,k}$ and provides the branch metrics $\gamma_k(s', s)$ of Equation 6.9 to the α and β modules. As mentioned in Section 6.2.1, the *a priori* LLRs $b^{\text{t}-1,\text{a}}_{1,k}$ may be scaled by a constant factor in order to mitigate the BER performance degradation associated with the employment of FX numbers [89]. Moreover, the value of the scaling factor can be chosen to optimize the BER performance of the FPTD. Our experimental results suggest that a scaling factor of $\tau = 3/4 = 0.75$ facilitates the best trade-off between BER performance and hardware complexity. More specifically, the multiplication $0.75 \times b^{t-1,\text{a}}_{1,k}$ can be implemented in FX arithmetic by performing logical shifts and additions, as shown in the lower part of Figure 6.7. Here, the *a priori* LLR $b^{\text{t}-1,\text{a}}_{1,k}$ is shifted one position to the left in order to obtain $2 \cdot b^{\text{t}-1,\text{a}}_{1,k}$. Following this, the shifted LLR is added to $b^{\text{t}-1,\text{a}}_{1,k}$ in order to obtain $3 \cdot b^{\text{t}-1,\text{a}}_{1,k}$. Finally, the result is truncated to obtain only the q_e Most Significant Bits (MSBs), discarding the two Least Significant Bits (LSBs). This truncation is equivalent to shifting $3 \cdot b^{\text{t}-1,\text{a}}_{1,k}$ two positions to the right in order to obtain $3 \cdot b^{\text{t}-1,\text{a}}_{1,k}/4$. Note that in this approach, the hardware complexity of the scaling is equivalent to only a single adder.

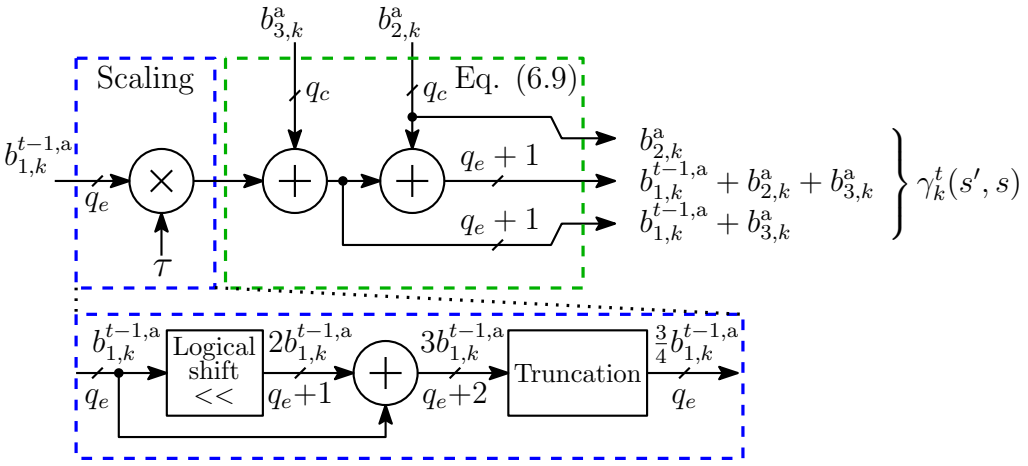


Figure 6.7: Schematic of the hardware implementation of the state metrics $\gamma^t_k(s',s)$ of the FPTD.

The scaled $b^{t-1,a}_{1,k}$ is added to the *a priori* LLRs $b^{a}_{2,k}$ and to $b^{a}_{3,k}$ in order to obtain the branch metrics $\gamma_k(s',s)$ of Equation 6.9. More specifically, Equation 6.9 quantifies the branch metric $\gamma_k(s',s)$ for each of the 16 transitions of the LTE transition diagram of Figure 4.1(c) from a previous state s' into the next state s. However, all of these 16 transitions are caused by one of only four possible combinations of the systematic and parity $b_1(s',s)$, $b_2(s',s)$ and $b_3(s',s)$, since $b_3(s',s) \equiv b_1(s',s)$. As a result of this, Equation 6.9 can be expressed as

$$\gamma^t_k(s',s) = \begin{cases} 0 & (s',s) = \{(0,0),(7,3),(1,4),(6,7)\} \\ b^{a}_{2,k} & (s',s) = \{(3,1),(4,2),(2,5),(5,6)\} \\ b^{t-1,a}_{1,k} + b^{a}_{3,k} & (s',s) = \{(2,1),(5,2),(3,5),(4,6)\} \\ b^{t-1,a}_{1,k} + b^{a}_{2,k} + b^{a}_{3,k} & (s',s) = \{(1,0),(6,3),(0,4),(7,7)\} \end{cases}$$

Note that some state metrics $\gamma^t_k(s',s)$ adopt zero-values and so there is no need for the module γ of Figure 6.5 to explicitly provide these values. Owing to this, the branch metrics $\gamma^t_k(s',s)$ of Equation 6.9 can be implemented as shown in the green box of Figure 6.7.

The additions $b^{t-1,a}_{1,k} + b^{a}_{3,k}$ and $b^{t-1,a}_{1,k} + b^{a}_{2,k} + b^{a}_{3,k}$ of $\gamma^t_k(s',s)$ require (q_e+1) bits in order to prevent these value from overflowing. By contrast, the *a priori* LLR $b^{a}_{2,k}$ may be provided to the α and β modules using only q_e bits.

6.2.2.5 State metrics

The modules α and β of Figure 6.3 compute the forward state metrics of Equation 6.10 and the backward state metrics of Equation 6.11, respectively. The hardware implementation of the state metrics is exemplified for the case of the forward state metric $\alpha^t_k(s)$ of Equation 6.10. The implementation of the backward state metrics $\beta^t_{k-1}(s')$ of

Equation 6.11 can be performed following the same principles. According to the state transition diagram of Figure 4.1(c) employed in the LTE standard, the forward state metrics $\alpha_k^t(s)$ of Equation 6.10 can be expressed for each of the $s = 8$ states as follows

$$\begin{aligned}
\alpha_k^t(0) &= \max[\alpha_k^{t-1}(0), \alpha_k^{t-1}(1) + b_{1,k}^{t-1,\mathrm{a}} + b_{2,k}^{\mathrm{a}} + b_{3,k}^{\mathrm{a}}] \\
\alpha_k^t(1) &= \max[\alpha_k^{t-1}(2) + b_{1,k}^{t-1,\mathrm{a}} + b_{3,k}^{\mathrm{a}}, \alpha_k^{t-1}(3) + b_{2,k}^{\mathrm{a}}] \\
\alpha_k^t(2) &= \max[\alpha_k^{t-1}(4) + b_{2,k}^{\mathrm{a}}, \alpha_k^{t-1}(5) + b_{1,k}^{t-1,\mathrm{a}} + b_{3,k}^{\mathrm{a}}] \\
\alpha_k^t(3) &= \max[\alpha_k^{t-1}(6) + b_{1,k}^{t-1,\mathrm{a}} + b_{2,k}^{\mathrm{a}} + b_{3,k}^{\mathrm{a}}, \alpha_k^{t-1}(7)] \\
\alpha_k^t(4) &= \max[\alpha_k^{t-1}(0) + b_{1,k}^{t-1,\mathrm{a}} + b_{2,k}^{\mathrm{a}} + b_{3,k}^{\mathrm{a}}, \alpha_k^{t-1}(1)] \\
\alpha_k^t(5) &= \max[\alpha_k^{t-1}(2) + b_{2,k}^{\mathrm{a}}, \alpha_k^{t-1}(3) + b_{1,k}^{t-1,\mathrm{a}} + b_{3,k}^{\mathrm{a}}] \\
\alpha_k^t(6) &= \max[\alpha_k^{t-1}(4) + b_{1,k}^{t-1,\mathrm{a}} + b_{3,k}^{\mathrm{a}}, \alpha_k^{t-1}(5) + b_{2,k}^{\mathrm{a}}] \\
\alpha_k^t(7) &= \max[\alpha_k^{t-1}(6), \alpha_k^{t-1}(7) + b_{1,k}^{t-1,\mathrm{a}} + b_{2,k}^{\mathrm{a}} + b_{3,k}^{\mathrm{a}}].
\end{aligned}$$

As a result of this, Equation 6.10 can be implemented using FX adders and max operations, as shown in Figure 6.8. Here, $(q_e + 1)$-bit FX adders are employed for the additions of the non-zero-values of $\gamma_k^t(s', s)$ with the incoming state metrics $\alpha_{k-1}^{t-1}(s)$, which were calculated by the $(k-1)^{th}$ block of the FPTD in the $(t-1)^{th}$ clock period. Note that the incoming forward state metrics $\alpha_{k-1}^{t-1}(s)$ are quantized using q_e bits. However, the branch metrics $\gamma_k^t(s', s)$ are quantized using $(q_e + 1)$ or q_c bits, according to the combination of previous and current state (s', s). Owing to this, the incoming forward state metrics $\alpha_{k-1}^{t-1}(s)$ are required to have the same number of quantization bits as $\gamma_k^t(s', s)$, before these LLRs can be provided to the FX adders. This is achieved by performing the sign extension of $\alpha_{k-1}^{t-1}(s)$ and of $\gamma_k^t(s', s) = b_{2,k}^{\mathrm{a}}$. More specifically, the MSB of each $\alpha_{k-1}^{t-1}(s)$ represents the sign of the LLR and is replicated $(q_e + 1) - q_e = 1$ times before being positioned as the MSBs of the sign-extended state metrics. Similarly, the MSB of $b_{2,k}^{\mathrm{a}}$ is replicated $(q_e + 1) - q_c = 3$ times and positioned as the two MSBs of the sign-extended $b_{2,k}^{\mathrm{a}}$. This process is not explicitly shown in Figure 6.8 for the sake of simplifying the schematic representation.

Following the addition of the incoming branch metrics and forward state metrics, the resulting $(q_e + 2)$-bit LLRs are provided to the max blocks, which perform the approximate max* operation. In analogy to the stochastic implementation of turbo decoders, the state metrics of the FPTD must be normalized to prevent the FX implementation of $\alpha_k^t(s)$ from overflowing. This may be achieved by subtracting $\alpha_k^t(0)$ from the vector of state metrics $\boldsymbol{\alpha}_k^t = [\alpha_k^t(s)]_{s=0}^7$ and clipping the result using q_e bits [61]. This is achieved using the subtractors and the 'sat' blocks of Figure 6.8. Here, the state metrics are saturated to the maximum value of 2^{q_e-1} and a minimum value of -2^{q_e} according to

$$\alpha_k^t(s) = \begin{cases} 2^{q_e-1} - 1 & \alpha_k^t(s) > 2^{q_e-1} - 1 \\ -2^{q_e-1} & \alpha_k^t(s) < -2^{q_e-1} \\ \alpha_k^t(s) & \text{otherwise.} \end{cases} \tag{6.15}$$

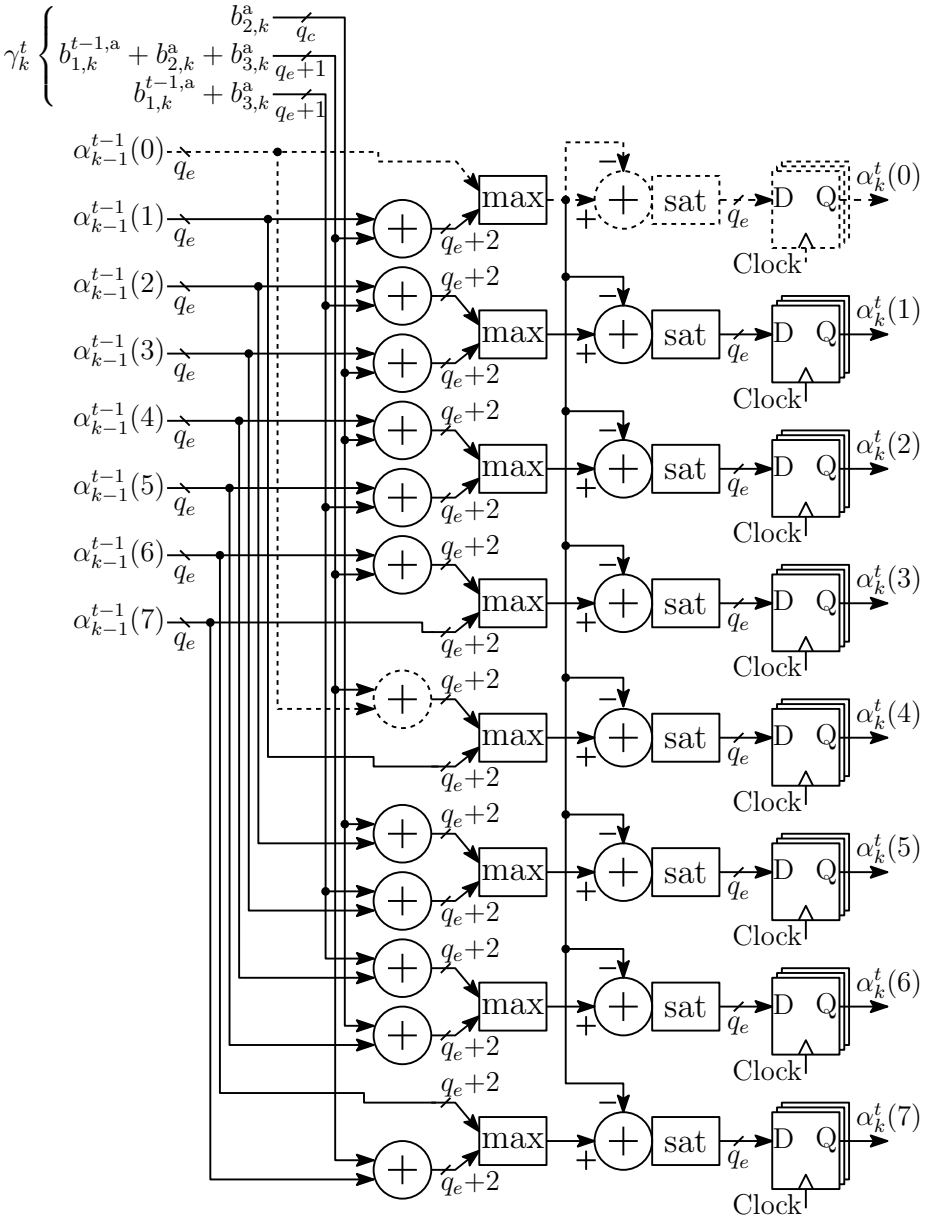


Figure 6.8: Schematic of the hardware implementation of the forward state metrics $\alpha_k^t(s)$ of the FPTD.

Note that in this configuration, $\alpha_k^t(0)$ adopts the constant value of 0 and so its hardware implementation may be omitted, as shown using dotted lines in Figure 6.8. Moreover, the clock signal of Figure 6.8 is simply referred to as 'Clock', since this signal may adopt either of the Clock_{light} and Clock_{dark} signals according to the index of the block of the FPTD, as detailed in Section 6.2.2.2.

In analogy to the hardware implementation of $\alpha_k^t(s)$, Figure 6.9 shows the hardware implementation of the state metrics $\beta_{k-1}^t(s')$ of Equation 6.11. As mentioned in Section 6.2.1, at the start of the decoding process, $\boldsymbol{\alpha}_k^t$ and $\boldsymbol{\beta}_{k-1}^t$ adopt zero values. An exception to this are the state metrics $\boldsymbol{\alpha}_0^t$ and $\boldsymbol{\beta}_{N+3}^t$. Here $\beta_{N+3}(0) = 0$, while $\beta_{N+3}(s') = -\infty$ for all other $s' \in [1, 7]$. Likewise, $\alpha_0(0) = 0$, while $\alpha_0(s) = -\infty$ for all other $s \in [1, 7]$.

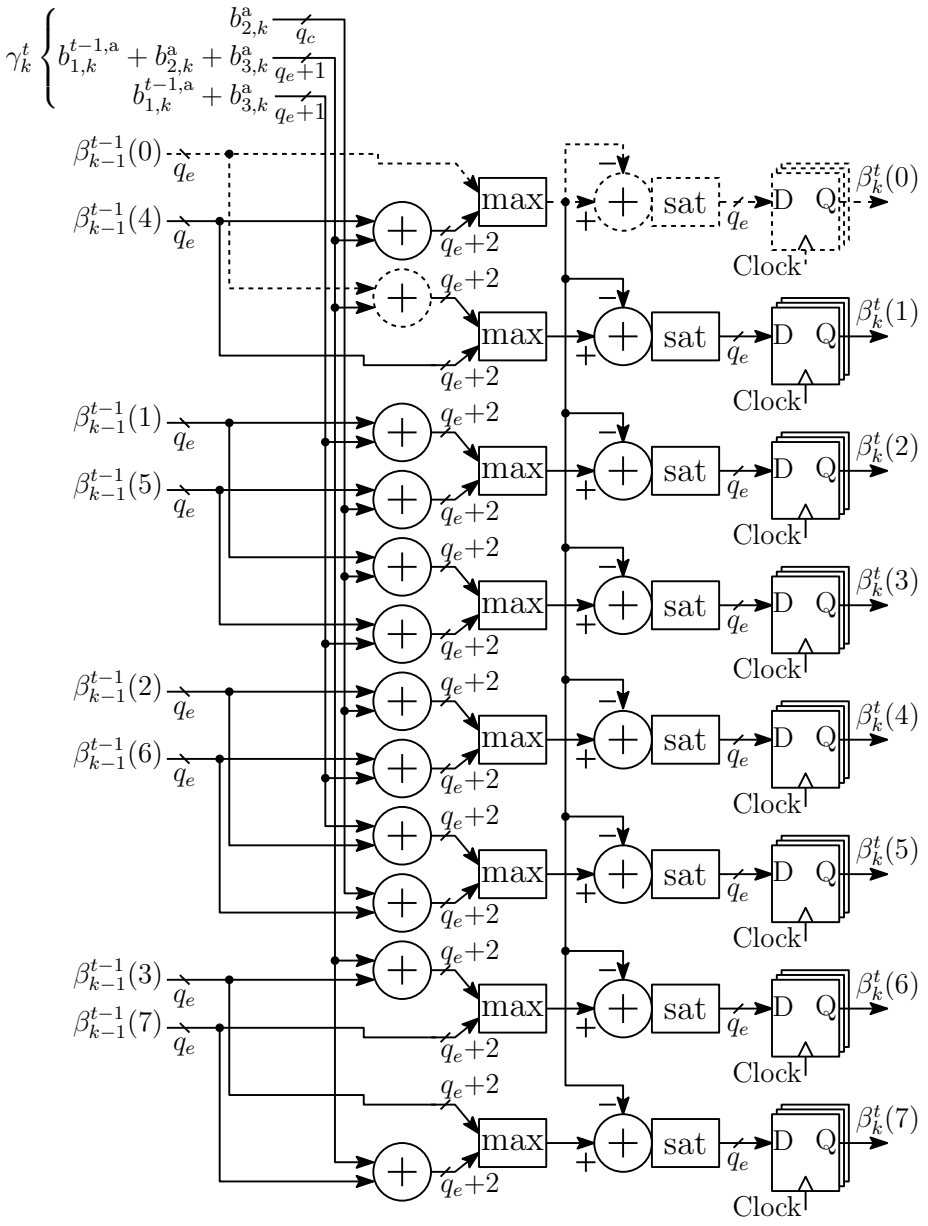


Figure 6.9: Schematic of the hardware implementation of the backward state metrics $\beta_{k-1}^t(s')$ of the FPTD.

Since $-\infty$ cannot be represented using FX numbers, this value may be replaced by the most negative number that can be represented by the FX number representation, namely -2^{q_e-1}. This may be achieved by initializing the MSB of each LLR of $\alpha_0(s)|_{s=1}^7$ and $\beta_{N+3}(s')|_{s'=1}^7$ with the bit value of 1 and the rest of the bits with logic 0.

6.2.2.6 Extrinsic LLR

The calculation of the *extrinsic* LLR of Equation 6.12 is performed by the module Ext of Figure 6.5. The hardware implementation of Equation 6.12 can be performed using the circuit of Figure 6.10. In the following discussion, the hardware implementation of the

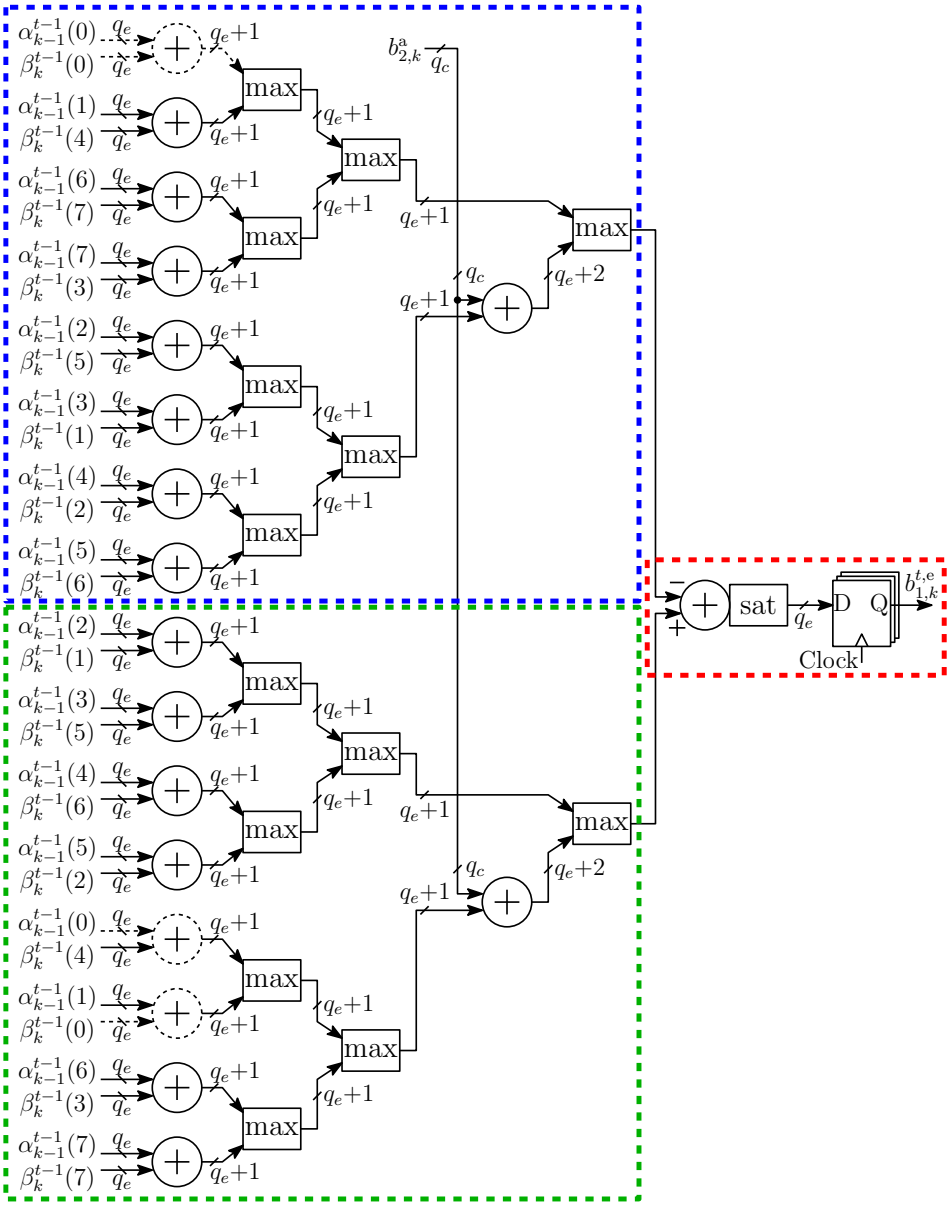


Figure 6.10: Schematic of the hardware implementation of the *extrinsic* LLR $b_{1,k}^{t,\mathrm{e}}$ of the FPTD.

extrinsic LLRs is described for the specific case where $b_1(s',s)=0$ in Equation 6.12, as shown in the blue box in the upper part of Figure 6.10. The hardware implementation of Equation 6.12 for the case where $b_1(s',s)=1$ can be performed following the same principles. According to the state transition diagram of Figure 4.1(c), the input bit $b_1(s',s)=0$ triggers the set of transitions $(s',s)=\{(0,0),\ (1,4),\ (2,5),\ (3,1),\ (4,2),\ (5,6),\ (6,7),\ (7,3)\}$. As a result of this, $\max^*_{\{(s',s)|b_1(s',s)=0\}}\left[\gamma_k^t(s',s)+\alpha_{k-1}^{t-1}(s')+\beta_k^{t-1}(s)\right]$ can

be expressed as

$$\begin{aligned}\max^*[&\gamma_k^t(0,0)+\alpha_{k-1}^t(0)+\beta_k^t(0),\gamma_k^t(1,4)+\alpha_{k-1}^t(1)+\beta_k^t(0),\\ &\gamma_k^t(2,5)+\alpha_{k-1}^t(2)+\beta_k^t(5),\gamma_k^t(3,1)+\alpha_{k-1}^t(3)+\beta_k^t(1),\\ &\gamma_k^t(4,2)+\alpha_{k-1}^t(4)+\beta_k^t(2),\gamma_k^t(5,6)+\alpha_{k-1}^t(5)+\beta_k^t(6),\\ &\gamma_k^t(6,7)+\alpha_{k-1}^t(6)+\beta_k^t(7),\gamma_k^t(7,3)+\alpha_{k-1}^t(7)+\beta_k^t(3)].\end{aligned}$$

Substituting the values of $\gamma_k^t(s',s)$ and exploiting the associative property of the $\max^*$ operations $\max^*(a,b,\ldots,x,y) = \max^*(\max^*(a,b),\max^*(\ldots),\ldots,\max^*(x,y))$ and the observation that $\max^*(a+c,b+c) = \max^*(a,b)+c$, we have

$$\begin{aligned}\max^*[&\alpha_{k-1}^t(0)+\beta_k^t(0),+\alpha_{k-1}^t(1)+\beta_k^t(4),\\ &\alpha_{k-1}^t(6)+\beta_k^t(7),+\alpha_{k-1}^t(7)+\beta_k^t(7),\\ &\alpha_{k-1}^t(2)+\beta_k^t(5)+b_{2,k}^{\mathrm{a}},+\alpha_{k-1}^t(3)+\beta_k^t(1)+b_{2,k}^{\mathrm{a}},\\ &\alpha_{k-1}^t(4)+\beta_k^t(2)+b_{2,k}^{\mathrm{a}},+\alpha_{k-1}^t(5)+\beta_k^t(6)+b_{2,k}^{\mathrm{a}}] =\\ \max^*[&A,B+b_{2,k}^{\mathrm{a}}],\end{aligned}$$

where

$$A = \max^*[\alpha_{k-1}^t(0)+\beta_k^t(0),\alpha_{k-1}^t(1)+\beta_k^t(4),\alpha_{k-1}^t(6)+\beta_k^t(7),\alpha_{k-1}^t(7)+\beta_k^t(7)]$$

and

$$B = \max^*[\alpha_{k-1}^t(2)+\beta_k^t(5),\alpha_{k-1}^t(3)+\beta_k^t(1),\alpha_{k-1}^t(4)+\beta_k^t(2),\alpha_{k-1}^t(5)+\beta_k^t(6)].$$

As a result of this, the hardware implementation of Equation 6.12 for the case where $b_1(s',s) = 0$ may be performed with the aid of FX adders and max blocks, as shown in the blue box of Figure 6.10. Note that the number of additions required has been minimized by performing only one addition of the *a priori* LLR $b_{2,k}^{\mathrm{a}}$. In analogy to the hardware implementation of the state metric of the α and β modules described in Section 6.2.2.5, the sign extension of the q_c-bit *a priori* LLR $b_{2,k}^{\mathrm{a}}$ is required, prior its addition with the (q_e+1)-bit shown in Figure 6.10.

The hardware implementation of Equation 6.12 for the case where $b_1(s',s) = 1$ is shown in the green box in the bottom part of Figure 6.10. Finally, the red box of Figure 6.10 shows the saturation of the *extrinsic* LLR $b_{1,k}^{t,\mathrm{e}}$ using q_e bits.

6.2.2.7 Hard Decision

The estimation of the decoded bit $\hat{b}_{1,k}^t$ of Equation 6.13 is performed by the module Dec of Figure 6.5. This is achieved with the aid of FX adders, as shown in Figure 6.11. Here, the hard decision is calculated using the addition of the *a priori* LLRs $b_{1,k}^{t-1,\mathrm{a}}$ and $b_{3,k}^{\mathrm{a}}$,

as well as the *extrinsic* LLR $b_{1,k}^{t-1,\mathrm{e}}$. The decoded bit $\hat{b}_{1,k}^{t}$ is calculated according to

$$\hat{b}_{1,k}^{t} = \begin{cases} 1 & (b_{1,k}^{t-1,\mathrm{a}} + b_{1,k}^{t-1,\mathrm{e}} + b_{3,k}^{\mathrm{a}}) > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{6.16}$$

which is implemented by inverting the MSB of the sum $b_{1,k}^{t-1,\mathrm{a}} + b_{1,k}^{t-1,\mathrm{e}} + b_{3,k}$.

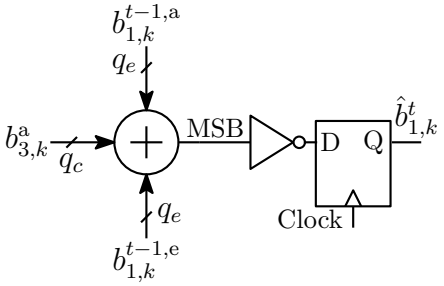


Figure 6.11: Schematic of the hardware implementation of the hard decision performed by the FPTD.

6.3 Reduced-Critical-Path Fully-Parallel Turbo Decoder

The FPTD hardware implementation presented in Section 6.2 exhibits a critical path comprising six data path stages. More specifically, one scaling operation and two additions are required for the calculation of the branch metrics $\gamma_k^t(s', s)$ of Equation 6.9, as shown in Figure 6.7. Following this, the branch metrics $\gamma_k^t(s', s)$ are combined in Equations (6.10) and (6.11) for calculating the state metrics $\alpha_k^t(s)$ and $\beta_{k-1}^t(s')$, respectively. This is achieved in three data path stages involving one addition, one max operation and one saturated subtraction, as shown in Figures 6.8 and 6.9. When combined with the three data path stages of the γ module, this results in a total of six data path stages, each equivalent to a FX adder. In parallel to this, the calculation of the *extrinsic* LLR $b_{1,k}^{t,\mathrm{e}}$ of Equation 6.12 employs six data path stages comprising two additions, three max operations and one saturated subtraction, as shown in Figure 6.10. This motivates a novel modification to the FPTD algorithm and hardware implementation for the sake of reducing the number of data path stages in the critical path. We refer to this modified algorithm as RCP-FPTD, which is described in Section 6.3.1. Moreover, the hardware implementation requirements of the RCP-FPTD algorithm are detailed in Section 6.3.2

6.3.1 Algorithm

The RCP-FPTD is described by Equations (6.17) to (6.22) for the specific case of a turbo decoder adopting the LTE standard.

$$\gamma_k^t(s',s) = b_1(s',s)\cdot b_{1,k}^{t-1,\mathrm{a}} + b_2(s',s)\cdot b_{2,k}^{\mathrm{a}} + b_3(s',s)\cdot b_{3,k}^{t-1,\mathrm{a}} \tag{6.17}$$

$$\alpha_k^t(s) = \max_{\text{all } s'}{}^{*}[\gamma_k^{t-1}(s',s) + \alpha_{k-1}^{t-1}(s')] \tag{6.18}$$

$$\beta_{k-1}^t(s') = \max_{\text{all } s}{}^{*}[\gamma_k^{t-1}(s',s) + \beta_k^{t-1}(s)] \tag{6.19}$$

$$\varepsilon_{k,m,n}^t = \max_{\{(s',s)|\{b_1(s',s)=m, b_2(s',s)=n\}\}}{}^{*}[\alpha_{k-1}^{t-1}(s') + \beta_k^{t-1}(s)] \tag{6.20}$$

$$\begin{aligned} b_{1,k}^{t,\mathrm{e}} = &\left[\max_{\{n\in\{0,1\}\}}{}^{*}\left[n\cdot b_{2,k}^{t-1,\mathrm{a}} + \varepsilon_{k,1,n}^{t-1}\right]\right] - \left[\max_{\{n\in\{0,1\}\}}{}^{*}\left[n\cdot \bar{b}_{2,k}^{t-1,\mathrm{a}} + \varepsilon_{k,0,n}^{t-1}\right]\right] \\ &+ \left[b_{1,k}^{t-1,\mathrm{a}} + b_{3,k}^{\mathrm{a}}\right] \end{aligned} \tag{6.21}$$

$$\hat{b}_{1,k}^t = (b_{1,k}^{t-1,\mathrm{a}} + b_{1,k}^{t-1,\mathrm{e}} + b_{3,k}^{\mathrm{a}}) > 0 \tag{6.22}$$

Equations (6.18) and (6.19) in clock period t are a function of the *a priori* branch metrics $\gamma_k^{t-1}(s',s)$ calculated in the previous clock period $(t-1)$. This is in contrast to Equations (6.10) and (6.11), which are functions of the branch metrics $\gamma_k^t(s',s)$ calculated in the *same* clock period t. Furthermore, Equations (6.20) and (6.21) split Equation 6.12 into two stages, which are calculated in two consecutive clock periods. This reduces the number of stages in the critical path from six to three. More specifically, the *extrinsic* LLR $b_{1,k}^{t,\mathrm{e}}$ of Equation 6.21 becomes a function of the intermediate calculations results $\varepsilon_{k,m,n}^{t-1}$ of Equation 6.20 obtained in previous clock period $(t-1)$. Note that the RCP-FPTD processes the state metrics at a different rate to the *extrinsic* LLRs. More specifically, while the *extrinsic* state metrics $\alpha_k^t(s)$ and $\beta_{k-1}^t(s')$ are affected by the *a priori* state metrics $\alpha_{k-1}^{t-1}(s')$ and $\beta_k^{t-1}(s)$ provided in the previous clock period $(t-1)$, they are not affected by the *a priori* LLR $b_{1,k}^{t-1,\mathrm{a}}$ provided in clock period $(t-1)$. Instead, they are affected by the *a priori* LLR $b_{1,k}^{t-2,\mathrm{a}}$ provided in clock period $(t-2)$. Similarly, the *extrinsic* LLR $b_{1,k}^{t,\mathrm{e}}$ is not affected by the *a priori* state metrics $\alpha_{k-1}^{t-1}(s')$ and $\beta_k^{t-1}(s)$ provided in the previous clock period $(t-1)$. Instead, it is affected by the *a priori* state metrics provided in clock period $(t-2)$, namely $\alpha_{k-1}^{t-2}(s')$ and $\beta_k^{t-2}(s)$. Owing to this, the propagation of the state metrics $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ along each row of the RCP-FPTD occurs faster than the propagation of the *extrinsic* LLRs $\mathbf{b}_1^{\mathrm{e}}$ between the two rows. As a result of this, the FPTD and RCP-FPTD algorithms are different from each other. This results in the RCP-FPTD algorithm requiring more decoding iterations for achieving the same BER performance as the FPTD algorithm. However, the RCP-FPTD algorithm exhibits a reduced number of data path stages, which facilitates the employment of lower clock periods. Hence the RCP-FPTD facilitates improved latencies and throughputs, when compared to the FPTD algorithm, as we will demonstrate in Section 6.3.2.

Figure 6.12 shows the representation of the RCP-FPTD, which exploits the odd-even interleaver design of the LTE standard. Here, the top and bottom shaded regions of each block implement Equations (6.17) and (6.21), respectively, while the middle shaded

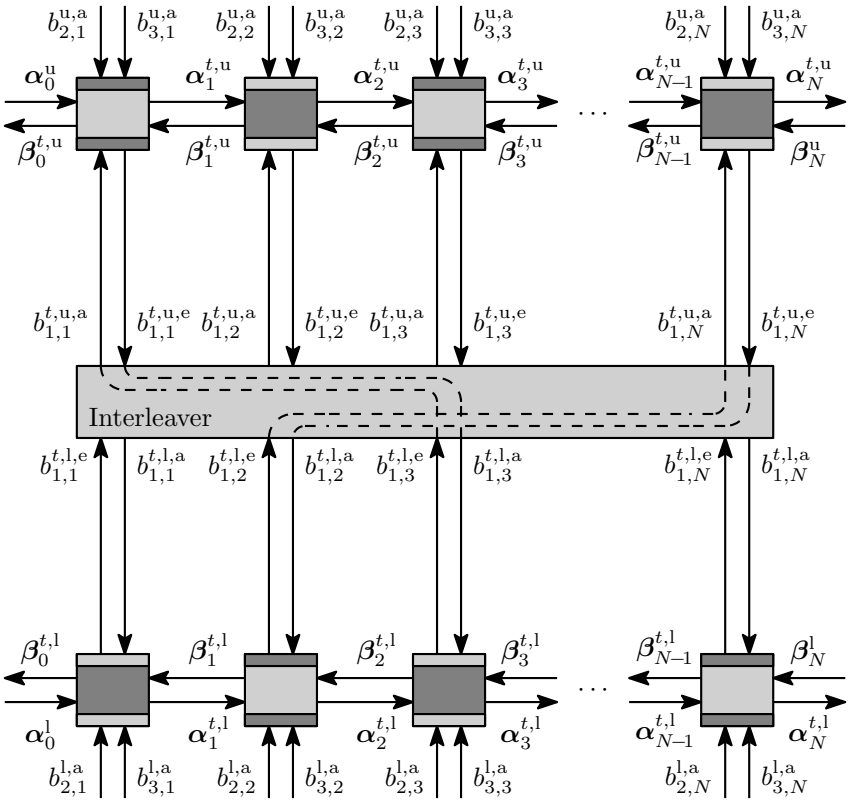


Figure 6.12: Block diagram of the RCP-FPTD algorithm.

regions implement Equations (6.18) to (6.20) and (6.22). In analogy to the FPTD of Section 6.2, the RCP-FPTD alternates the operation of the light-shaded regions and the dark-shaded regions of Figure 6.12 in alternate clock periods t. Note that the alternate operation of the light- and dark-shaded regions of Figure 6.12 is naturally supported by Equations (6.17) to (6.22), since all variables having a time index of $(t-1)$ are provided by regions having the opposite shading to that of the region under consideration.

In analogy to the FPTD algorithm, in the general case where the interleaver pattern prevents the grouping of the $2N$ blocks of Figure 6.12 into two groups, the simultaneous operation in clock period t of both light- and dark-shaded regions of Figure 6.12 is recommended.

6.3.2 Hardware Implementation

Figure 6.13 shows how Figure 6.12 may be modified to explicitly show the employment of DFFs in the hardware implementation of the RCP-FPTD algorithm. The hardware implementation of the RCP-FPTD employs DFFs for pipelining the calculation of the branch metrics $\gamma_k^t(s', s)$, as well as for pipelining the calculation of the intermediate results $\varepsilon_{k,m,n}^t$, as shown in the 'Reg' modules in lower part of Figure 6.13. The pipeline stages of the RCP-FPTD split the six-stage data paths of the α, β and Ext modules of the FPTD into two three-stage data paths, as described in Section 6.3.1. Moreover,

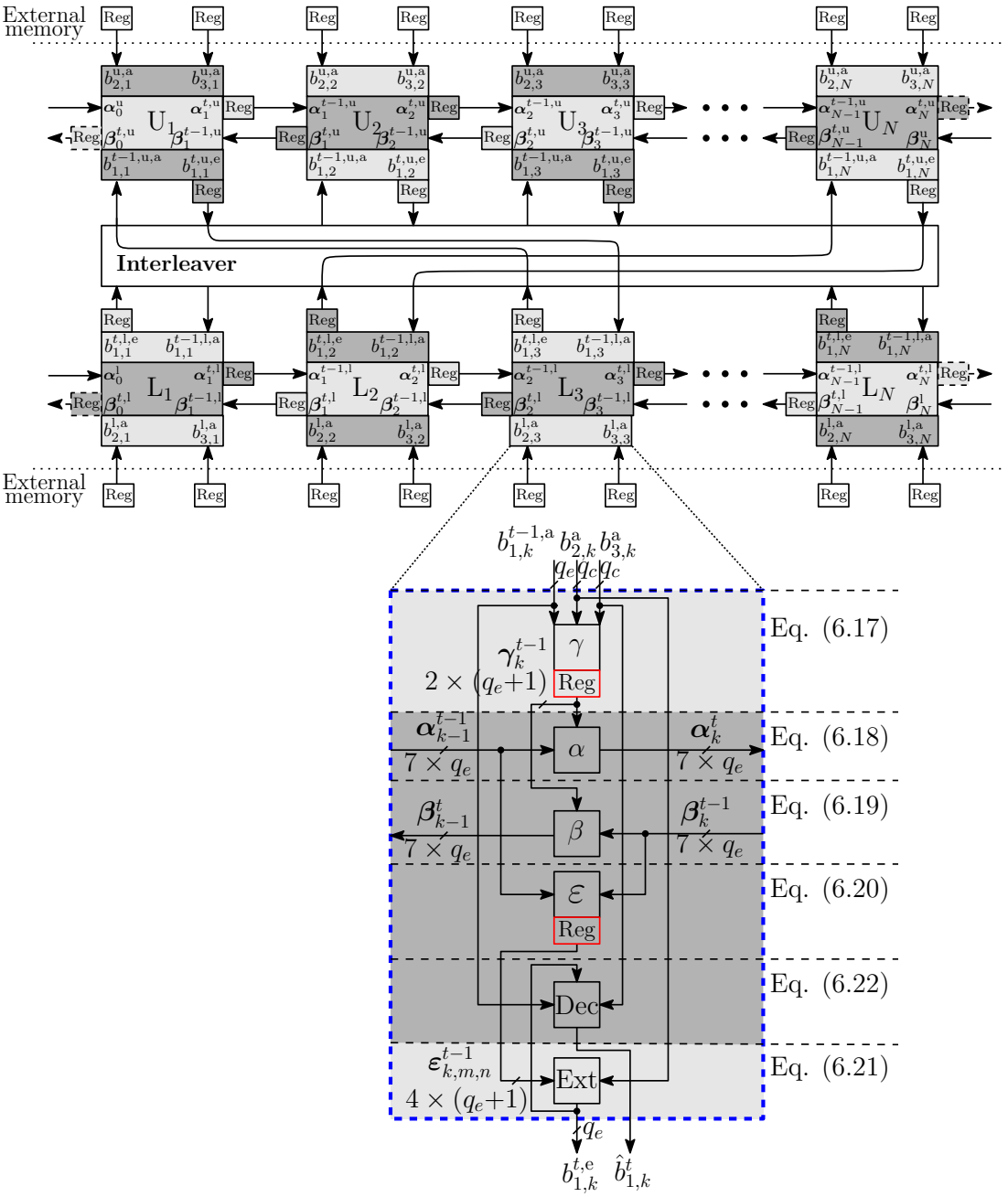


Figure 6.13: Block diagram of the RCP-FPTD algorithm.

the pipeline stage in the γ module causes the α and β modules to employ the branch metrics obtained in the previous clock period $(t-1)$. Similarly, the pipeline stage in the ε module causes the Ext module to employ the intermediate results $\varepsilon_{k,m,n}$ generated in the previous clock period $(t-1)$.

Figure 6.14 shows how Figure 6.7 may be modified in order to implement the pipeline stage in the γ module of Figure 6.13. The pipeline stage of the γ module does not modify the hardware implementation of the α and β modules, which implement Equations (6.18) and (6.19), respectively. Instead, the modules α and β of Figure 6.12 are implemented using the circuits shown in Figures 6.8 and 6.9 for the case of the FPTD, where the

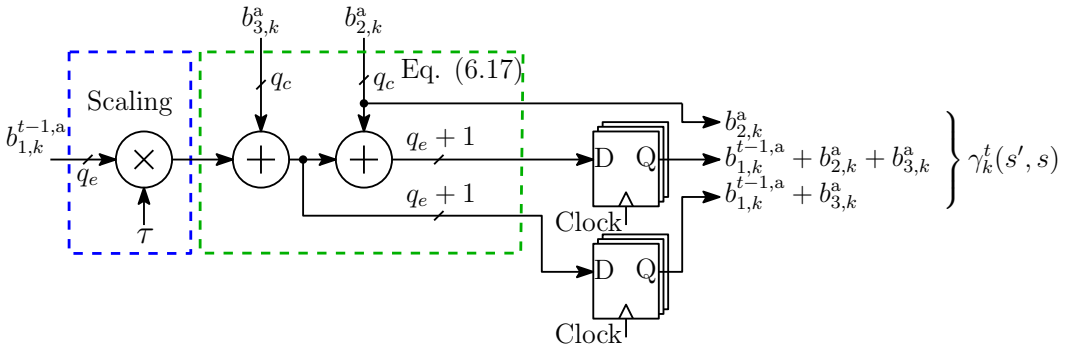


Figure 6.14: Schematic of the hardware implementation of the pipelined state metrics $\gamma^t_k(s',s)$ of the RCP-FPTD.

incoming branch metrics correspond to $\gamma^{t-1}_k(s',s)$ obtained in the previous clock period $(t-1)$.

The pipeline introduced in the calculation of the *extrinsic* LLRs $b^{t,\mathrm{e}}_{1,k}$ causes the module Ext of Figure 6.3 to be split into the modules ε and Ext in Figure 6.12. The hardware implementation of the module ε consists of a three-stage data path comprising one addition and two max operations, as shown in Figure 6.16.

The Ext module of the RCP-FPTD consists of a three-stage data path comprising one addition, one max and one saturated subtraction, as shown in Figure 6.15. Finally, the hardware implementation of the module Dec of the RCP-FPTD corresponds to that of the FPTD of Figure 6.11, since this module is not affected by any of the two pipeline stages.

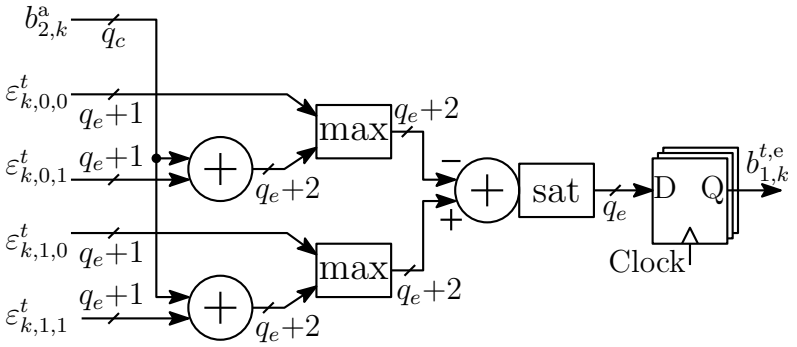


Figure 6.15: Schematic of the hardware implementation of the pipelined *extrinsic* LLR $b^{t,\mathrm{e}}_{1,k}$ of the RCP-FPTD.

In analogy to the hardware implementation of the FPTD presented in Section 6.2.2, the hardware implementation of the RCP-FPTD assumes the employment of NDS by the demodulator for scaling the received parity and systematic LLRs $\mathbf{b}^{\mathrm{a}}_2$ and $\mathbf{b}^{\mathrm{a}}_3$, respectively, as detailed in Section 6.2.2.1. Similarly, the RCP-FPTD employs the odd-even scheduling described in Section 6.2.2.2. Here, both of the clock signals Clock_{light} and Clock_{dark} of Figure 6.6 are provided to each block of Figure 6.13 for driving each of

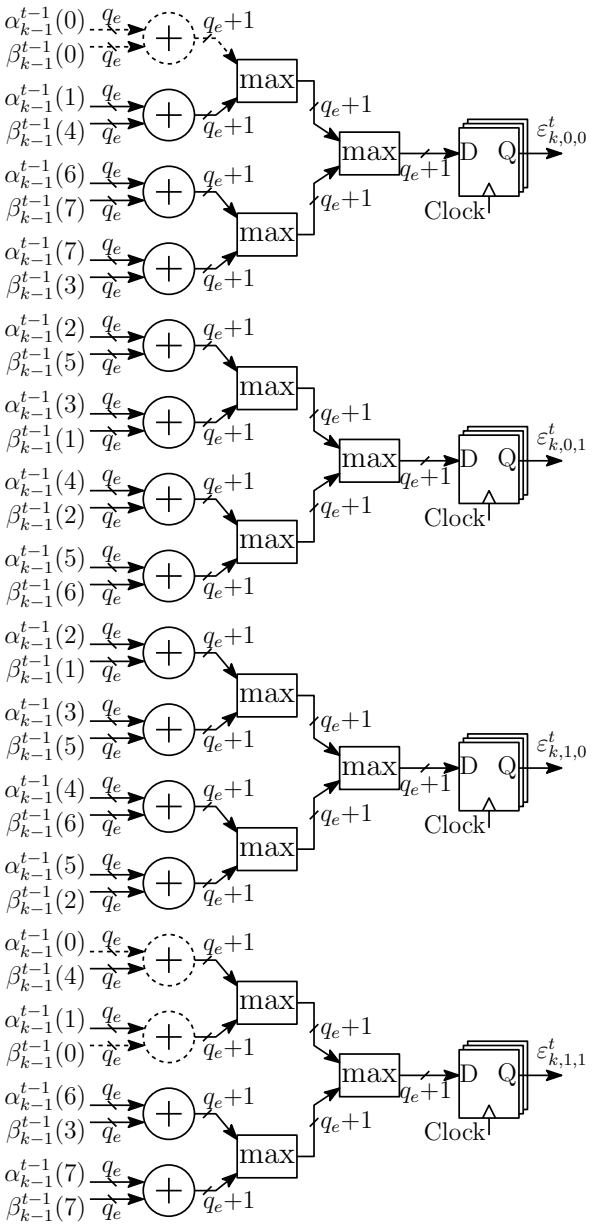


Figure 6.16: Schematic of the hardware implementation of the pipelined $\varepsilon_{k,m,n}^t$ of the RCP-FPTD.

the light- and dark-shaded regions of each block, respectively. This is contrast to the FPTD, where only one clock signal is required in each block. In addition to this, when the interleaver pattern prevents the grouping of the $2N$ blocks of into two groups, both light- and dark-shaded regions of Figure 6.12 may employ the Clock_{main} signal for their operation and the clock generation circuits of Figure 6.6(a) may be omitted.

6.4 Trade-off Analysis of the FPTD and RCP-FPTD Implementations

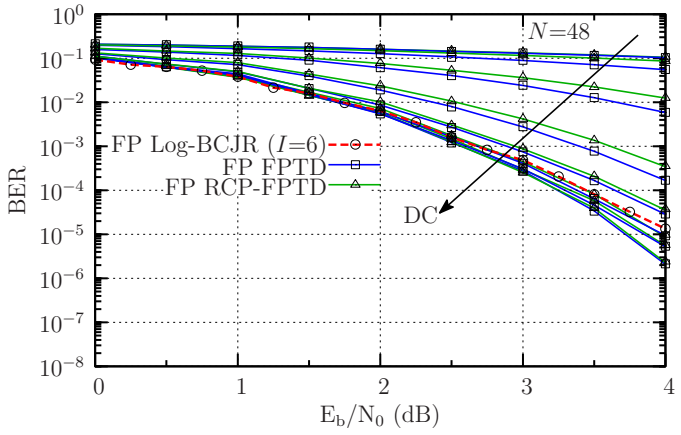
The following sections present the error correction capabilities of both the FPTD and the RCP-FPTD algorithms. We also characterize the trade-offs associated with the hardware implementation of the algorithms.

In the following discussions, we employ the concept of DC for the case of the FPTD algorithms, in analogy to the Stochastic LDPC Decoder (SLDPCD) of Chapter 3 and to the STDs of Chapters 4 and 5. As mentioned in Sections 6.2.1 and 6.3.1, one iteration of the FPTD and RCP-FPTD algorithm requires two consecutive clock periods for activating all $2N$ blocks of Figures 6.3 and 6.12, respectively. As a result of this, we refer to the activation of each group of blocks as a DC, where one iteration of the FPTDs is $I = 2\text{DC}$. Additionally, one DC corresponds to one clock period.

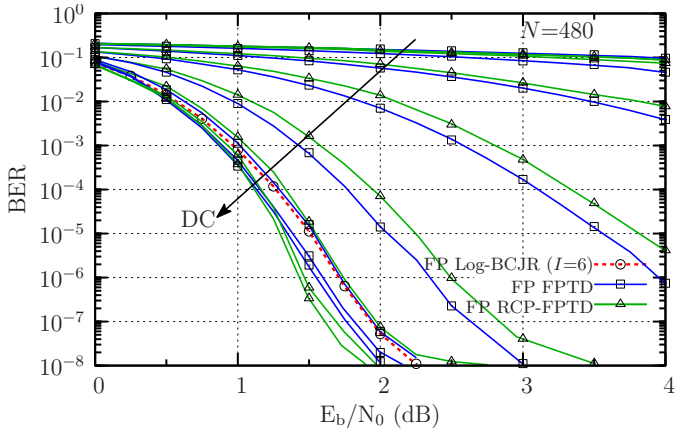
6.4.1 Error Correction Capabilities of the Floating Point FPTD and RCP-FPTD algorithms

Figure 6.17 presents the BER performance of the Floating-Point (FP) versions of both the FPTD and the RCP-FPTD algorithms, when adopting the LTE standard for the cases where the frame length is $N \in \{48, 480, 4800\}$. These results were obtained for the case of BPSK communication over an AWGN channel and using the exact $\max^*$ operation, when performing {2,4,8,16,32,64,128,256} DCs, which corresponds to {1,2,4,8,16,32,64,128} iterations.

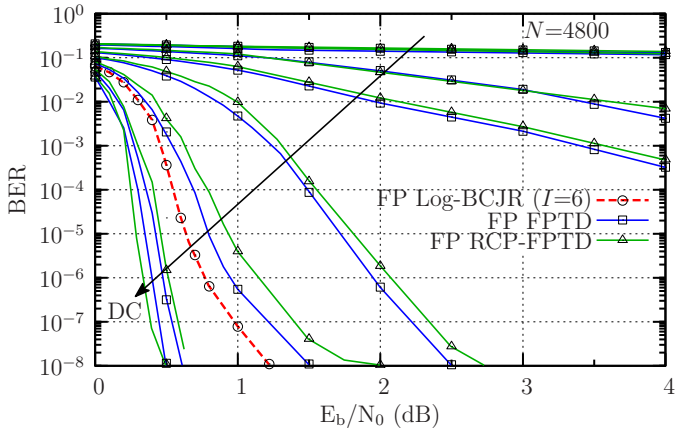
Figures 6.17(a) to 6.17(c) show how the BER of the FPTDs is only marginally improved by performing 128 or 256 DCs, instead of 64 DCs. Moreover, Figure 6.17 shows that the proposed RCP-FPTD algorithm exhibits a slight BER performance degradation, when employing the same number of DCs as the FPTD algorithm. However, the time duration of a DC of the RCP-FPTD may expected to be significantly lower than that of the FPTD, owing to the significantly reduced data path length of the RCP-FPTD, as described in Section 6.3. As a result of this, a fair comparison of the two algorithms is obtained by comparing the BER performance when the decoding time is fixed, as we will present in Section 6.4.3 and further expand in Section 6.6. Furthermore, Figure 6.17 compares the BER of the FPTD and the RCP-FPTDs algorithms to that of the FP version of the Log-BCJR algorithm, when using the exact $\max^*$ operation and performing $I = 6$ iterations. Figure 6.17 demonstrates that when performing 64 DCs, the FP versions of the FPTD and RCP-FPTD algorithms achieve the same BER performance as the Furthermore, when performing higher numbers of DCs, the FP FPTD and FP RCP-FPTD algorithms outperform the



(a)



(b)



(c)

Figure 6.17: BER performance of FP versions of the FPTD and RCP-FPTD algorithms, as well as of the FP Log-BCJR algorithm. The BER performance was obtained for the case of the exact $\max^*$ operation, when using BPSK modulation to transmit N={48,480,4800}-bit frames over an AWGN channel, where the FPTD and RCP-FPTD algorithms perform DC=$\in${2,4,8,16,32,64,128,256} DCs and the Log-BCJR algorithm performs $I = 6$ iterations. (a) $N = 48$. (b) $N = 480$. (c) $N = 4800$.

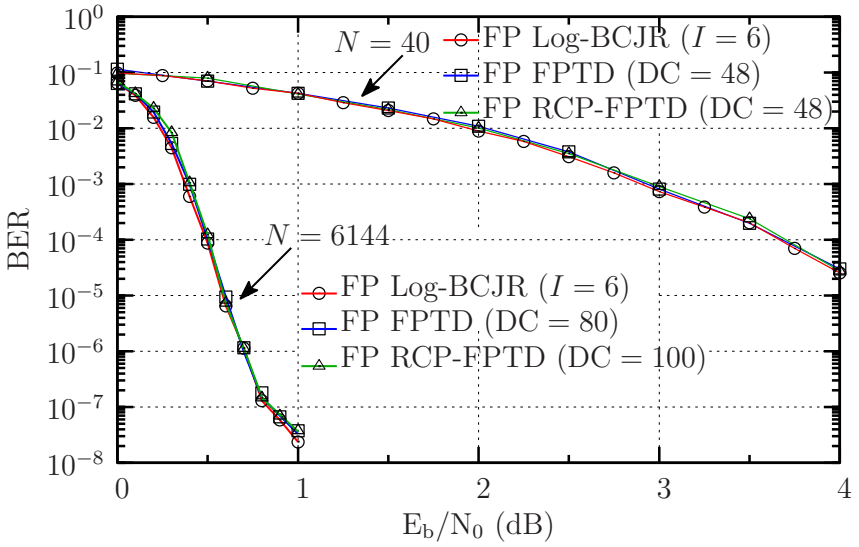


Figure 6.18: BER performance of FP versions of the FPTD, RCP-FPTD and Log-BCJR algorithms. The BER performance was obtained for the case of the exact $\max^*$, when using BPSK modulation to transmit $N \in \{40, 6144\}$-bit frames over an AWGN channel.

Figure 6.18 presents the BER performance of the FP FPTD, FP RCP-FPTD and Here, the number of DCs performed by the FP FPTD and FP RCP-FPTD algorithms is set to the values required to obtain the same BER performance as the Note that the FP version of the FPTD and RCP-FPTD algorithms may seem to require a large number of DCs to achieve the same BER as the However, one iteration of the Log-BCJR requires $4N$ consecutive clock periods, as detailed in Section 6.1. As a result of this, the Log-BCJR algorithm requires a total of $6 \times 4 \times 40 = 960$ consecutive clock periods for a frame length of $N = 40$ bits, compared to the 48 DCs required by both the FPTD and RCP-FPTD algorithms. Similarly, the Log-BCJR requires $6 \times 4 \times 6144 = 147456$ consecutive clock periods for the case of a frame length of $N = 6114$ bits, compared to the 80 and 100 DCs required by the FPTD and RCP-FPTD algorithms, respectively.

6.4.2 Error Correction Capabilities of the Fixed-Point FPTD and RCP-FPTD algorithms

Figure 6.19 presents the BER performance of FX versions of both the FPTD and RCP-FPTD algorithms presented in Sections 6.2.2 and 6.3.2, respectively, when using the approximate $\max^*$ operation of Equation 6.8 and frame lengths of $N \in \{40, 6144\}$-bits. Figure 6.19 also plots the BER of the FP versions of the FPTD and RCP-FPTD algorithms as benchmarks. The results of the FX FPTD and RCP-FPTD were obtained for the cases where the LLRs are quantized using the $\{q_e, q_c\}$ values specified in Table 6.1. Additionally, we employ NDS associated with the values $\{\eta, \psi\}$ of Table 6.1. The values of $\{q_e, q_c\}$ were chosen based on the best trade-off between BER performance and

hardware complexity. Similarly, the values of $\{\eta, \psi\}$ of Table 6.1 were obtained based on the best BER performance of each FX FPTD.

Table 6.1: Simulation parameters for the FX FPTD and FX RCP-FPTD.

	FPTD		RCP-FPTD	
Parameter	$N = 40$	$N = 6144$	$N = 40$	$N = 6144$
$\{q_c, q_e\}$	{4,6}	{4,6}	{4,6}	{4,6}
$\{\eta, \psi\}$	{1,2}	{7,10}	{1,2}	{7,10}

Figure 6.19 shows that when $N = 40$, the FX versions of the FPTD and RCP-FPTD algorithms achieve similar BER performance as their FP counterparts using the exact $\max^*$ operation of Equation 6.7 and employing the same number of DCs. However, when $N = 6144$, the FX versions exhibit a 0.3 dB E_b/N_0 degradation, when compared to their FP counterparts using the exact $\max^*$ operation of Equation 6.7 and the same number of DCs. However, owing to their use of NDS and to the scaling of the *a priori* LLR $b^a_{1,k}$, the FX versions of the FPTD and RCP-FPTD algorithms offer slight BER improvements, when compared to their FP counterparts using the approximate $\max^*$ operation of Equation 6.8.

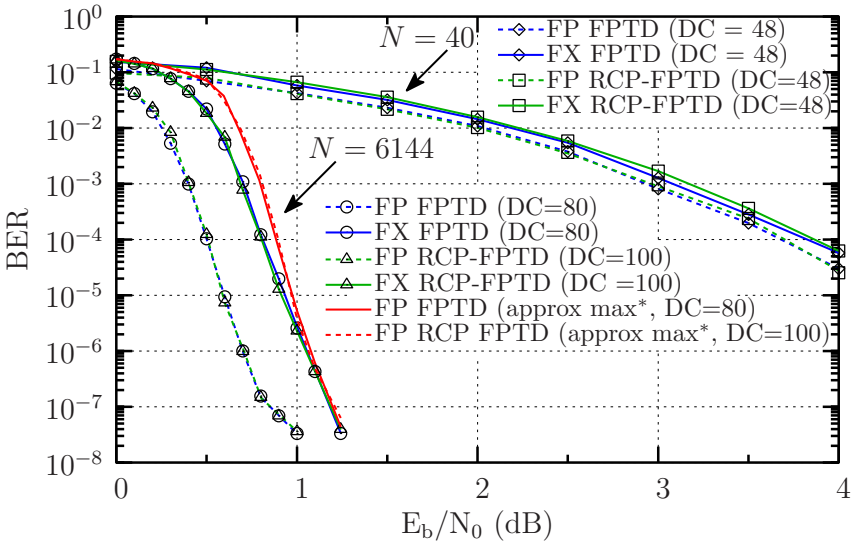


Figure 6.19: Comparison of the BER performance of the FX and FP FPTDs. The BER performance of the FP FPTDs was obtained for the cases of both the exact and approximate $\max^*$, when $N \in \{40, 6144\}$

Our experimental results suggest that the BER performance of the FPTD and of the RCP-FPTD implementations remains unchanged, whether the three termination blocks of Figure 6.4 are operated prior the beginning of the decoding process or during the first three DCs. As a result of this, the three termination blocks of the FPTD and RCP-FPTD implementations are operated simultaneously to all other blocks, only during the first

three DCs of the iterative decoding process. Based on these results, we recommend 80 and 100 as the maximum number of DCs performed by the FX versions of the FPTD and the RCP-FPTD algorithms, respectively.

6.4.3 Hardware Efficiency

This section presents the different hardware trade-offs associated with the FPTD and RCP-FPTD hardware implementations of Sections 6.2 and 6.3, respectively. Table 6.2 characterizes the FPTD and RCP-FPTD hardware implementations in terms of diverse characteristics, including the chip area per decoded bit, number of equivalent NAND gates, clock period, clock frequency, latency, throughput, as well as as area and energy efficiency, when using Taiwan Semiconductor (TSMC) 40 nm technology. The results of Table 6.2 were obtained from the physical layout generated by the automatic place and route of one block of the FPTD and RCP-FPTD implementations using Cadence SoC Encounter [72]. These results were obtained for the cases where the supply voltage is set to 1.0 V and in the absence of power supply or clock period variations. Both the critical clock period and the energy consumption are obtained from Synopsys PrimeTime [73]. For the case of using early stopping, the average number of DCs, latency, throughput and energy efficiency were obtained from post-layout gate-level simulations with the extracted parasitics and annotated delays without timing errors, when allowing a maximum of 80 and 100 DCs for the FPTD and RCP-FPTDs, respectively. For the purpose of our investigations throughout this chapter, we assume that an external fully-parallel Cyclic Redundancy Check (CRC) [84] is employed in each DC for determining whether the frame of estimated decoded bits provided by the FPTD and RCP-FPTD contains any errors. We assume that the FPTDs operate at their critical clock period, for BPSK transmission over an AWGN channel having the particular SNR where a BER of 10^{-5} is achieved.

We first compare the hardware efficiency of the FPTD and RCP-FPTD implementations with the RLSTD presented in Chapter 5. Table 6.2 presents results for the rate 48/156, 8-state, FPTD and RCP-FPTD having a frame length of $N = 48$ bits, using the LTE interleaver and the state transition diagram of Figure 4.1(c). We also present the hardware efficiency of the 50-bit, rate 1/3, 8-state, tailbiting RLSTD of Chapter 5, when using an S-Random interleaver and TSMC 90 nm technology. In order to facilitate a fair comparison with the FPTD and RCP-FPTD implementations, the characteristics of the RLSTD in 90 nm technology have been scaled to 40 nm technology, in accordance to the scaling techniques of [90]. Here, the technology scaling factor $s = 40/90$ corresponds to the ratio between the two different technologies. More specifically, the 90 nm chip area is scaled according to $\text{Area}_{40\text{nm}} = \text{Area}_{90\text{nm}} \times s^2$, the 90 nm clock period is scaled according to $T_{\text{clk 40nm}} = T_{\text{clk 40nm}} \times s$ and the 90 nm energy consumption is scaled according to $\text{Energy}_{40\text{nm}} = \text{Energy}_{90\text{nm}} \times s \times \text{V}_s{}^2$, where $V_s = 1/1.2$ corresponds to the ratio of

the nominal supply voltages in 40 nm and 90 nm, respectively. The results of chip area of Table 6.2 do not consider the hard-wired interleaver connecting the upper and lower rows of each FPTD. However, it is assumed that the hard-wired interleaver may be routed using the different metal layers available in TSMC 40 nm technology, without significantly increasing the chip area requirements, as detailed in [61]. Similarly, it is assumed that the clock distribution network does not impose additional timing constraints when increasing the frame length from $N = 40$ to $N = 6144$.

Table 6.2: Hardware efficiency of short-frame FPTDs.

	FPTD	RCP-FPTD	RLSTD
Implementation	FX	FX	Stochastic
Frame length (bits)	48	48	50
Coding rate	48/156	48/156	1/3
Technology	40 nm	40 nm	90 nm
Voltage	1.0 V	1.0 V	1.2 V
Area per bit (μm^2)	6800	7100	11000 (4888)[a]
Gate count per bit	13.2 K	14.0 K	4.4 K
T_{clk} (ns)	4.5	3.0	1.7 (0.755)[b]
Frequency (MHz)	222	333	588 (1324)[b]
BER @ E_b/N_0	10^{-5} 4.0 dB	10^{-5} 4.0 dB	10^{-5} 3.8 dB
Average DCs	64	64	200
Latency (ns)	288	192	340 (0.151)[b]
Throughput (Mbps)	166	250	147 (331)[b]
Area eff. (bps/gates)	262	384	669 (1504)[a,b]
Area eff. (Mbps/mm^2)	508	744	267 (1354)[a,b]
Energy eff. (nJ/frame)	72	74	38 (12)[a,b,c]

[a] Area $\sim s^2$
[b] $T_{\text{clk}} \sim s$
[c] Energy $\sim s \cdot \text{V}_s^2$

As discussed above, Figure 6.19 demonstrates that the RCP-FPTD algorithm requires a larger number of DCs than the FPTD for achieving the same BER performance. However, Table 6.2 shows that the latency and throughput of the RCP-FPTD implementation are superior to those of the FPTD implementation, owing to the employment of a lower clock period. Moreover, Table 6.2 demonstrates that both the FPTD and RCP-FPTD algorithms require fewer DCs than the RLSTD algorithm. This facilitates

latency, throughput and area efficiency improvements in terms of Mbps/mm^2, when compared to those of the RLSTD using 90 nm technology. However, the RLSTD outperforms both the FPTD and RCP-FPTD implementations, if the implementation results of the RLSTD are scaled to a 40 nm technology. Note that the RLSTD uses an S-Random interleaver, in contrast to the LTE interleaver used in the FPTD and RCP-FPTD implementations. Owing to this, the number of DCs, latency, throughput and area efficiency of the RLSTD may be further improved, when using the LTE interleaver. These results suggest that, when using short-frame lengths and the same technology scaling, the RLSTD offers a superior performance over the FPTD and RCP-FPTD, which confirms that RLSTD is suitable for short-frame length applications, such as MCMTC, as described in Chapter 5. Motivated by these results, we now compare the hardware efficiency of the FPTDs when using a frame length of N =6144.

Table 6.3 compares the hardware efficiency of the proposed FPTD and RCP-FPTD implementations with different state-of-the-art Log-BCJR LTE hardware implementations. Note that the proposed FPTD and RCP-FPTD implementation achieve the lowest latency and therefore the highest throughput, compared to all other implementations. More specifically, the proposed FPTD and RCP-FPTD implementations improve the throughput by factors of 7.9 and 9.5, respectively, when compared to that of the state-of-the-art turbo decoder of [57]. Moreover, although the proposed FPTD and RCP-FPTD have the largest area and gate count per bit, their normalized area efficiency is superior to that of [59]. Furthermore, the proposed FPTDs have lower energy consumptions than the other implementations.

6.5 Fixed-Point FPTDs in the Presence of Timing Errors

This section investigates the inherent tolerance to timing errors of the FPTD and RCP-FPTD algorithms. This is achieved by characterizing the BER performance and hardware efficiency of the FX FPTD and RCP-FPTD implementations in the presence of timing errors owing to the employment of overclocking for the sake of further improving the attainable throughput of the decoders. Here, the employment of overclocking is justified since the critical path delay of a Very-Large-Scale Integration (VLSI) circuit may be rarely incurred, with most clock cycles having significantly lower propagation delays. Hence timing errors may only occasionally occur when employing overclocking and the performance of the circuit may not be significantly degraded. In this way, the system may be operated using a clock period below the delay of the critical path, for the sake of improving its hardware characteristics such as latency, throughput and energy efficiency [19]. Sections 6.5.1 and 6.5.2 provide a timing analysis and a timing error model, respectively, of the FPTD and RCP-FPTD implementations. Following this,

Table 6.3: Hardware efficiency of long-frame FPTDs.

	FPTD	RCP-FPTD	Ilnseher 2012, [57]	Wang 2014, [58]	Sun 2011, [59]
Frame length (bits)	6144	6144	6144	6144	6144
Coding rate	6144/18444	6144/18444	6144/184444	1/3	6144/6468
Technology	40 nm	40 nm	65 nm	45 nm	65 nm
Voltage	1.0 V	1.0 V	1.1 V	0.81 V	0.9 V
Area per bit (μm^2)	6800	7000	1280 (484)[a]	395 (312)[a]	1350 (511)[a]
Gate count per bit	13.2 K	13.5 K	-	244	940
T_{clk} (ns)	4.5	3.0	2.2 (1.35)[b]	1.6 (1.42)[b]	2.5 (1.54)[b]
Frequency (MHz)	222	333	450 (740)[b]	600 (704)[b]	400 (649)[b]
Average iterations	80 DCs	100 DCs	6 I	5.5 I	6 I
Latency (ns)	360	300	2857 (1723)[b]	3679 (3200)[b]	4800 (2953)[b]
Throughput (Gbps)	17.1	20.48	2.15 (3.56)[b]	1.67 (1.92)[b]	1.28 (2.08)[b]
Area eff. (bps/gates)	210	247	-	1114	221
Area eff. (Mbps/mm^2)	409	476	279 (1197)[a,b]	687 (871)[a,b]	154 (407)[a,b]
Energy eff. (μJ/frame)	2.7	2.4	-	3.2 (4.35)	4.05 (3.07)

[a] Area $\sim s^2$
[b] $T_{clk} \sim s$

Section 6.5.3 characterizes the BER performance of the FPTD and RCP-FPTD implementations, when operating in the presence of timing errors caused by different degrees of overclocking.

6.5.1 Timing analysis

As detailed in Section 3.4, the propagation delay of a logic gate depends on the previous and current values of its inputs, as well as its supply voltage and its driving load. Hence, the cumulative propagation delay of a path comprising several logic gates may vary between consecutive clock cycles. One section of each of FX FPTD comprises $q_c \times 2 + q_e \times 15 = 98$ input bits, when $\{q_c, q_e\} = \{4, 6\}$. As a result of this, there are $2^{2\times 98} \approx 10^{59}$ combinations of current and previous input values in a block of each FPTD. Owing to this, it is not feasible to perform a timing analysis based on all possible combinations of inputs. Instead, we propose to perform a Monte Carlo simulation to determine the propagation delay distribution of the various paths in a block of the

FX FPTD and of the FX RCP-FPTD, when using TSMC 40 nm technology and a nominal supply voltage of $V_{DD} = 1.0$ V. We present results obtained from the physical layout generated by the automatic place and route of a block of the FPTD and of the RCP-FPTD using Cadence SoC Encounter [72]. In this analysis, we determine the propagation delay distribution for each of the $q_e \times 7$ bits comprising the output state metrics $\boldsymbol{\alpha}_k = [\alpha_k(s)]_{s=1}^{7}$, the $q_e \times 7$ bits of the output $\boldsymbol{\beta}_{k-1} = [\beta_{k-1}(s')]_{s'=1}^{7}$, as well as for the q_e output bits of the *extrinsic* LLR $b_{1,k}^{e}$ of one block of the FPTD and RCP-FPTD. We also present the propagation delay distribution of the $(q_e + 1) \times 2$ bits of the intermediate results γ_k and the $(q_e + 1) \times 4$ bits of the intermediate results $\varepsilon_{k,m,n}$ for the case of the RCP-FPTD. In this way, our analysis considers the propagation delay of all 90 output bits of the FPTD, as well as the 90 output bits and the 42 internal bits of the RCP-FPTD. These delay distributions were obtained by performing a digital simulation using the extracted parasitics and annotated delays of the physical layouts using Cadence SimVision [72]. More specifically, different random input patterns were provided to one block of each FPTD and the propagation delay generated by each of the input patterns was quantified. To elaborate further, in each clock cycle, each of the 98 input bits of a block of each FPTD adopts an equiprobable bit value. This is justified since the delays using input patterns recorded during the Matlab simulations of the FPTDs' iterative decoding process at different SNR ranges were found to have a similar distribution to that of the equiprobable random bit patterns. Moreover, the distributions obtained using random inputs may be employed for all SNR ranges and for all frame lengths of the FPTDs, which significantly reduces the overall complexity of the timing analysis. For the purpose of our analysis, we employed 10^6 input patterns and a clock period of $T_{clk} = 100$ ns for allowing sufficient time for all the signals to correctly propagate, hence preventing the occurrence of timing violations during this characterization.

The results of our Monte Carlo simulations were used for obtaining a Complementary Cumulative Distribution Function (CCDF) of the propagation delay for each of the 90 output bits of each block of the FPTD and RCP-FPTD, as well as for the 42 internal bits of the RCP-FPTD. As an example of this, Figure 6.20 shows the CCDFs corresponding to the delays of the MSBs of $\alpha_k(1)$, $\beta_{k-1}(1)$ and $b_{1,k}^{e}$ for the FPTD, as well as of the MSBs of $\alpha_k(1)$, $\beta_{k-1}(1)$, $b_{1,k}^{e}$, $\varepsilon_{k,0,1}$ and γ_k and the LSB of γ_k for the RCP-FPTD, in the case when $\gamma_k = b_{1,k} + b_{2,k} + b_{3,k}$.

Similar CCDFs were obtained for the MSBs of all other state metrics $[\alpha_k(s)]_{s=2}^{7}$ and $[\beta_{k+1}(s')]_{s'=2}^{7}$ in the FPTD and RCP-FPTD. Moreover, similar CCDFs were obtained for all the other bits of these state metrics. In addition to this, similar CCDFs were obtained for all other bits of the extrinsic LLR $b_{1,k}^{e}$ and for all other bits of the intermediate variables $\varepsilon_{k,m,n}$. The similarity of these CCDFs may be attributed to the operation of the 'max' and 'sat' blocks employed in the FPTD and RCP-FPTD. To elaborate further, a max block employs a FX comparator and Multiplexers (MUXs) for determining the

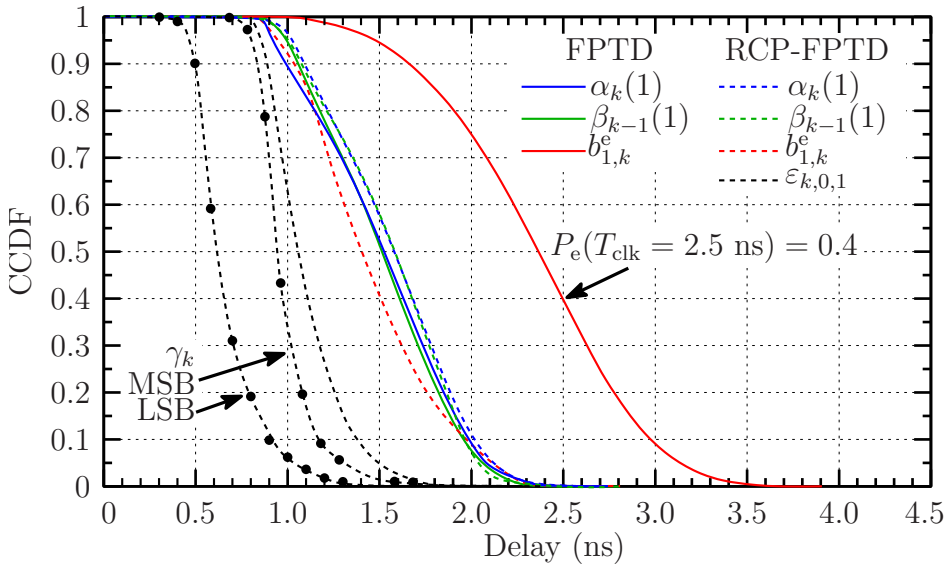


Figure 6.20: Example of CCDFs of propagation delays for the FPTD and RCP-FPTD. These CCDFs were obtained for the case of the MSB of $\alpha_k(1)$, $\beta_{k-1}(1)$ and $b^e_{1,k}$ for the FPTD, as well as of the MSB of $\alpha_k(1)$, $\beta_{k-1}(1)$, $b^e_{1,k}$, $\varepsilon_{k,0,1}$ and γ_k the LSB of γ_k for the RCP-FPTD, in the case when $\gamma_k = b_{1,k} + b_{2,k} + b_{3,k}$.

maximum of two LLRs, as shown in Figure 6.21. Here, all q_e output bits of the MUX are updated according to the bit value of the A > B signal, which is provided by the comparator. As a result of this, all the bits of the output LLRs may exhibit a similar propagation delay. A similar structure to that of Figure 6.21 is employed for performing the saturation operation of the 'sat' blocks.

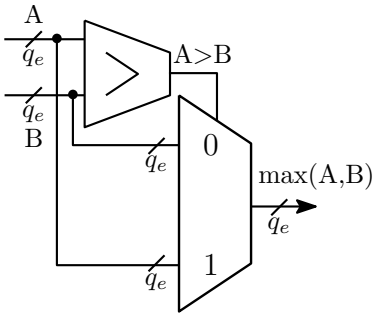


Figure 6.21: Structure of a max block.

Note that the maximum propagation delays reported by the place and route tool are 4.5 ns and 3.0 ns for the FPTD and RCP-FPTD, respectively. However, these propagation delays were not observed in our Monte Carlo simulations. Instead, the maximum propagation delays obtained from our simulations are 4.0 ns and 2.6 ns for the FPTD and RCP-FPTD, respectively.

Figure 6.20 demonstrates that the bits of the *extrinsic* LLRs in the FPTD exhibit larger propagation delays, when compared to the propagation delays of the bits of the state metrics $\alpha_k(s)$ and $\beta_{k-1}(s')$. This may be attributed to the three max operations involved

in the six-stage data path of the *extrinsic* LLRs, compared to only one max operation involved in the six-stage data path of $\alpha_k(s)$ and $\beta_{k-1}(s')$. Moreover, Figure 6.20 demonstrates that the *extrinsic* LLRs, as well as $\alpha_k(s)$ and $\beta_{k-1}(s')$ of the RCP-FPTD exhibit propagation delays similar to those of the state metrics of the FPTD, owing to the reduced critical path consisting of only three data path stages. In addition to this, the propagation delays of the intermediate results $\varepsilon_{k,m,n}$ and γ_k of the RCP-FPTD are significantly smaller, compared to all other propagation delays of Figure 6.20.

6.5.2 Timing error model

The CCDFs obtained in Section 6.5.1 may be used in order to model the occurrence of timing errors in the FPTD and RCP-FPTD, when operated at a fixed clock period T_{clk}, as shown in Figure 6.20. More specifically, during a BER simulation, a timing error may be imposed upon a particular bit if $P_{\text{e}}(T_{\text{clk}}) > R(\text{DC})$, where $P_{\text{e}}(T_{\text{clk}})$ is the probability of an error occurring for that bit at the clock period T_{clk} and $R(\text{DC}) \in [0, 1]$ is a uniform random number generated in the current DC. In each DC of the BER simulations, random numbers are compared to the probabilities of error obtained from the CCDF, in order to simulate the occurrence of timing errors in each bit of each FPTD. To elaborate further, each block of the FPTD employs its 90 CCDFs and 90 random numbers in each DC to determine the occurrence of timing errors. Similarly, each block of the RCP-FPTD employs its 132 CCDFs and 132 random numbers in each DC. Note that owing to the odd-even operation of each FPTD, the CCDFs and the random numbers are employed only in those blocks of each FPTD that are operated in each DC. Owing to this error model, some bits of each FPTD will experience timing errors in some clock cycles, but not in others. Similarly, a large overall number of timing errors will occur in some clock cycles and only a small number of timing errors will be encountered in others. When a timing error is encountered in a particular bit, our error model assumes that a random bit value will be clocked into the corresponding DFF of the affected register and that this bit value will be propagated to the subsequent stages of the decoder.

In addition to this, the CCDFs may be employed for estimating the propagation delay of each signal path t_p in each FPTD and for determining the causes and effects of timing errors, as shown in Figure 6.22 and Table 6.4, respectively. The particular example of Figure 6.22 shows two possible propagation delays. The effect of each delay of Figure 6.22 is summarized in Table 6.4 as follows.

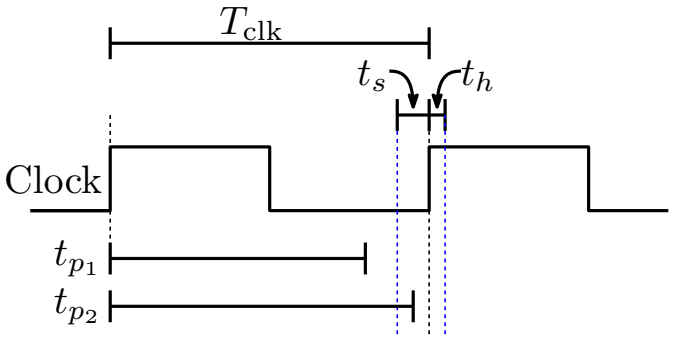


Figure 6.22: Example of propagation delays in the FPTDs.

Table 6.4: Causes and effects of timing errors in the FPTDs.

Scenario	Propagation delay	Effect
1	$t_p \leq T_{\text{clk}} - t_s$	No error
2	$t_p > T_{\text{clk}} - t_s$	Timing error

Scenario 1.

Cause: The signal p_1 arrives before the required setup time t_s of the DFF at the end of the combinational path.

Effect: No timing error occurs, since the signal is correctly propagated in the time allotted. The DFF will propagate the correct bit value to the subsequent stages.

Scenario 2.

Cause: The signal p_2 arrives after the required setup time t_s of the DFF at the end of the combinational path.

Effect: A timing error occurs. The DFF will propagate a random bit value to the subsequent stages.

Note that propagation delay $T_{\text{clk}} \leq t_p \leq T_{\text{clk}} + t_h$ results in a hold time violation of the DFF. However, this is equivalent to the situation described in *Scenario 2*, owing to $t_p > T_{\text{clk}} - t_s$. Similarly, a timing error occurring during the interval $T_{\text{clk}} - t_s < t_p < T_{\text{clk}} + t_h$ may trigger the occurrence of metastability in the affected DFF, which may propagate to the rest of the circuit, as detailed in Chapter 4. For the purpose of our investigations in this chapter, we assume that the metastability event is resolved to a defined, but random logic state, for the sake of simplifying our timing error model.

As a result of this error model, the effect of the timing error is determined by the significance of each affected bit. More specifically, an error in the MSB of an LLR

results in an absolute error of 2^{q_e-1}. However, an LSB error results in a maximum absolute error of $2^0 = 1$. In the example of Figure 6.20, a timing error will occur on the MSB of the *extrinsic* LLR $b^e_{1,k}$ of the FPTD with probability of $P_{\text{error}} = 0.4$, when the clock period is set to $T_{\text{clk}} = 2.5$ ns. By contrast, timing errors in the state metrics $\alpha_k(s)$ and $\beta_{k-1}(s')$ of the FPTD will only occur if the clock period is set below 2.5 ns.

6.5.3 Error decoding performance

Figure 6.23 shows the BER performance of the FX FPTD and FX RCP-FPTD employing the timing error model described in Section 6.5.2. These results correspond to the cases of transmitting $N = \{40, 6144\}$ bits using BPSK modulation over an AWGN channel, when using different degrees of overclocking.

Figure 6.23 compares the BER obtained at the maximum number of DCs required by each FPTD for achieving iterative decoding convergence in the absence of timing errors, as previously shown in Figure 6.19. More specifically, we employ a maximum of 48 DCs for both the FPTD and the RCP-FPTD, when $N = 40$. Similarly, we employ a maximum of 80 and 100 DCs for the FPTD and RCP-FPTD, respectively, when $N = 6144$. According to the simulation results of Figure 6.23, the FPTD operated at the clock periods of 4.5 ns and 3.4 ns exhibit similar BER performance, when $N = 40$. Similarly, the clock period of the FPTD having a frame length of 6144 may be reduced to 3.5 ns, without imposing a significant BER degradation. Moreover, the overclocked RCP-FPTDs operating at 2.3 ns and 2.4 ns achieve similar BER performances to those of the RCP-FPTDs operated at 3.0 ns, when the frame length is 40 and 6144, respectively.

The employment of clock periods moderately lower than the critical path may be expected to enhance the latency, throughput and energy efficiency of the decoders, without significantly affecting their error correction capability, as we will show in Section 6.6.2.3. Similarly, the BER performance of the various overclocked FPTDs of Figure 6.23 may be improved by increasing the number of DCs beyond the maximum suggested limits. However, this degrades the hardware efficiency of the decoders, as we will also demonstrate in Section 6.6.2.3.

The results presented in this section suggest that the FPTD and RCP-FPTD implementations of Sections 6.2.2 and 6.3.2 have an inherent, but only partial, tolerance to timing errors. Section 6.6 details how the inherent tolerance to errors of the FPTDs may be further enhanced by applying error-tolerant design techniques.

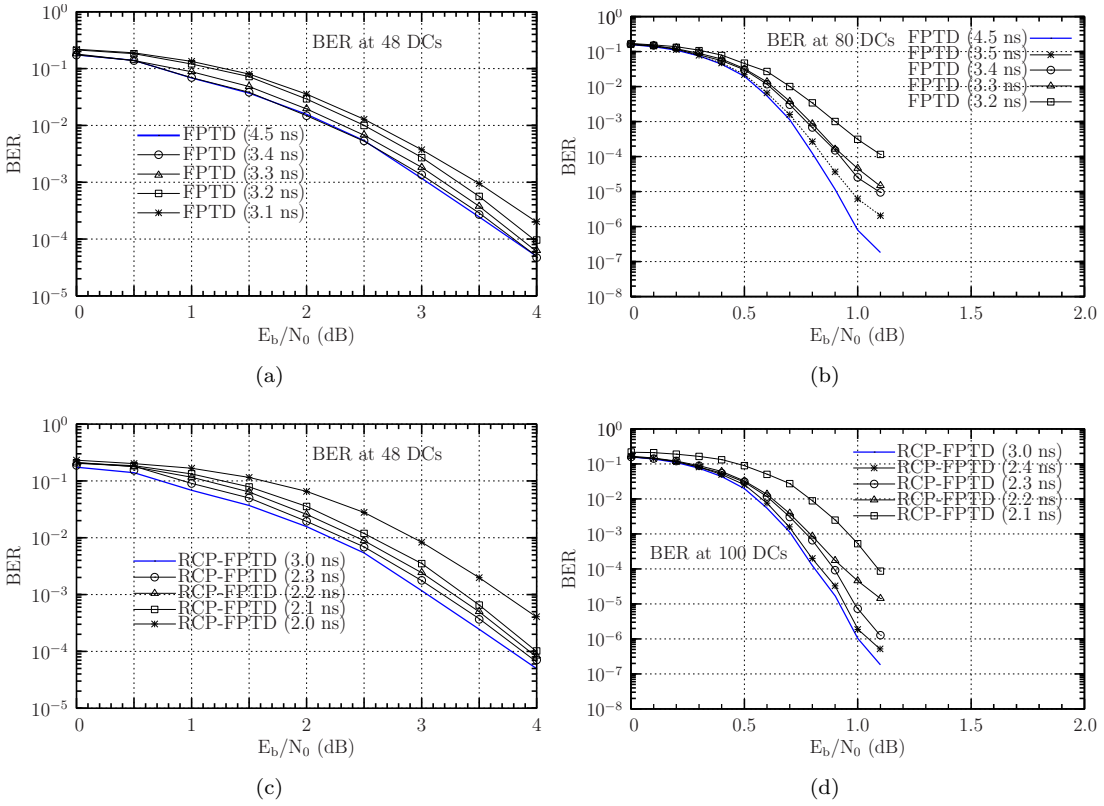


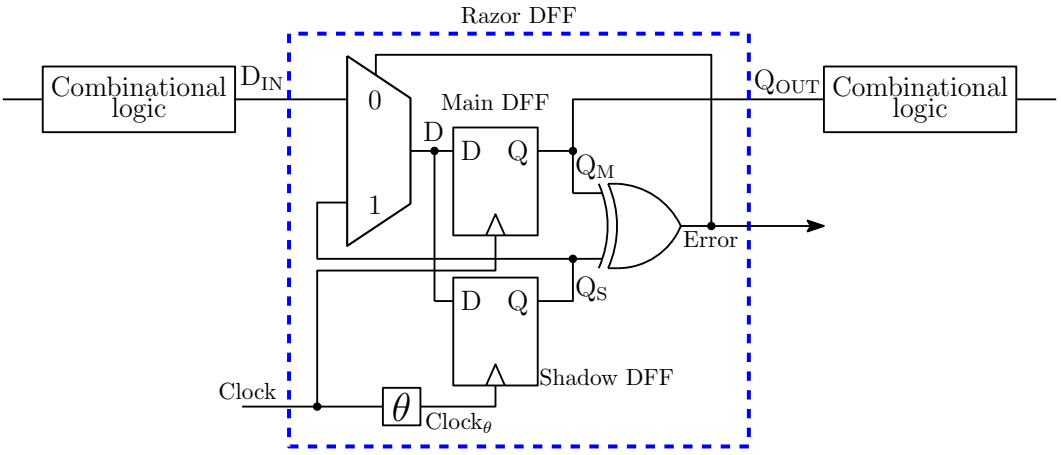
Figure 6.23: BER performance of the FPTD and RCP-FPTD in the presence of timing errors owing to different degrees of overclocking, when transmitting N={40,6144} bits using BPSK over an AWGN channel. (a) BER of the FPTD, when N=40, DC=48 and T_{clk} ={4.5, 3.4, 3.3, 3.2, 3.1} ns. (b) BER of the FPTD, when N=6144, DC=80 and T_{clk} ={4.5, 3.5, 3.4, 3.3, 3.2} ns. (c) BER of the RCP-FPTD, when N=40, DC=48 and T_{clk} ={3.0, 2.3, 2.2, 2.1, 2.0} ns. (d) BER of the RCP-FPTD, when N=6144, DC=100 and T_{clk} ={3.0, 2.4, 2.3, 2.2, 2.1} ns.

6.6 Better-than-worst-case Design in FPTDs

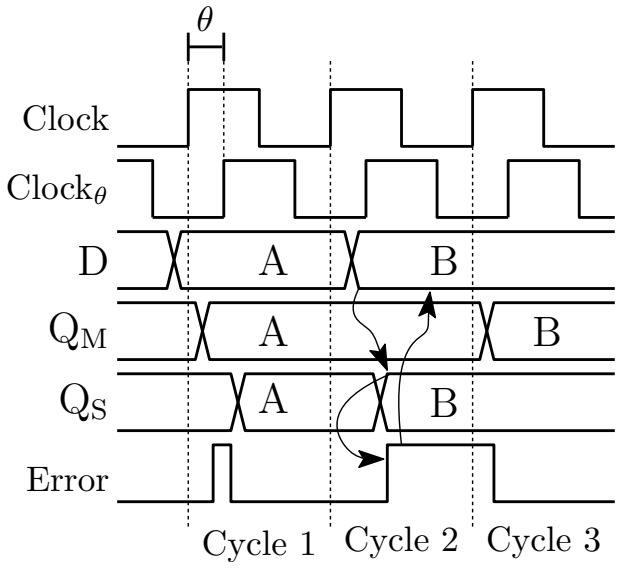
The results of Section 6.5.3 suggest that the BER performance of the FPTD and RCP-FPTD is not significantly degraded, when timing errors occur owing to the employment of a clock period smaller than the critical path. Motivated by this, we propose the employment of EDAC techniques [13, 20, 21, 22] for mitigating the effect of timing errors in the FPTD and RCP-FPTD in overclocking scenarios, for the sake of improving both their BER performance and their hardware efficiency. This methodology is commonly referred to as BTWC design [19], since the operating conditions of the system are relaxed below the recommended safety margins and the occasional occurrence of timing errors is compensated by the employment of EDAC techniques.

6.6.1 Razor DFF

Figure 6.24 presents the structure of a Razor D-Type Flip Flop (RDFF) [13] and a timing diagram showing its operation. An RDFF consists of one MUX, one main DFF, one shadow DFF and one XOR gate, as shown in the blue box of Figure 6.24(a). The operation of the RDFF is exemplified in Figure 6.24(b). In the RDFF, both the main DFF and the shadow DFF are provided with the same input D. However, the clock signal of the shadow DFF is delayed by a time θ, with respect to the clock signal of the main DFF, as shown in the top part of Figure 6.24(b).



(a)



(b)

Figure 6.24: Razor DFF. (a) Razor circuit. (b) Timing diagram showing the operation of a Razor DFF.

The operation of the RDFF is summarized as follows. In Cycle 1 of Figure 6.24(b), the input data A arrives before the first rising edge of the Clock signal, whereupon the output Q_M of the main DFF updates its value with A. Following the rising edge of the

delayed signal Clock$_\theta$, the output Q_S of the shadow DFF updates its value with A. In Cycle 2 of Figure 6.24(b), a timing error results in the input data B arriving after the rising edge of Clock but before the rising edge of Clock$_\theta$. In this case, the output Q_M maintains the value of A from the previous clock cycle, while Q_S is updated with B. As a result of this, the output Error of the XOR gate of Figure 6.24(a) adopts the value of 1 for indicating that a timing error has been detected in Cycle 2. In Cycle 3, the output Q_M is corrected with B. This is achieved with the aid of the MUX of Figure 6.24(a), which provides the value of B to the input D of both DFFs, owing to Error=1 driving the selector signal of the MUX. In this way, the penalty for detecting and correcting an error is only one clock cycle. Note that the Error signal exhibits a glitch in Cycle 1, owing to the outputs Q_M and Q_S updating their values at different instants. Despite this, the operation of the RDFF is not affected, since the Error signal is cleared after the rising edge of Clock$_\theta$, when $Q_M = Q_S$. Moreover, the locally generated Error signal may be employed by a global control unit in order to halt the operation of the system and avoid the propagation of unreliable computations.

The delay θ is usually referred to as the error-detection window, since this is the period in which late transitions of D may be correctly detected. In this way, a late transition of D occurring after θ will not be detected by the RDFFs. Note that effectively, the detection window θ corresponds to delay between the two clock signals minus the required setup time of the shadow DFF. However, the error-detection window may also detect changes in D owing to short combinational paths, when the RDFF is required to sample input data in consecutive clock cycles. This is referred to as the short-path problem, which occurs when the propagation delay of a path is smaller than the delay θ. In this case, the RDFFs may trigger a false positive error detection. In order to avoid this situation, additional timing constraints must be considered during the design phase of the system. This results in a significant design complexity increase, owing to the careful insertion of buffers in the short paths of the system for extending their propagation delay above the value of θ [13]. Moreover, this solution increases the area requirements and energy consumption of the system. Alternatively, the delay θ may be reduced below the propagation delay of the short paths, hence avoiding the insertion of buffers, albeit at the cost of reducing the detection window of the RDFFs. Note that the occurrence of timing errors in the interval comprising the required setup and hold times of the main DFF of the RDFF may trigger the occurrence of metastability. Similarly, metastability may occur in the shadow DFF if the input D changes during the interval comprising its required setup and hold times. In order to prevent the catastrophic propagation of metastability through the circuit, the authors of [13, 20, 21, 22] have proposed the employment of metastability detector circuits within the RDFF. However, these techniques rely on determining the optimal dimensions of Complementary Metal-Oxide Semiconductor (CMOS) transistors, in order to detect the occurrence of metastability, resulting in a significant increase in the complexity of the implementation of RDFFs. For the purpose of our investigations, we assume that the occurrence of metastability

does not affect the operation of the RDFF, in order to reduce the complexity of the implementation.

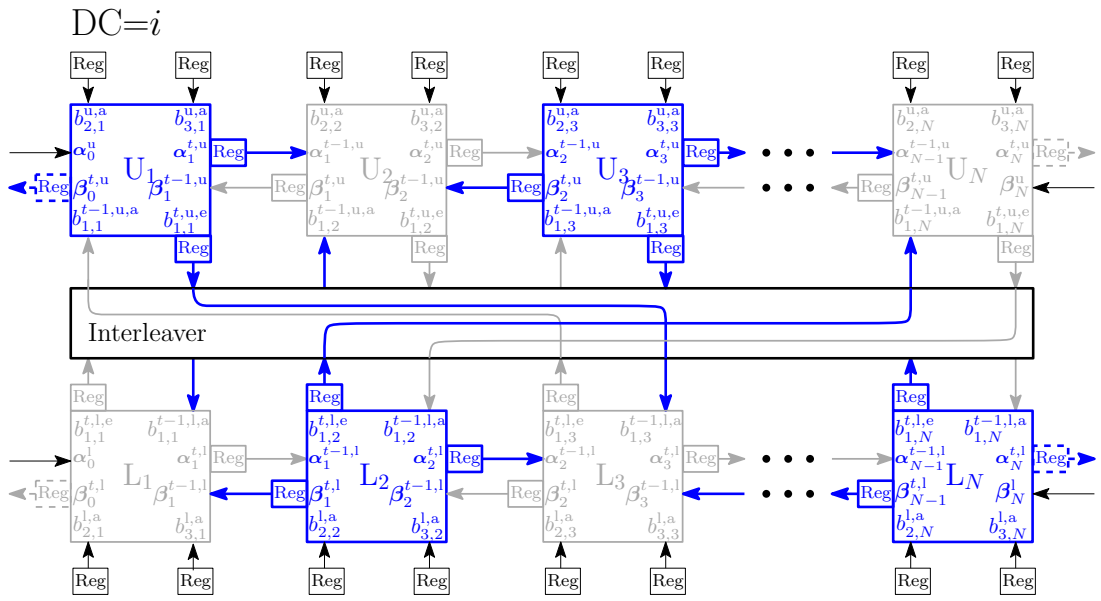
Sections 6.6.2 and 6.6.3 detail how the RDFFs may be employed in the FPTD and RCP-FPTD, respectively, for improving their tolerance to timing errors, as well as their BER performance and hardware efficiency. Furthermore, Sections 6.6.2 and 6.6.3 describe how the odd-even operation of the FPTDs and RCP-FPTD effectively overcomes the short path problem in the RDFFs described above.

6.6.2 BTWC-FPTD

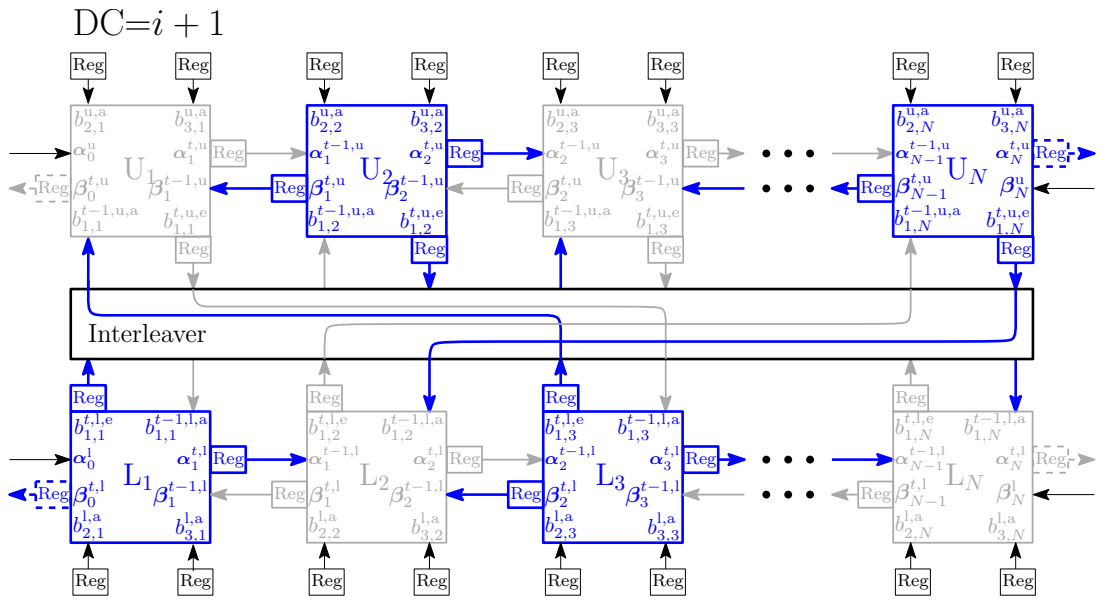
The following sections detail the employment of RDFFs in the FPTD of Section 6.2, for the sake of improving its tolerance to timing errors and its hardware efficiency. We refer to this implementation as the Better-Than-Worst-Case Fully-Parallel Turbo Decoder (BTWC-FPTD). Section 6.6.2.1 details the hardware implementation of the BTWC-FPTD. Following this, Section 6.6.2.2 presents a timing analysis and a timing error model of the BTWC-FPTD. This timing error model is employed in Section 6.6.2.3 for characterizing the different trade-offs associated with the hardware implementation of the BTWC-FPTD in the presence of timing errors owing to overclocking.

6.6.2.1 Hardware Implementation

As described in Section 6.2, the LTE FPTD operates using an odd-even scheduling, where only one group of blocks of the FPTD is operated in each DC. This is exemplified in Figure 6.25. In the i^{th} DC of Figure 6.25(a), only the blocks of the upper row having odd indexes and the blocks of the lower row having even indexes are operated. Following this, only the blocks having even indexes in the upper row and the blocks having odd indexes in the lower row are operated in the $(i+1)^{th}$ DC, as shown in Figure 6.25(b). This operation guarantees that each block of the FPTD is not operated in two consecutive DCs. To elaborate further, let us analyze in more detail the operation of the block U_1 of Figure 6.25, as a particular example. In the i^{th} DC of Figure 6.25(a), the block U_1 provides the forward state metrics $\boldsymbol{\alpha}_1^{t,u}$ to its neighboring block U_2. Likewise, U_1 will provide the *extrinsic* LLR $b_{1,1}^{t,u,e}$ as the *a priori* LLR $b_{1,3}^{t,l,a}$ to L_3 in the same i^{th} DC. After this, in the $(i+1)^{th}$ DC of Figure 6.25(b), U_2 and L_3 will employ their inputs provided by U_1 and their other neighboring blocks for generating the outputs $\boldsymbol{\alpha}_2^{t,u}$, $\boldsymbol{\beta}_1^{t,u}$, $b_{1,2}^{t,u,e}$ for U_2, as well as $\boldsymbol{\alpha}_3^{t,l}$, $\boldsymbol{\beta}_2^{t,l}$ and $b_{1,3}^{t,l,e}$ for L_3. However, U_1 is not operated in the $(i+1)^{th}$ DC. Hence the outputs of U_1 maintain the values that they had in the i^{th} DC. This operation eliminates the short-path problem of the RDFF described in Section 6.6.1, since the outputs of the blocks of the FPTD do not update their value in consecutive clock cycles. As an explicit benefit of this, it is possible to eliminate the short-path constraint of the RDFF of Figure 6.24 for the implementation of the



(a)



(b)

Figure 6.25: Odd-even operation of the FPTD. Only those blocks highlighted in blue lines are operated in each DC.(a) Odd blocks of the upper row and even blocks of the lower row are operated. (b) Even blocks of the upper row and odd blocks of the lower row are operated.

BTWC-FPTD. In this way, the operation of the BTWC-FPTD guarantees that the data sampled during the detection window θ corresponds to the data generated in the previous $(i-1)^{th}$ DC, rather than data generated in the current i^{th} DC owing to a short combinational path.

Figure 6.26 shows how the RDFF of Figure 6.24 may be modified for eliminating the short-path constraint in the BTWC-FPTD. Here, the shadow DFF of Figure 6.24 is

replaced by a shadow D-type latch and the delay block θ is eliminated. In this configuration, the output Q_M of the main DFF will take the value of the input D at the rising edge of the clock. By contrast, the output Q_S of the shadow D-type latch will take the value of the input D, when the clock signal takes the value of 1. The output Error of the XOR gate will take the value of 1, if $Q_M \neq Q_S$, to indicate that a timing error has been encountered, in analogy to the behavior of the RDFF of Figure 6.24. As a result of this, the detection window θ of this configuration corresponds to the time that the clock signal has the value of 1, minus the required setup time of the shadow D-type latch. Moreover, note that the MUX of Figure 6.24 is no longer needed in the BTWC-FPTD, since the output Q_M of the main DFF does not need to be updated with the correct value, if a timing error is encountered. This is justified since an erroneous value of Q_M in the i^{th} DC will prevent the propagation of unreliable data, as will be described later in this section.

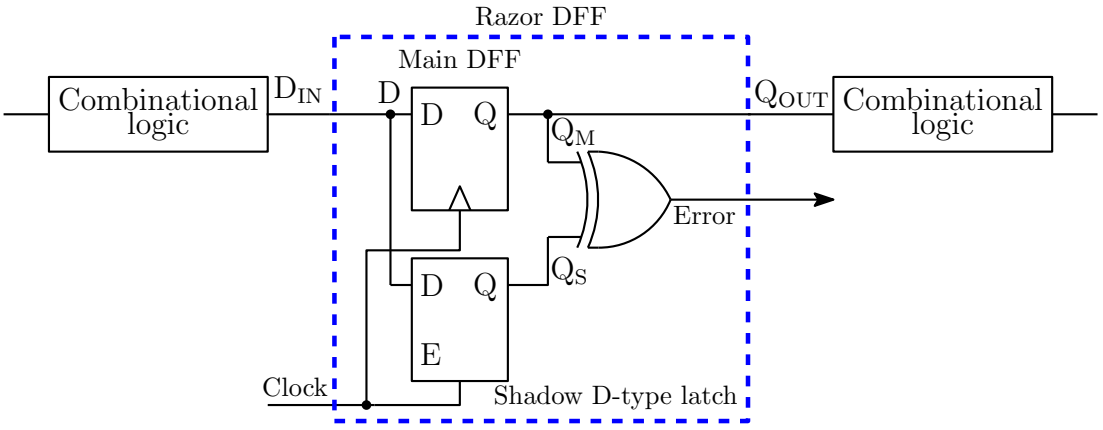


Figure 6.26: Razor DFF employed in the BTWC-FPTD and BTWC-RCP-FPTD.

In the BTWC-FPTD, we replace some of the conventional DFFs with the latch-based RDFFs of Figure 6.26. More specifically, we replace only the DFFs of the MSBs of the *extrinsic* LLR $b^e_{1,k}$ with RDFFs, as shown in Figure 6.27. The rest of the $(q_e - 1)$ bits of $b^e_{1,k}$, as well as all q_e bits of the state metrics $\alpha_k(s)$ and $\beta_{k-1}(s')$ employ conventional DFFs. As a result of this, only 1 out of the 90 DFFs of each block of the FPTD employ a RDFF. Note however that the BTWC-FPTD requires the interleaving and de-interleaving of the Error signals of each block of the FPTD. This additional Error signal does not affect the timing characteristics or the area efficiency of the BTWC-FPTD, as Sections 6.6.2.3 and 6.7 will demonstrate. This is justified since the CCDFs of the FPTD of Figure 6.20 demonstrate that the *extrinsic* LLRs $b^e_{1,k}$ are more susceptible to timing errors, when compared to the bits of the state metrics, as described in Section 6.5.2. Moreover, we only employ RDFFs in the MSB of $b^e_{1,k}$, since this bit determines the sign of the represented LLR and timing errors in this bit severely degrade the BER performance of iterative decoders [46, 91].

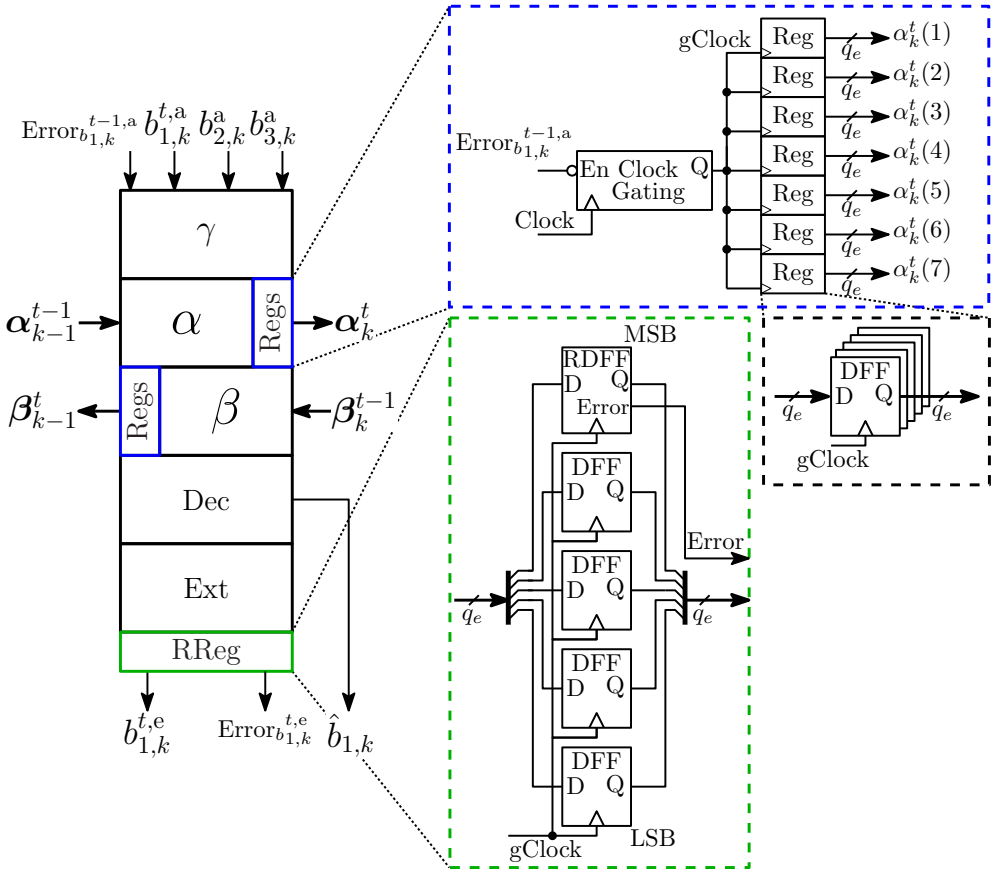


Figure 6.27: Implementation of RDFFs in a block of the BTWC-FPTD.

Figure 6.27 shows how the Error signal of the RDFFs may be employed for preventing the propagation of unreliable LLRs. Here, the DFFs are grouped in the 'Reg' blocks in order to simplify the discussions. In the example of Figure 6.27, the incoming error signal $\text{Error}_{b1,k}^{t-1,a}$ is employed by a clock gating module for disabling the clock signal of all the Regs of a block of the BTWC-FPTD. In this way, if an error is encountered in the *a priori* LLR $b_{1,k}^{a}$, the Regs of the modules α, β, as well as the Razor-based Reg of the module Ext of Figure 6.27, denoted as 'RReg', will maintain the value that they had in the previous DC. In this way, the *a priori* LLR $b_{1,k}^{t,a}$ is effectively discarded during the current DC of the BTWC-FPTD's iterative decoding process.

This error containment scheme avoids the need to provide an update with the correct bit value for the main DFF of the RDFF of $b_{1,k}^{t,a}$, reducing the complexity of the BTWC-FPTD implementation. Instead, the subsequent data path stages are not operated using the unreliable data. This differs from the traditional EDAC strategies presented in the open literature [13, 20, 21, 22], where the state of the system is corrected and restored by repeating those operations that triggered the timing error, after increasing either the clock period or the supply voltage, in order to allow the correct completion of the operation.

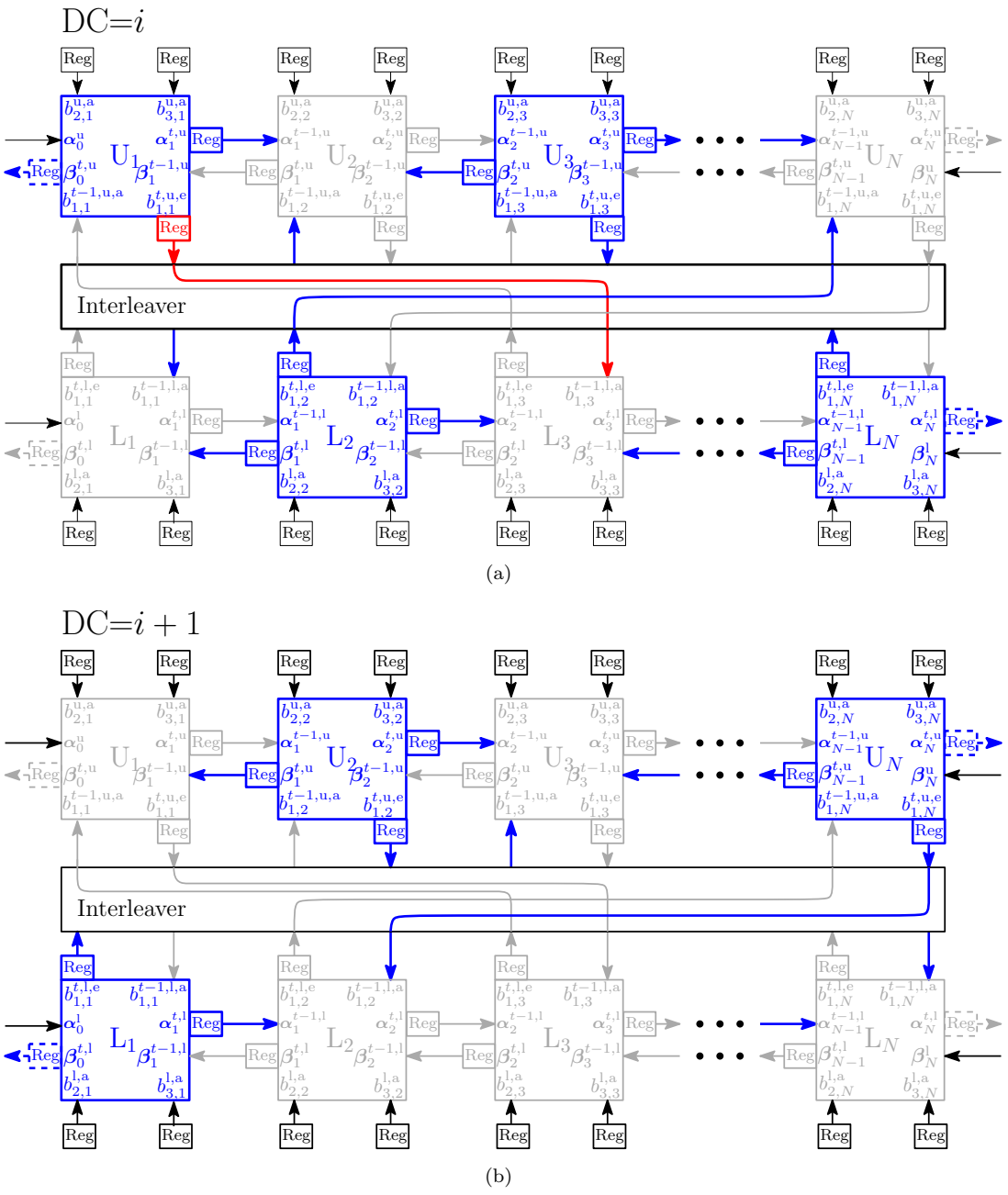


Figure 6.28: Operation of the BTWC-FPTD in the presence of timing errors. (a) A timing error is detected in the MSB of the *extrinsic* LLR of U_1. (b) L_3 is not operated owing to the timing error detected in U_1.

Figure 6.28 shows how the odd-even operation of the FPTD is no longer guaranteed in the BTWC-FPTD. Here, the red line represents a timing error, that is encountered in $b_{1,1}^{u,e}$ in the block U_1 in the i^{th} DC of Figure 6.28(a). Following this, in the $i+1^{th}$ DC of Figure 6.28(b), the block L_3 is not operated, owing to the detection of the timing error encountered in the previous DC. As a result of this, only a small number of light-shaded blocks of Figure 6.5 may be operated in alternate DCs, but a large number of

light-shaded blocks may be operated in others. Similarly, only a small number of dark-shaded blocks of Figure 6.5 may be operated in alternate DCs, but a large number of dark-shaded blocks may be operated in others. This operation effectively modifies the FPTD algorithm in a dynamic way, resulting in the operation of different algorithms for different received frames, depending on the occurrence of timing errors. However, the BTWC-FPTD exhibits an enhanced tolerance to timing errors and to the dynamic changes in the algorithm, as we will demonstrate in Section 6.6.2.3.

6.6.2.2 Timing analysis and timing error model

In this section, we characterize the timing characteristics of the BTWC-FPTD, in analogy to the timing analysis presented in Section 6.5.1 for the case of the FPTD. We performed a Monte Carlo simulation for determining the CCDFs of each bit of the BTWC-FPTD, when using TSMC 40 nm technology and a nominal supply voltage of V_{DD} = 1.0 V. We present results for the physical layout generated by the automatic place and route of the BTWC-FPTD using Cadence SoC Encounter [72]. According to our post-layout results, the employment of RDFF does not modify the timing characteristics of the FPTD. As a result of this, the critical clock period, as well as the CCDFs of Figure 6.20 obtained for the case of the FPTD remain unchanged, for the case of the BTWC-FPTD design. This allows us to employ the timing error model of Section 6.5.2 for determining the occurrence of timing errors in the BTWC-FPTD. In each DC of the BER simulations, we employ random numbers and the CCDFs of Figure 6.20 for determining the propagation delay of each signal. Following this, we employ Figure 6.22 and Table 6.4 for determining the causes and effects of timing errors for each of the bits of the BTWC-FPTD that are implemented using conventional DFFs, as detailed in Section 6.6.1. However, we employ Figure 6.29 and Table 6.5 for determining the causes and effects of timing errors in each of the bits of the BTWC-FPTD that employ RDFFs.

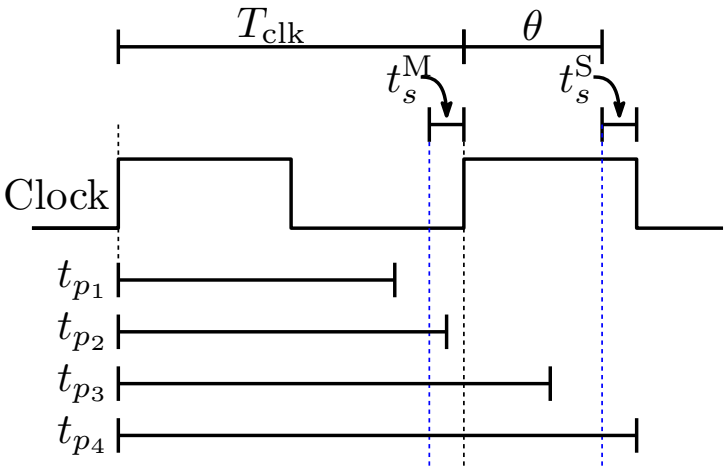


Figure 6.29: Example of propagation delays in the BTWC FPTDs.

Table 6.5: Timing error conditions in BTWC designs.

Scenario	Propagation delay	Effect
1	$t_p \leq T_{\text{clk}} - t_s^{\text{M}}$	No error
2	$T_{\text{clk}} - t_s^{\text{M}} < t_p < T_{\text{clk}}$	Undetected error
3	$T_{\text{clk}} \leq t_p \leq T_{\text{clk}} + \theta$	Detected error
4	$t_p > T_{\text{clk}} + \theta$	Undetected error

Each of the four possible scenarios of Figure 6.29 and Table 6.5 is further detailed as follows.

Scenario 1.

Cause: The signal p_1 arrives before the required setup time t_s^{M} of the main DFF of the RDFF at the end of the combinational path.

Effect: No timing error occurs, since the signal is correctly propagated in the time allotted. The RDFF will propagate the correct bit value to the subsequent stages.

Scenario 2.

Cause: The signal p_2 arrives during the interval comprising the setup time t_s^{M} of the main DFF of the RDFF at the end of the combinational path and the rising edge of the clock.

Effect: A timing error occurs but it is not detected by the RDFF, since it occurs outside the detection window θ. The RDFF will propagate a random bit value to the subsequent stages.

Scenario 3.

Cause: The signal p_3 arrives after the rising edge of the clock but before the required setup time t_s^{S} of the shadow D-type latch of the RDFF.

Effect: A timing error occurs and it is detected by the RDFF, since it occurs during the detection window θ. The Error signal of the RDFF will be asserted, preventing the updating of the subsequent DFFs.

Scenario 4.

Cause: The signal p_4 arrives after the required setup time t_s^{S} of the shadow D-type latch of the RDFF.

Effect: A timing error occurs but it is not detected by the RDFF, since it occurs outside the detection window θ. The RDFF will propagate a random bit value to the subsequent stages.

Note that in analogy to the causes and effects of timing errors of Figure 6.22 and Table 6.4, a hold time violation of the main DFF of the RDFF will result in a timing error. However, in this case, the RDFF will detect this error. By contrast a hold time violation of the shadow D-type latch of the RDFF will result in an undetected error, as described in *Scenario 4*.

For the purpose of our investigations, our timing error model assumes clock signals associated with a 50% duty cycle. As a result of this, the detection window is $\theta = T_{\text{clk}}/2 - t_s^{\text{S}}$, where the typical required setup time of the shadow D-type latch is $t_s^{\text{S}} =$ 100 ps for TSMC 40 nm technology [92].

6.6.2.3 Trade-off analysis in the presence of timing errors

This section characterizes the different trade-offs associated with the BER performance and hardware efficiency of the BTWC-FPTD in the presence of timing errors owing to overclocking. We employ the timing error model described in Section 6.6.2.2 for our BER simulations of the BTWC-FPTD, for the cases when transmitting N={40,6144}-bit frames, using BPSK modulation for communication over an AWGN channel, when using different degrees of overclocking and when allowing a maximum of 256 DCs with early stopping. We also present benchmark results for the FPTD operating at a clock period equal to its critical path delay in the absence of timing errors, as well as at different degrees of overclocking in the presence of timing errors.

Figure 6.30 characterizes the BER performance, average number of DCs performed, throughput and energy efficiency of the FPTD and BTWC-FPTD implementations, for the case when $N = 40$. These results suggest that the BER performance and hardware efficiency of the BTWC-FPTD are improved, when compared to those of the FPTD. As an example of this, Figure 6.30(a) demonstrates that the BER performance of the $N = 40$ BTWC-FPTD operated at 3.3 ns is similar to that of the FPTD operated at the critical clock period of 4.5 ns. Similarly, the BTWC-FPTD operated at 3.3 ns requires a similar number of DCs for achieving iterative decoding convergence to a target BER of 10^{-4}, when compared to the FPTD operated at the critical clock period of 4.5 ns and in the absence of timing errors, as shown in Figure 6.30(b). As a result of this, the BTWC-FPTD operated at 3.3 ns exhibits the largest throughput and lowest energy consumption per decoding frame of the presented schemes, as shown in Figures 6.30(c) and 6.30(d), respectively. More specifically, the BTWC-FPTD operated at 3.3 ns offers 1.18 and 1.33 times improvements to throughput and energy, when compared to those of the FPTD operated at 3.3 ns and 4.5 ns, respectively. This is achieved despite the occurrence of timing errors in the BTWC-FPTD.

Furthermore, the results of Figure 6.31 suggest that the BTWC-FPTD significantly enhances both the BER performance and the hardware efficiency of the FPTD in the

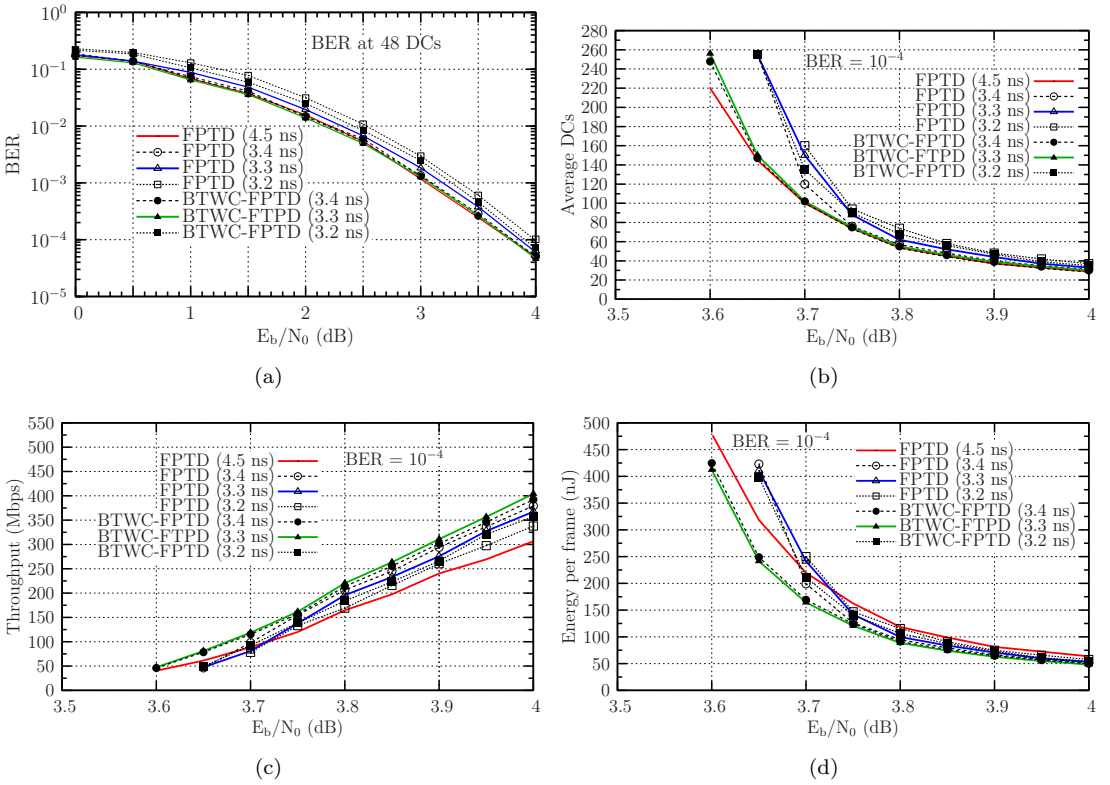


Figure 6.30: Trade-off analysis of the FPTD and of the BTWC-FTPD in the presence of timing errors owing to different degrees of overclocking, when transmitting $N = 40$ bits using BPSK over an AWGN channel. The results of average DCs, throughput and energy efficiency were obtained for the case of a target BER of 10^{-4} and when using a maximum of 256 DCs with early stopping. (a) BER performance at 48 DCs. (b) Average number of DCs for achieving a BER of 10^{-4}. (c) Throughput. (d) Energy per decoding frame.

presence of timing errors, when $N = 6144$. Here, the BER performance may be improved by an order of magnitude by employing the BTWC-FPTD, when the clock period is set to 3.3 ns. Similarly, the BTWC-FPTD improves the throughput and energy efficiency of the FPTD, despite the occurrence of timing errors. As an example of this, the throughput of the BTWC-FPTD operated at 3.3 ns and an $E_b/N_0 = 0.8$ dB for achieving a BER of 10^{-5} is 2.07 and 1.18 greater than those of the FPTD operated at the same SNR value and 3.3 ns and 4.5 ns, respectively.

Figures 6.30 and 6.31 demonstrate that the BTWC-FPTD enhances the tolerance to timing errors of the FPTD. This is particularly observed in Figure 6.31, where the FPTD operated at the critical clock period and in the absence of timing errors outperforms the FPTD operated at overclocked periods. However, the FPTD operated at the critical clock period does not outperform the BTWC-FPTD operated at an overclocked period of 3.3 ns. As a result of this, we propose the employment of the BTWC-FPTD operated at a clock period of 3.3 ns.

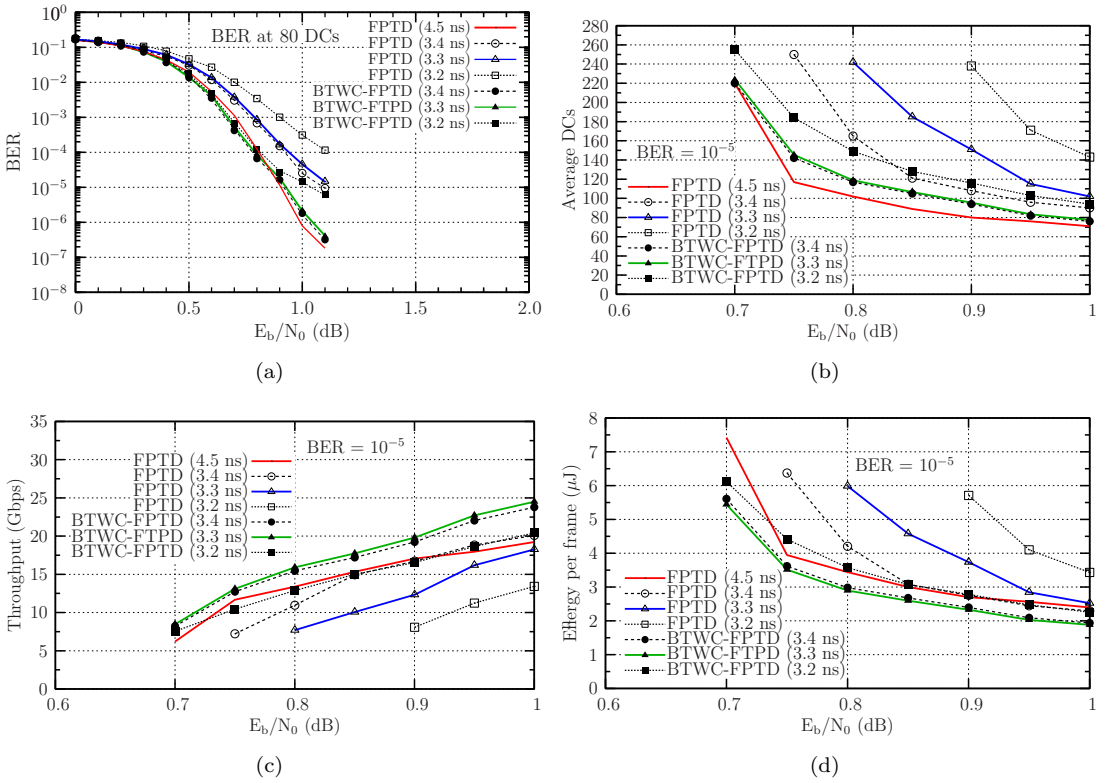


Figure 6.31: Trade-off analysis of the FPTD and of the BTWC-FTPD in the presence of timing errors owing to different degrees of overclocking, when transmitting $N = 6144$ bits using BPSK over an AWGN channel. The results of average DCs, throughput and energy efficiency were obtained for the case of a target BER of 10^{-5} and when using a maximum of 256 DCs with early stopping. (a) BER performance at 80 DCs. (b) Average number of DCs for achieving a BER of 10^{-5}. (c) Throughput. (d) Energy per decoding frame.

Motivated by these results, Section 6.6.3 details the employment of the BTWC design techniques in the RCP-FPTD of Section 6.3.

6.6.3 BTWC-RCP-FPTD

In analogy to Section 6.6.2, the following sections detail the employment of RDFFs in the RCP-FPTD described in Section 6.3 for the sake of improving its tolerance to timing errors and its hardware efficiency. We refer to this implementation as Better-Than-Worst-Case Reduced-Critical-Path Fully-Parallel Turbo Decoder (BTWC-RCP-FPTD). Section 6.6.3.1 details the hardware implementation of the BTWC-RCP-FPTD. Following this, Section 6.6.3.2 presents a timing analysis and a timing error model of the BTWC-RCP-FPTD. This timing error model is employed in Section 6.6.3.3 for characterizing the different trade-offs associated with the hardware implementation of the BTWC-FPTD in the presence of timing errors owing to overclocking.

6.6.3.1 Hardware Implementation

In analogy to the behavior of the FPTD, the RCP-FPTD operates using an odd-even scheduling, as described in Section 6.3. However, in the RCP-FPTD, only specific sections of each block are operated. This is exemplified in Figure 6.32. Here, the lower part of U_1 provides the *extrinsic* LLR $b_{1,1}^{u,e}$ as the *a priori* LLR $b_{1,3}^{l,a}$ to L_3 in the i^{th} DC of Figure 6.32(a). In the same i^{th} DC, the middle part of U_2 provides $\boldsymbol{\alpha}_2^{u}$ and $\boldsymbol{\beta}_1^{u}$ to U_3 and U_1, respectively. However, in $(i+1)^{th}$ DC of Figure 6.32(b), the top and bottom parts of U_1 are not operated. Similarly, the middle part of U_2 is not operated in the $(i+1)^{th}$ DC. This scheduling avoids the operation of the same section of each block of the RCP-FPTD in two consecutive DCs. As an explicit benefit of this, the BTWC-RCP-FPTD may employ the latch-based RDFF of Figure 6.26, rather than the short-path constrained RDFF of Figure 6.24(a).

Figures 6.33 and 6.34 show how RDFFs may be employed in the RCP-FPTD. In Figure 6.33, RDFF are employed by the MSB of the state metrics $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$, as well as by the MSB of the *extrinsic* LLR $b_{1,k}^{e}$. By contrast, the γ, ε and Dec modules employ conventional DFFs. This is justified since the CCDFs of the RCP-FPTD of Figure 6.20 demonstrate that $\boldsymbol{\alpha}_k$, $\boldsymbol{\beta}_k$ and $b_{1,k}^{e}$ are more susceptible to the occurrence of timing errors in the RCP-FPTD. As a consequence of the employment of RDFFs in the α module of Figure 6.27, this module is required to provide the error signal $\text{Error}_{\alpha_k}^{t}$ to the $(k+1)^{th}$ neighboring block. This signal is the result of the logical OR of the individual error signals generated by each RDFF of $\boldsymbol{\alpha}_k$, as shown in the blue box of Figure 6.27. Owing to this, the k^{th} block of the BTWC-RCP-FPTD prevents the module α from operating, if a timing error has been detected in the module α of the $(k-1)^{th}$ block in the $(i-1)^{th}$ DC. This is achieved by the clock gating unit of the α module shown in Figure 6.27. Likewise, the module β of Figure 6.27 employs a structure similar to that of the blue box for providing the error signal $\text{Error}_{\beta_{(k-1)}}^{t}$ to the $(k-1)^{th}$ neighboring block, which corresponds to the logical OR of the individual error signals generated by each RDFF of $\boldsymbol{\beta}_{k-1}$. As a result of this, the k^{th} block of the BTWC-RCP-FPTD prevents the module β from operating by using a clock gating unit in the β module, if a timing error has been detected in the module β of the $(k+1)^{th}$ block in the $(i-1)^{th}$ DC. Finally, the module ε performs the logical OR operation of the incoming error signals $\text{Error}_{\alpha_{(k-1)}}^{t-1}$ and $\text{Error}_{\beta_k}^{t-1}$, in order to determine the occurrence of errors in the α module of the $(k-1)^{th}$ block or in the β module of the $(k+1)^{th}$ block. When a timing error is encountered in a neighboring block owing to $\text{Error}_{\alpha_{(k-1)}}^{t-1}$ or $\text{Error}_{\beta_k}^{t-1}$, a clock gating unit in the ε module disables the clock signal Clock_{middle} for preventing the operation of its DFFs. Note that the clock gating of the ε module is not explicitly shown in Figure 6.27 for the sake of simplifying the schematic diagrams. In addition to this, the incoming error signal $\text{Error}_{b1,k}^{t-1,a}$ prevents the updating of the Dec module. In this configuration, each of the four middle modules of Figure 6.27 employ a clock gating unit in order to prevent their DFFs from being updated with unreliable data. Moreover, the incoming

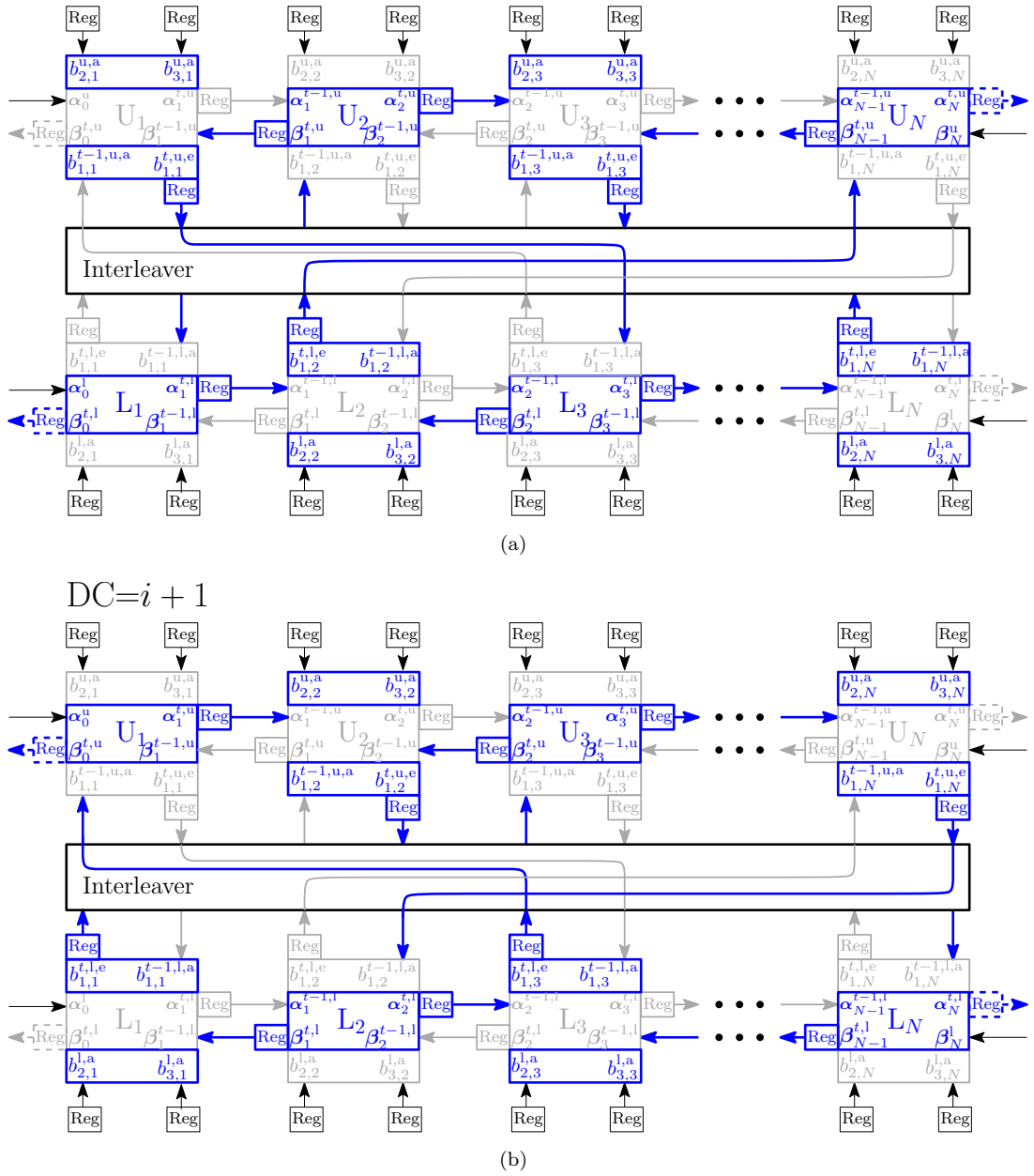


Figure 6.32: Odd-even operation of the RCP-FPTD. Only those sections highlighted in blue lines are operated in each DC. (a) Operation of the top and bottom sections of the blocks of the upper row having odd indexes, middle sections of the block of the upper row having even indexes, top and bottom sections of the blocks of the lower row having even indexes and middle sections of the block of the lower row having odd indexes. (b) Operation of the top and bottom sections of the blocks of the upper row having even indexes, middle sections of the block of the upper row having odd indexes, top and bottom sections of the blocks of the lower row having odd indexes and middle sections of the block of the lower row having even indexes.

error signal $\text{Error}_{b1,k}^{t-1,\text{a}}$ prevents the update of the DFFs of the γ module, which is driven by clock signal Clock_{top}, as shown in the red box of Figure 6.27. Note that the inputs of the Ext module correspond to the outputs provided by the ε module, as well as the

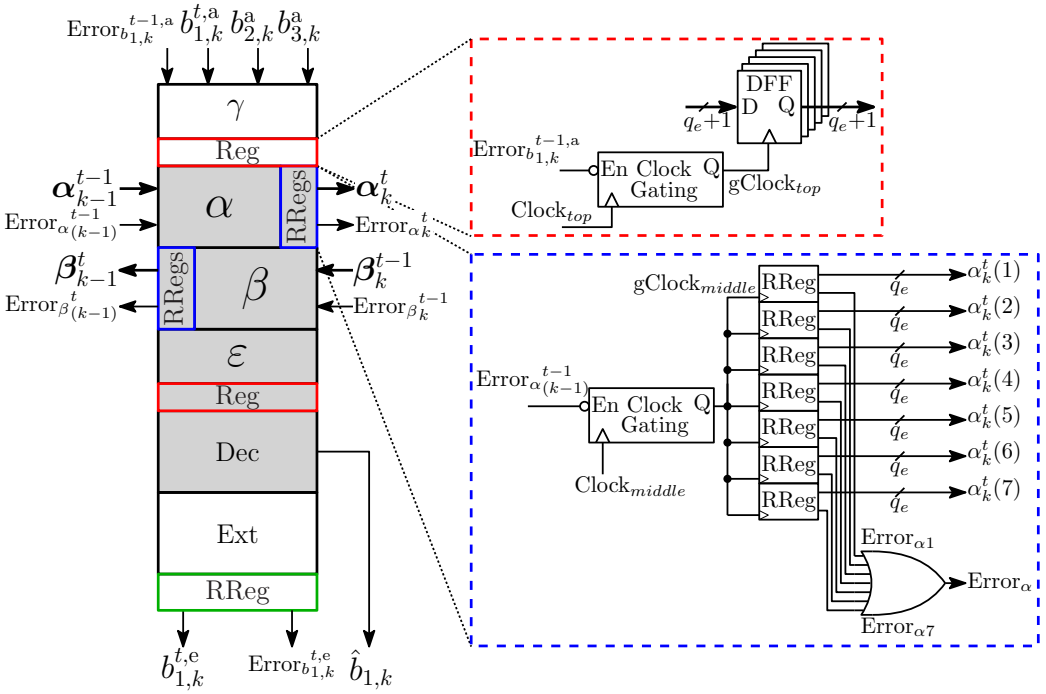


Figure 6.33: Implementation of RDFFs in a block of the BTWC-RCP-FPTD.

received channel LLR $b_{2,k}^{\mathrm{a}}$. However, the outputs ε do not employ RDFFs, owing to their low probability of timing error occurrence, as detailed above. Furthermore, the received channel LLR $b_{2,k}^{\mathrm{a}}$ remains constant throughout the iterative decoding process. As a consequence of this, we do not employ clock gating in the Ext module, since the inputs provided to this module are not susceptible to timing errors. Here, only one clock gating module is required in each block of the BTWC-RCP-FPTD for enabling and disabling the top modules of the decoding blocks. As a result of this, each block of the BTWC-RCP-FPTD employs a total of five clock gating modules. Additionally, only 14 out of a total of 132 DFF employ RDFFs.

In analogy to the BTWC-FPTD, only a small number of light-shaded blocks of Figure 6.13 may be operated in alternate DCs, but a large number of light-shaded blocks may be operated in others. Similarly, only a small number of dark-shaded blocks of Figure 6.13 may be operated in alternate DCs, but a large number of dark-shaded blocks may be operated in others. As a result of this, the BTWC-RCP-FPTD dynamically modifies the RCP-FPTD algorithm, owing to the occurrence of timing errors, as exemplified in Figure 6.34. Here, timing errors in $b_{1,1}^{\mathrm{u,e}}$ and $\boldsymbol{\alpha}_2^{\mathrm{u}}$ in the i^{th} DC prevent the operation of the bottom part of L_3, as well as the α module in the middle part of U_3, respectively, in the $i+1^{th}$ DC. Section 6.6.3.3 will demonstrate that both the BER performance and the hardware efficiency are significantly improved by the BTWC-RCP-FPTD, when compared to those of the RCP-FPTD in the presence of timing errors, despite this dynamic algorithm adaptation.

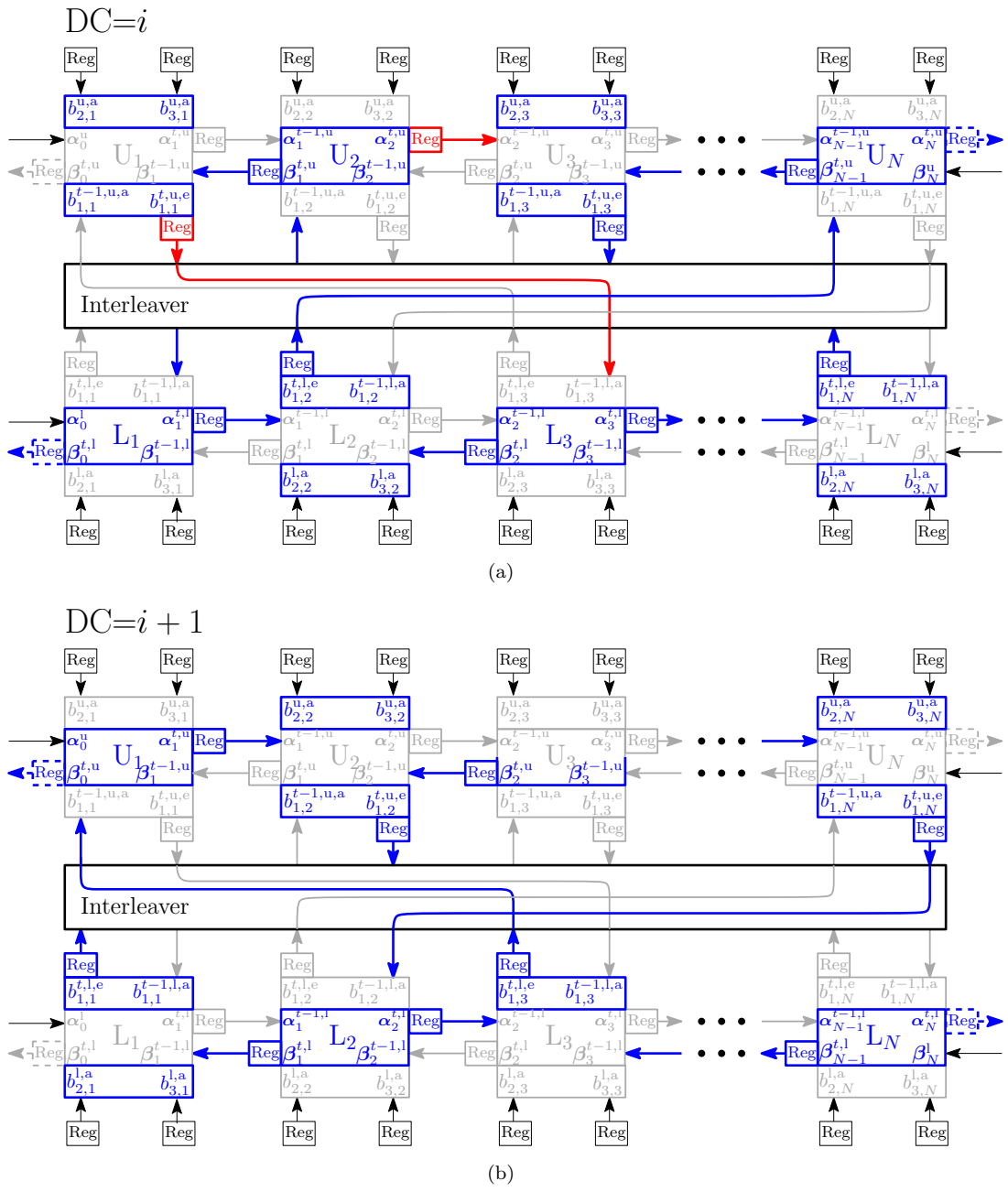


Figure 6.34: Odd-even operation of the BTWC-RCP-FPTD in the presence of timing errors. (a) Timing errors are detected in the MSB of the *extrinsic* LLR of U_1 and in the forward state metrics $\boldsymbol{\alpha}_2^{t,u}$. (b) The middle part of L_3, as well as the top and lower parts of U_3 are not operated, owing to the timing errors.

6.6.3.2 Timing analysis and timing error model

According to our post-layout Monte Carlo simulations for determining the CCDFs of each bit of the BTWC-RCP-FPTD, the employment of RDFF does not modify the timing characteristics of the RCP-FPTD. As a result of this, the critical clock period, as well as the CCDFs of the RCP-FPTD of Figure 6.20 remain unchanged, for the case of

the BTWC-RCP-FPTD design. This allows us to employ the timing error model of Section 6.6.2.2 for determining the occurrence of timing errors in the BTWC-RCP-FPTD. In each DC of the BER simulations, we employ random numbers and the CCDFs of Figure 6.20 for determining the propagation delay of each signal in the BTWC-RCP-FPTD. Following this, we employ Figure 6.22 and Table 6.4 for determining the causes and effects of timing errors for each of the bits of the BTWC-RCP-FPTD that are implemented using conventional DFFs, as detailed in Section 6.6.1. However, we employ Figure 6.29 and Table 6.5 for determining the causes and effects of timing errors in each of the bits of the BTWC-RCP-FPTD that employ RDFFs, as described in Section 6.6.2.2. Similarly, our timing error model assumes clock signals associated with a 50% duty cycle and a typical required setup time of the shadow D-type latch of $t_s^{\mathrm{S}} = 100$ ps for TSMC 40 nm technology [92]. As a result of this, the detection window is $\theta = T_{\mathrm{clk}}/2 - t_s^{\mathrm{S}}$.

6.6.3.3 Trade-off analysis in the presence of timing errors

This section characterizes the different trade-offs associated with the BER performance and hardware efficiency of the BTWC-RCP-FPTD in the presence of timing errors owing to overclocking. We employ the timing error model described in Section 6.6.2.2 for our BER simulations of the BTWC-RCP-FPTDs, for the cases when transmitting N={40,6144}-bit frames, using BPSK modulation over an AWGN channel, when using different degrees of overclocking and when allowing a maximum of 256 DCs with early stopping. We also present benchmark results for the RCP-FPTD operating at a clock period equal to its critical path delay in the absence of timing errors and at different degrees of overclocking in the presence of timing errors.

Figure 6.35 characterizes the BER performance, average number of DCs, throughput and energy efficiency of the RCP-FPTD and BTWC-RCP-FPTD implementations, for the case when $N = 40$. In analogy to the BTWC-FPTD, the results of Figure 6.35 demonstrate that the BER performance and hardware efficiency of the BTWC-RCP-FPTD are improved, when compared to those of the RCP-FPTD. More specifically, the BTWC-RCP-FPTD operated at 2.2 ns offers a 1.3 times throughput improvement, compared to the RCP-FPTD operated at both 3.0 ns and 2.2 ns, as shown in Figure 6.35. The same 1.3 times throughput improvement is observed for the $N = 6144$ BTWC-RCP-FPTD operated at 2.2 ns, when compared to the RCP-FPTD operated at both 3.0 ns and 2.2 ns, as shown in Figure 6.36. This is achieved despite the occurrence of timing errors in the BTWC-FPTD.

These results confirm that the BTWC designs of each FPTD enhances not only their inherent tolerance to timing errors, but also their hardware efficiency. This is particularly observed for the case of the BTWC-RCP-FPTD in Figure 6.31, where the RCP-FPTD operated at the critical clock period and in the absence of timing errors outperforms the RCP-FPTD operated at overclocked periods. However, the RCP-FPTD operated

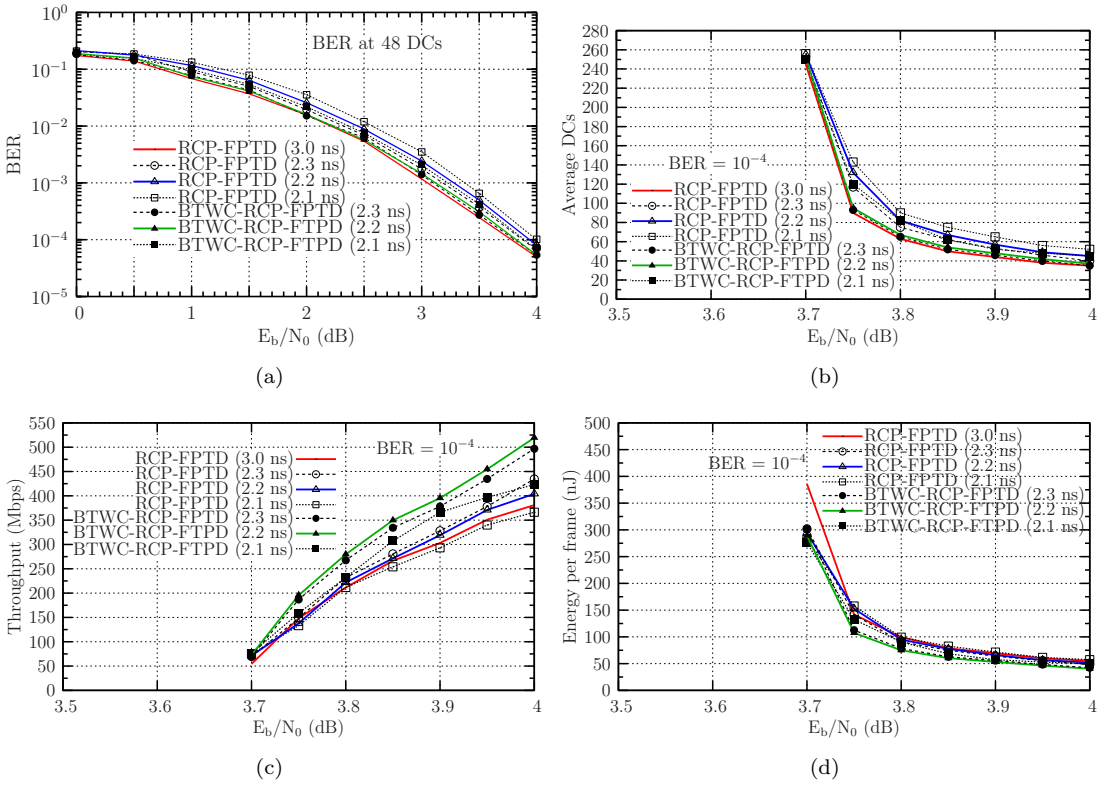


Figure 6.35: Trade-off analysis of the RCP-FPTD and of the BTWC-RCP-FTPD in the presence of timing errors owing to different degrees of overclocking, when transmitting $N = 40$ bits using BPSK over an AWGN channel. The results of average DCs, throughput and energy efficiency were obtained for the case of a target BER of 10^{-4} and when using a maximum of 256 DCs with early stopping. (a) BER performance at 48 DCs. (b) Average number of DCs. (c) Throughput. (d) Energy per decoding frame.

at the critical clock period does not outperform the BTWC-RCP-FPTD operated at an overclocked period of 2.2 ns. As a result of this, we propose the employment of the BTWC-RCP-FPTD operated at a clock period of 2.2 ns.

6.7 Hardware Efficiency of the Various FPTD Implementations

Tables 6.6 and 6.7 summarize the hardware efficiency of the various FPTD implementations presented in this chapter, for the cases of the shortest and longest frame lengths specified in the turbo code of the LTE standard [9], respectively.

Table 6.6 presents results for the hardware efficiency of the various FPTD designs, when using the frame length of $N = 40$, as well as results for a benchmark, namely the RLSTD presented in Chapter 5. Note that the BTWC-RCP-FPTD achieves the highest processing throughput, compared to all other implementations listed in Table 6.6. By

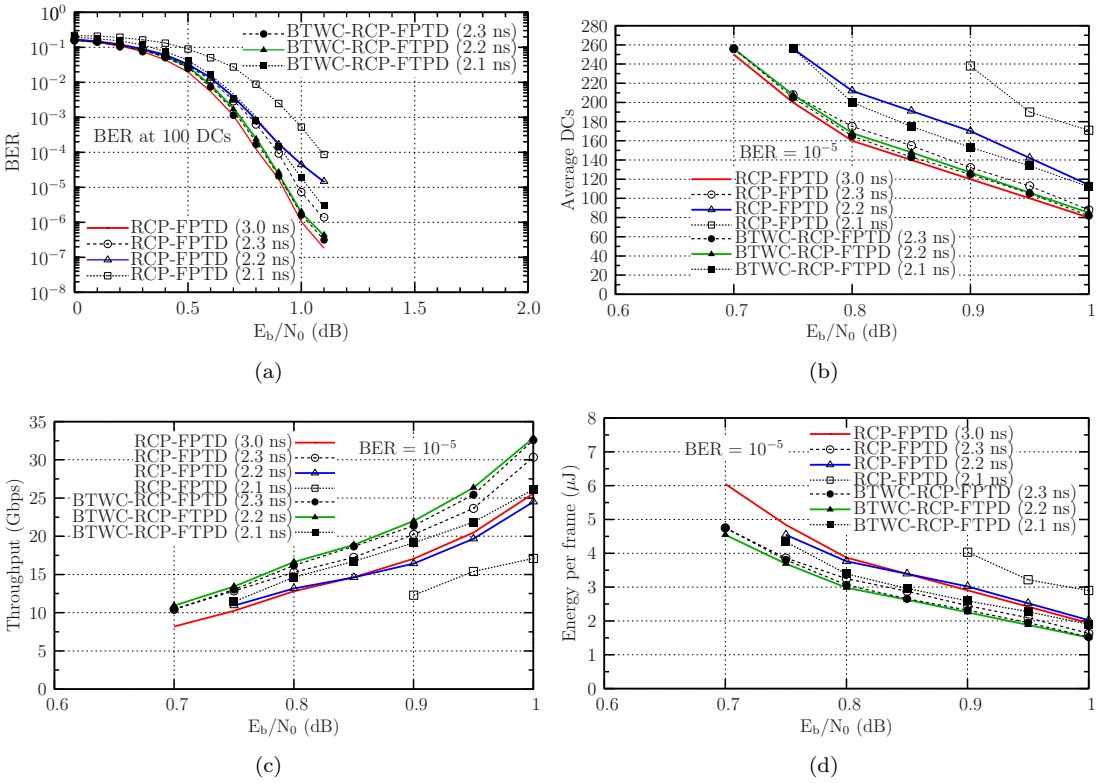


Figure 6.36: Trade-off analysis of the RCP-FPTD and of the BTWC-RCP-FTPD in the presence of timing errors owing to different degrees of overclocking, when transmitting $N = 6144$ bits using BPSK over an AWGN channel. The results of average DCs, throughput and energy efficiency were obtained for the case of a target BER of 10^{-5} and using a maximum of 256 DCs with early stopping. (a) BER performance at 100 DCs. (b) Average number of DCs. (c) Throughput. (d) Energy per decoding frame.

contrast, the area efficiency of the RLSTD scaled to 40 nm is superior to all other listed schemes. However, this area efficiency may be expected to be reduced, when timing errors occur in the RLSTD, since this design does not employ timing-error-tolerant techniques.

Table 6.7 presents the hardware efficiency of the various FPTD designs, when using the largest frame length $N = 6144$ of the turbo code specified in the LTE standard [9], as well as results for a benchmark, namely the state-of-the-art Application Specific Integrated Circuit (ASIC) turbo decoder of [57]. Note that the proposed BTWC-RCP-FPTD achieves the highest processing throughput, compared to all other implementations listed in Table 6.7. Moreover, the throughput of the BTWC-RCP-FPTD is 12.2 and 7.4 times the throughput of the state-of-the-art turbo decoder, when it uses 65 nm and 40 nm technologies, respectively. This is achieved by increasing the chip area by a factor of 5.7 and 15.2, for the cases of 65 nm and 40 nm technologies, respectively. However, the normalized area efficiency of the BTWC-RCP-FPTD using 40 nm is superior to that of the state-of-the-art turbo decoder using 65 nm.

Table 6.6: Hardware efficiency comparison of various 40-bit frame length FPTD implementations, when using TSMC 40 nm technology and V_{DD} = 1.0 V.

	FPTD		BTWC-FPTD	RCP-FPTD		BTWC-RCP-FPTD	RLSTD (Stoch.)
Frame length (bits)	40		40	40		40	40
Coding rate	40/132		40/132	40/132		40/132	1/30
Technology	40 nm		40 nm	40 nm		40 nm	90 nm
Voltage	1.0 V		1.0 V	1.0 V		1.0 V	1.2 V
Area per bit (μm^2)	6800		7100	7000		7250	11000 (4888)[a]
Gate count per bit	13.2 K		14.0 K	13.5 K		14.5 K	4.4 K
T_{clk} (ns)	4.5	3.3	3.3	3.0	2.2	2.2	1.7 (0.755)[b]
Frequency (MHz)	222	300	300	333	455	455	588 (1324)
E_b/N_0 (dB) @ BER = 10^{-4}	3.82	3.82	3.82	3.87	3.87	3.87	3.6
Average DCs	48	57	49	48	62	50	160
Latency (ns)	216	188	159	144	136	110	272 (120)[b]
Throughput (Mbps)	185	212	251	277	294	363	147 (333)
Area eff. (bps/gates)	349	400	447	512	543	626	835 (1892)[b]
Area eff. (Mbps/mm^2)	680	779	884	989	1050	1252	334 (1703)[a,b]
Energy eff. (nJ/frame)	108	89	77	75	74	56	32 (10)[a,b,c]

[a] Area $\sim s^2$
[b] $T_{clk} \sim s$
[c] Energy $\sim s \cdot V_s^2$

As a result of this trade-off analysis, we consider that the BTWC-RCP-FPTD is the most compelling design, owing to its attractive area efficiency, its significantly improved throughput and its enhanced tolerance to timing errors.

6.8 Chapter Conclusions

In this chapter, we have summarized the FPTD algorithm of [60]. We have presented a RCP-FPTD algorithm for reducing the number of data path stages in the decoder and hence the required clock period. We have detailed the hardware implementation of each of the FPTD and RCP-FPTD, as well as the corresponding trade-off analysis between chip area, latency, throughput and energy efficiency, when using TSMC 40 nm technology. Our results of Table 6.2 demonstrate that the proposed N = 48-bit FPTD

Table 6.7: Hardware efficiency comparison of various 6144-bit frame length FPTD implementations, when using TSMC 40 nm technology and $V_{DD} = 1.0$ V.

	FPTD		BTWC-FPTD	RCP-FPTD		BTWC-RCP-FPTD	Ilnseher 2012, [57]
Frame length (bits)	6144		6144	6144		6144	6144
Technology	40 nm		40 nm	40 nm		40 nm	65 nm
Area per bit (μm^2)	6800		7100	7000		7250	1280 (484)[a]
Core area (μm^2)	41.7		43.6	43.0		44.5	7.7 (2.92)[a]
Gate count per bit	13.2 K		14.0 K	13.5 K		14.5 K	-
T_{clk} (ns)	4.5	3.3	3.3	3.0	2.2	2.2	2.2 (1.35)[a]
Frequency (MHz)	222	300	300	333	455	455	450 (740)[b]
E_b/N_0 (dB) @ BER $= 10^{-5}$	0.90	0.90	0.90	0.95	0.95	0.95	-
Average iterations	80 DCs	151 DCs	94 DCs	100 DCs	142 DCs	106 DCs	6 I
Latency (ns)	360	498	310	300	312	233	2857 (1723)[b]
Throughput (Gbps)	17.1	12.3	19.8	20.4	19.6	26.3	2.15 (3.56)[b]
Area eff. (bps/gates)	210	151	230	245	235	295	-
Area eff. (Mbps/mm^2)	409	294	453	474	455	590	279 (1197)[a,b]
Energy eff. (μJ/frame)	2.7	3.7	2.4	2.4	2.5	1.8	-

[a] Area $\sim s^2$
[b] $T_{clk} \sim s$

implementations offer an attractive trade-off between the different hardware characteristics, when compared to the $N = 48$-bit RLSTD of Chapter 5. Similarly, our results of Table 6.3 demonstrate that the proposed $N = 6144$-bit FPTD and RCP-FPTD implementations achieve throughputs that are 7.9 and 9.5 times superior to those of the state-of-the-art turbo decoder of [57], respectively. However this is achieved by employing chip areas that are 5.42 and 5.58 times greater than that of the state-of-the-art turbo decoder. In addition to this, we have investigated the inherent tolerance to timing errors of the FPTD and RCP-FPTD. This was achieved by developing the timing error model of Section 6.5 for determining the causes and effects of timing errors in each FPTD, when overclocking is employed for the sake of improving the processing throughput. This was achieved by performing extensive digital post-layout simulations of each FPTD using Cadence SimVision [72], in order to determine the probability of observing timing errors, when the decoders are operated at different clock periods. In addition to this, we

have proposed the employment of BTWC design techniques in FPTD implementations, for the sake of further enhancing the inherent tolerance to timing errors of the various FPTDs, as detailed in Section 6.6. Our results were summarized in Section 6.7 and demonstrate that the proposed BTWC-FPTD and BTWC-RCP-FPTD operating in the presence of timing errors achieve a processing throughput that is 1.7 and 2.47 times superior to that of the proposed RLSTD of in Chapter 5, respectively, when employing the shortest frame length of $N = 40$ supported by the LTE standard. Similarly, when employing LTE's longest frame length of $N = 6144$, the proposed BTWC-FPTD and BTWC-RCP-FPTD operating in the presence of timing errors achieve processing throughputs that are 9.2 and 12.3 times superior than that of the state-of-the-art turbo decoder of [57] in the absence of timing errors, respectively.

Chapter 7

Conclusions and Future Work

This chapter presents our concluding remarks and summarizes our main findings in Section 7.1, while ideas for future research are discussed in Section 7.2. Finally, Section 7.3 presents our concluding remarks.

7.1 Conclusions and Summary

In this thesis, we have investigated the timing-error tolerance of various iterative decoders, with Stochastic LDPC Decoders (SLDPCDs), Stochastic Turbo Decoders (STDs) and Fixed-Point (FX) Fully-Parallel Turbo Decoders (FPTDs) as particular cases. We have demonstrated that these different iterative decoders have an inherent tolerance to correct not only *transmission* errors but also *timing* errors, which occur when the clock period or the supply voltage of the circuit are modified below the recommended safety margins, for the sake of improving their throughput or energy consumption, respectively. Along this way, we have developed different timing analyses in order to characterize the causes and effects of timing errors in iterative decoders. This has been achieved by performing extensive simulations of each implementation. Moreover, we have proposed different enhancements to the hardware implementations of the various iterative decoders, which significantly improve their tolerance to timing errors. Throughout this thesis, this has been achieved by considering the close relationship between the different trade-offs involved in the hardware implementation of Low-Density Parity-Check (LDPC) and turbo decoders, as listed in Figure 1.1. In this context, a compelling Pareto-optimal design has a set of characteristics, where none of them can be further enhanced without degrading at least one of the others. As a result, our proposed timing-error-tolerant design methodology simultaneously considers the design constraints and parameters that affect not only the Bit Error Ratio (BER) performance, but also the chip area, latency,

throughput and energy efficiency of each implementation. We summarize the our main contributions and findings as follows.

In **Chapter 1**, we have presented the introduction of this thesis, including a perspective of the related previous work in the selected areas of fault-tolerance in iterative decoders. We have also presented the motivation and structure of this thesis, as well as a summary of our novel contributions.

In **Chapter 2**, we have reviewed the basic concepts of iterative decoding and some fundamental aspects of the hardware implementation of iterative decoders. We have also reviewed the concept of stochastic computing, which was the foundation for the stochastic implementation of LDPC decoders in Chapter 3 and the stochastic implementation of turbo decoders in Chapters 4 and 5.

In **Chapter 3**, we have reviewed the concept of LDPC decoding and the implementation of the Sum-Product Algorithm (SPA) algorithm using stochastic computing. In Section 3.3 we have characterized the hardware efficiency and the BER performance of the (1056,528) WiMAX SLDPCD of [29], when using ST 90 nm technology. Following this, our extensive SPICE simulations of Section 3.4 allowed us to determine the causes and effects of timing errors in the SLDPCD of [29]. Our results of Figures 3.23 and 3.24 demonstrate that the SLDPCD has an inherent tolerance to timing errors caused by overclocking and power supply variations. Motivated by these results, in Section 3.5, we proposed a modified SLDPCD that further improves the timing error tolerance of the stochastic implementation of LDPC decoders. This has been achieved by replacing the Shift Register (SR)-based Edge Memory (EM) by a Ring Buffer (RB)-based EM structure, in order to mitigate the latching problem in SLDPCD. Our proposed RB-based SLDPCD operated at a supply voltage of 0.8 V, an overclock period of 800 ps and in the presence of timing errors owing to 10% power supply variations, offers the same BER performance as the SLDPCD of [29] operated a 0.8 V and a clock period of 1160 ps in the absence of timing errors, while increasing the throughput by a factor of 1.22, reducing the energy consumption by a factor of 0.7 and requiring only 0.77 the chip area of that of the SR-based SLDPCD of [29].

In **Chapter 4**, we have first reviewed the concept of turbo decoding and the implementation of the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm using stochastic computing. In Section 4.3, we have presented various modifications to the STD of [39] that enhanced its tolerance to timing errors and significantly improved its hardware efficiency, when using TSMC 90 nm technology, as characterized in Table 4.12 and Figure 4.12 of Section 4.4. Moreover, our timing analysis of Section 4.5 allowed us to determine the causes and effects of timing errors caused by power supply variations in the various STD designs. This has been achieved by carefully considering the close relationship of the different trade-offs associated with each modification. As a result, our proposed Tracking Forecast Memory (TFM)-5 STD design operated at 1.20 V, a clock period of

2.2 ns and in the presence of timing errors owing to 7% power supply variation offers the same BER performance as the state-of-the-art STD of [39], when operated at 1.20 V, a clock period of 4.0 ns and in the absence of power supply variations. This has been achieved while increasing the throughput by a factor of 2.42, reducing the latency by a factor of 0.83 and consuming only 0.25 times the energy, without increasing the chip area.

In **Chapter 5**, we have further improved the hardware efficiency of the STD of [39] by significantly reducing the number of Decoding Cycles (DCs) required in order to achieve iterative decoding convergence. This has been achieved by employing OR gates in order to perform approximate additions in stochastic computing. Additionally, we have proposed a Reduced-Complexity Tracking Forecast Memory (RCTFM), which benefits from the low-complexity of a SR, as well as from the TFMs's enhanced capability for tracking changes in the represented probabilities. Our proposed Reduced-Latency STD (RLSTD) design achieves the same BER performance as the state-of-the-art STD of [39], while improving its latency, throughput and energy efficiency by an order of magnitude and without imposing an area extension. More specifically, our proposed RLSTD requires only 0.015 times the latency, 0.005 times the energy consumption and 0.51 times the chip area of the state-of-the-art STD, while offering a 65 times throughput increase. We found that our proposed RLSTD design is particularly suited for short-frame-length and low-latency communication systems, such as those required in next-generation Mission-Critical Machine-Type Communication (MCMTC).

In **Chapter 6**, we have expanded the review of the BCJR algorithm into the logarithmic domain, in order to directly compare it with the recently-proposed FPTD algorithm of [60]. We have also reviewed the hardware implementation of the FPTD algorithm, as detailed in [61]. In Section 6.3, we have presented a novel Reduced-Critical-Path Fully-Parallel Turbo Decoder (RCP-FPTD) algorithm that reduced the number of data path stages required in the FPTD and hence the required clock period of its implementation. Moreover, in Section 6.3, we detailed the hardware implementation of the proposed RCP-FPTD, as well as its corresponding trade-off analysis, when using TSMC 40 nm technology. Our results of Tables 6.2 and 6.3 in Section 6.4 demonstrate that the FPTD and RCP-FPTD implementations offer significant throughput increases, when compared to both the RLSTD of Chapter 5 and the state-of-the-art turbo decoder implementation of [57], while achieving the same BER performance. As an example of this, the FPTD and RCP-FPTD implementations offer throughputs that are 7.9 and 9.5 times superior to those of the turbo decoder of [57], respectively, albeit at the cost of requiring 5.42 and 5.58 times the chip area. Furthermore, in Section 6.5 we have developed a timing error model of the FPTD and RCP-FPTD implementations. This error model allowed us to demonstrate that the FPTD and RCP-FPTD implementations have an inherent tolerance to timing errors, when overclocking is employed for the sake of further

improving their throughput. In addition to this, in Section 6.6, we have employed Better-Than-Worst-Case (BTWC) design techniques for the implementation of the FPTD, in order to significantly enhance its inherent tolerance to timing errors and its hardware efficiency. Our results of Section 6.7 demonstrated that the proposed Better-Than-Worst-Case Fully-Parallel Turbo Decoder (BTWC-FPTD) and Better-Than-Worst-Case Reduced-Critical-Path Fully-Parallel Turbo Decoder (BTWC-RCP-FPTD) operated in the presence of timing errors achieve a processing throughput that is 1.7 and 2.47 times superior to that of the proposed RLSTD of Chapter 5, respectively, when employing the shortest frame length of $N = 40$ supported by the Long Term Evolution (LTE) standard. Similarly, when employing LTE's longest frame length of $N = 6144$, the proposed BTWC-FPTD and BTWC-RCP-FPTD operating in the presence of timing errors achieve processing throughputs that are 9.2 and 12.3 times superior than that of the state-of-the-art turbo decoder of [57] in the absence of timing errors, respectively.

7.2 Future Work

This section presents suggestions for future research. Section 7.2.1 highlights opportunities for improving the hardware efficiency of some of the implementations of iterative decoders presented in this thesis. Section 7.2.2 summarizes opportunities for further enhancing our different timing analyses. Finally, Section 7.2.4 presents opportunities regarding the validation of our timing error models.

7.2.1 Hardware Efficiency of Iterative Decoders

The hardware efficiency of the different implementations of iterative decoders conceived in this thesis may be further enhanced. As an example of this, the proposed RLSTD design presented in Chapter 5 design does not consider error-tolerant design techniques, such as those presented in Sections 3.5, 4.3 and 6.6. As a result of this, the occurrence of timing errors may degrade the error correction capabilities and the hardware performance of the RLSTD. More specifically, the number of DCs required by the RLSTD may be expected to increase in the presence of timing errors, as our simulations of Figures 4.16 and 4.17 suggest for the case of the STDs in Chapter 4. To overcome this problem, a timing analysis similar to those presented in Sections 3.4, 4.5 and 6.5 may be employed for determining the causes and effects of timing errors in the RLSTD. Similarly, the design techniques such as those described in Sections 4.3 and 6.6 may be used in the RLSTD for improving its tolerance to timing errors. More specifically, output synchronizers may be employed for mitigating the catastrophic propagation of metastability through the circuit, albeit at the cost of increasing the latency of the RLSTD. Similarly, Razor D-Type Flip Flops (RDFFs) may be employed in the RCTFM of the RLSTD, in order to prevent timing errors from affecting the Most Significant Bit (MSB)

of the TFM, which severely degrades the BER performance of the STD, as Section 4.5.2 demonstrated.

The hardware efficiency of the FPTD and RCP-FPTD implementations of Chapter 6 may be further improved by the more efficient chip area utilization. More specifically, this may be achieved by operating all $2N$ blocks of the FPTD in each DC for the simultaneous decoding of two received frames, in analogy to the proposed timing-error-tolerant STD presented in Chapter 4. Owing to the odd-even operation of the various FPTDs, two independent decoding processes may be achieved. This will result in a significant throughput increase and a negligible chip area increase, which will significantly increase the normalized area efficiency of the decoders. Alternatively, the chip area requirements may be halved by reusing hardware to alternate between the decoding process of the light- and dark-shaded blocks of Figures 6.5 and 6.13 in alternate clock cycles. However, this will require the employment of an alternative BTWC design, since the same hardware block of each FPTD will be operated in two consecutive DCs, which may cause timing errors occurring in one block to affect the operation of the other block operated using the same hardware.

7.2.2 Timing Error Model

The different timing error models conceived in this thesis assume a static operation of the circuit, where the nominal supply voltage and the clock period are fixed throughout the operation of the decoder. In these error models, power supply variations are assumed to have a Gaussian distribution with a fixed mean μ corresponding to the nominal supply voltage. However, these error models do not consider the employment of Dynamic Voltage and Frequency Scaling (DVFS) [68], which is a technique commonly found in current System-on-Chip (SoC) designs. In DVFS, the supply voltage and the clock period of the SoC are dynamically adjusted according to a workload metric, which is determined by the performance of the SoC core, as shown in Figure 7.1.

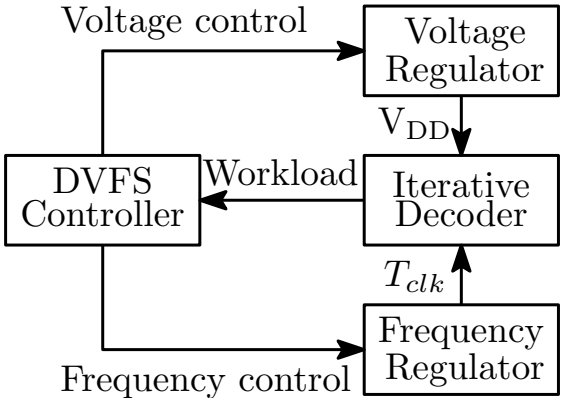


Figure 7.1: Example of DVFS in iterative decoders.

If DVFS is applied to the various iterative decoders presented in this thesis, the supply voltage may be dynamically overscaled in order to reduce the dynamic energy consumption. Similarly, the clock period may be dynamically overscaled, in order to increase the processing throughput of the decoders. In this way, the DVFS decoder may employ a workload metric related to the number of timing errors encountered in each decoding iteration, for example. However, this approach is expected to significantly increase the hardware complexity, owing to the employment of voltage and frequency regulators. Finally, the clock periods and nominal supply voltages recommended for the operation of the various iterative decoders in this thesis may be considered as a first step towards the DVFS design, since these values determine ranges of operating conditions in which the decoders exhibit near-optimal performance.

7.2.3 Power Gating in Stochastic Turbo Decoders

The employment of Clock Gating (CG) for reducing the dynamic energy consumption of the STDs of Chapter 4 does not mitigate the additional static energy consumption that is associated with the inclusion of the additional set of EMs for the simultaneous decoding of two received frames. In order to overcome this static energy increase, the STD may employ other low-power design techniques, such as power gating or multiple- and variable-threshold transistor design, although this is achieved at the cost of increasing the complexity of the design. As an example of this, power gating may be employed for switching off specific blocks of the STD, when they are not being operated. However, state retention registers are required for restoring the state of the blocks upon power-up. Owing to this overhead, power gating is only effective when specific blocks may be switched off for a significant amount of time. Motivated by this, our future work will conceive techniques for applying early-stopping to different blocks at different stages in the iterative decoding process.

7.2.4 FPTD ASIC Fabrication and Error Model Validation

The various timing error models presented in this work rely on transistor-level or digital post-layout simulations with extracted parasitics and annotated delays of specific blocks of each design. However, a full characterization of each design may only be possible after its tapeout. Motivated by the significant throughput improvement and timing-error tolerance offered by the different FPTDs and RCP-FPTDs implementations of Chapter 6, our future research will consider the fabrication of an FPTD and an BTWC-RCP-FPTD using TSMC 40 nm technology. Owing to the large chip area requirements of the FPTD and BTWC-RCP-FPTD implementations, we will limit the size of the frame length to only $N = 40$ for both designs. Despite this limitation, the tapeout of these designs will allow us to effectively determine and enhance the accuracy of our timing error model.

7.3 Closing Remarks

This thesis has demonstrated that iterative decoders are capable of exploiting their ***inherent*** error correction capability to correct not only ***transmission*** errors, but also ***timing*** errors caused by overclocking and power supply variations. Moreover, this thesis has proposed modifications to the iterative decoders designs, which further enhance their inherent tolerance to timing errors. This has been achieved by considering the close relationship between the different trade-offs associated with the hardware implementation of iterative decoders, with the aim of achieving Pareto optimality. As a result, our proposed implementations are particularly suited for short-frame-length and low-latency communication systems, such as the next-generation MCMTCs, as well as for next-generation high-throughput wireless communication standards, such as 5G.

Appendix A

Design Flow

This appendix presents the design flow used throughout this thesis, as shown in Figure A.1. A typical Application Specific Integrated Circuit (ASIC) design process starts with the design of the algorithm or specifications and proceeds through each of the steps of Figure A.1 until the fabrication of the ASIC. This complete process is detailed as follows for the specific case of the implementation of iterative decoders.

1. *Algorithm design.* In this step, parameters of the algorithm such as frame-length, coding rate and scheduling are defined.

2. *Floating-Point (FP) simulation.* In this step, a FP simulation of the algorithm is performed in order to validate that the algorithm may achieve the target Bit Error Ratio (BER) performance. This step may be performed with the aid of C++ or Matlab, for example.

3. *Fixed-Point (FX) simulation.* Following the FP simulation, a FX simulation is performed in order to validate that the employment of FX numbers does not degrade the error correction capabilities of the algorithm. In parallel to this, this step is performed to evaluate the bit width requirements of FX numbers in order to strike an attractive trade-off between the hardware complexity and error correction capabilities of the implementation. Similarly, this step allows to estimate the average number of Decoding Cycles (DCs) required by the hardware implementation. This step may be performed using C++ or Matlab, for example.

4. *Hardware description.* This step uses hardware description languages, such as SystemVerilog, in order to model the algorithm using elements of digital circuit design. This is achieved by considering the bit width requirements obtained in step 3, the data flow between processing elements in consecutive clock cycles and the control signals required by the design.

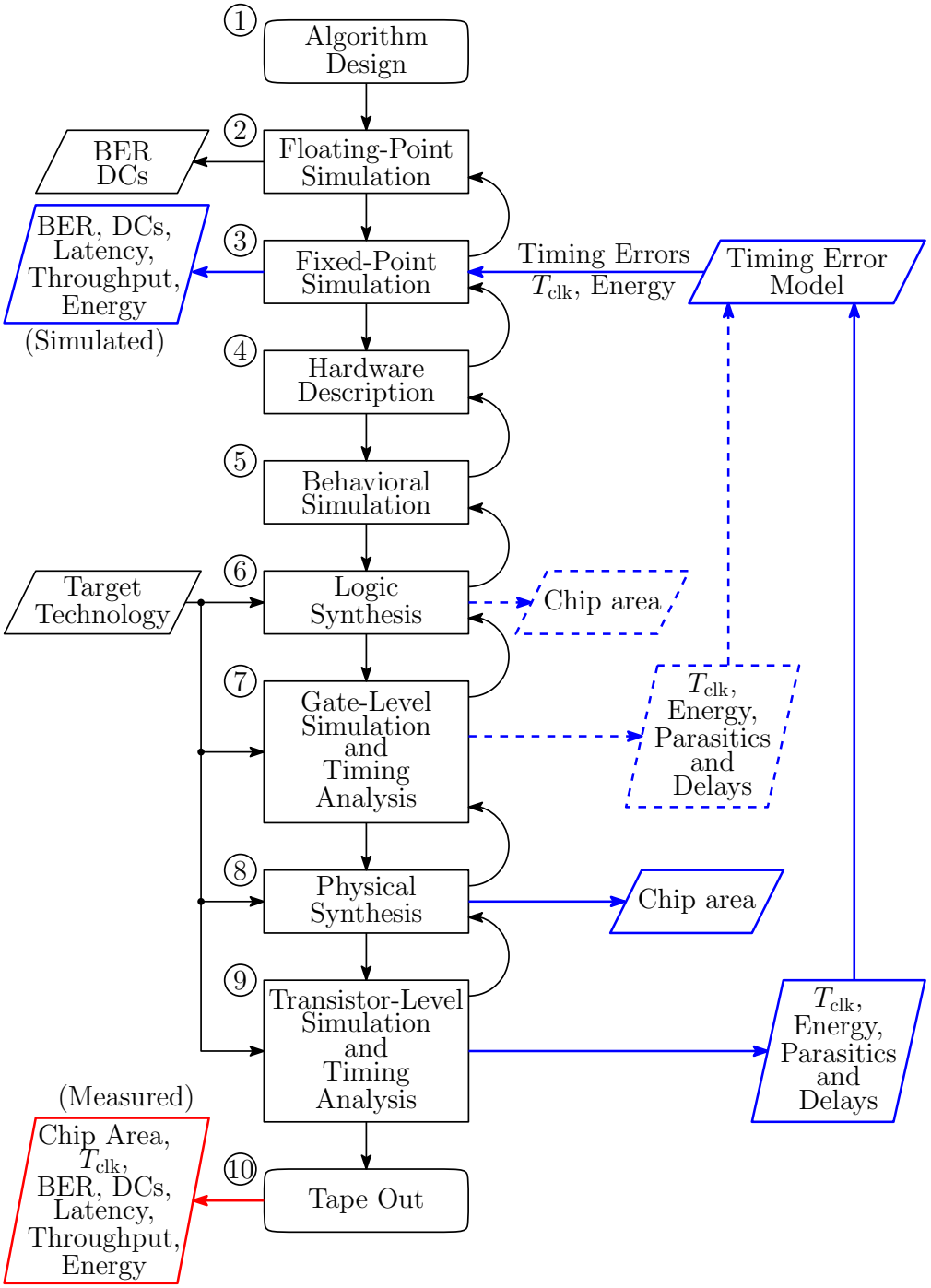


Figure A.1: Design flow used throughout this thesis.

5. *Behavioral simulation.* The hardware description of step 4 is simulated in order to validate the correct functionality of the algorithm using hardware description languages. This step may be performed using Cadence's SimVision [72], for example.

6. *Logic synthesis.* The hardware description of step 4 is translated into basic logic digital gates for a target technology. This steps allows to make an initial estimation of the chip area requirements of the implementation. More specifically, the

synthesis tools generate automatic reports to estimate the chip area of the design, based on the target technology. This step may be performed using Synopsys' DesignCompiler [73] or Cadence's RTL Compiler [72], for example.

7. *Gate-level simulation and timing analysis.* This step is performed in order to validate the correct functionality of the algorithm using the basic logic gates obtained in step 6. This simulation may be performed with the aid of Cadence's SimVision [72], for example. In parallel to this, a timing analysis is performed in order to determine the minimum clock period T_{clk} at which the gate-level version of the algorithm may operate without causing timing errors. Additionally, this step allows the initial estimation of the energy consumption and an approximate extraction of the parasitics and propagation delays, which may be employed in a timing error model, as shown in the blue lines of Figure A.1. The timing analysis and the parasitics extraction may be performed using Synopsys' PrimeTime [73], for example.

8. *Physical synthesis.* In this step, the specific gate geometries are considered in order to design an ASIC layout. Here, the gate-level version of the algorithm of step 6 is translated into a transistors-level version. An accurate estimation of the chip area may be performed in this step. This may be achieved through the processes of placement and routing using Cadence's Encounter [72].

9. *Transistor-level simulation and timing analysis.* In this step, a digital simulation using Cadence's SimVision [72] may be performed in order to validate the correct functionality of the algorithm using the transistor-level version of the design. Similarly, a timing analysis is performed in order to determine the minimum clock period T_{clk} at which the transistor-level version of the algorithm may operate without causing timing errors. Additionally, this step allows the accurate estimation of the energy consumption and an approximate extraction of the parasitics and propagation delays, which may be employed in a timing error model in order to evaluate the performance of the algorithm in the presence of timing errors, as shown in the blue arrows of Figure A.1. The timing analysis and the parasitics extraction may be performed using Cadence's Encounter [72].

10. *Tape out.* This is the final step in the ASIC design flow and consists of the fabrication of the chip. After the chip is fabricated, it is possible to obtain accurate results of the chip area, clock period, BER, DCs, latency, throughput and energy efficiency of the ASIC.

The design steps of Figure A.1 may be iterated several times, before the final fabrication of the ASIC. Moreover, the design flow is enhanced during each step, providing a more accurate characterization of the behavior of the implementation. However, the complexity of the design is significantly increased through each subsequent step. As an

example of this, during the logic synthesis process of step 6, the area required by the interconnections is not considered. Owing to this, the results obtained after steps 6 and 7 may be only considered as an initial approximation of the chip area, clock period and energy consumption, as shown by the dotted lines of Figure A.1. In contrast to this, the physical synthesis process of step 8 considers the interconnections of the design. As a result of this, the chip area, clock period and energy consumption of steps 8 and 9 may be considered more accurate and closer to the final fabricated ASIC, albeit at the cost of increasing the complexity of the design as well as the simulation time.

As an alternative to the digital simulation of the transistor-level design of step 9 of Figure A.1, an analogue simulation may be performed using SPICE [73]. Here, SPICE allows to evaluate the voltages and currents associated with each transistor of a design. Moreover, SPICE allows to simulate the resistive and capacitive parasitics associated with the interconnections of the transistors. As a result of this, an even more accurate estimation of the energy consumption and propagation delays may be obtained, albeit at the cost of a significant complexity and simulation time increase. Owing to this, we recommend the employment of SPICE simulations only on critical stages of the design. For example, in Chapter 3, we used SPICE simulations to determine the causes and effects of timing errors in Stochastic LDPC Decoders (SLDPCDs). A brief overview of the steps necessary to perform the analogue simulation of the SLDPCDs using SPICE is listed as follows.

i) Determine the logic gates associated with the critical path of the SLDPCDs.

ii) Interconnect the transistor-level version of the logic gates, using the extracted resistive and capacitive parasitics.

iii) Provide a clock signal and a supply voltage in order to avoid the occurrence of timing errors.

iv) Generate test patterns and perform the analogue simulation.

v) Measure the propagation delays.

vi) In order to observe timing errors, reduce the nominal supply voltage or reduce the clock period and perform step iv).

Note that the parameters obtained from steps 7 and 9 allow the elaboration of a timing error model, which may be used in a FX simulation in order to estimate the effects of timing errors in the algorithm. This is represented using a feedback loop using blue arrows in the design flow of Figure A.1. By considering the timing error model in the FX simulation, it is possible to estimate the error correction capabilities, number of DCs, latency, throughput and energy efficiency of the algorithm's implementation in the presence of timing errors. This is achieved without the high risk and financial cost associated with the ASIC fabrication of step 10 of Figure A.1.

References

[1] C. Shannon, "Communication in the presence of noise," *Proc. IRE*, vol. 37, no. 1, pp. 10 – 21, jan. 1949.

[2] R. Gallager, "Low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan 1962.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Rec. IEEE Int. Conf. Communications, (ICC-1993)*, Geneva, Switzerland, May 1993, pp. vol. 2, 1064–1070.

[4] *IEEE Standard for Wireless Metropolitan Area Networks*, IEEE 802.16 Std., 2012.

[5] *IEEE Standard for Wireless Local Area Networks*, IEEE 802.11 Std., 2012.

[6] *IEEE Standard for Ethernet*, IEEE 802.3 Std., 2012.

[7] *DVB-S2 Standard*, ETSI EN 302 307 Std., 2009.

[8] *UMTS Standard*, ETSI TS 125 331 Std., 2011.

[9] *LTE Standard*, 3GPP TS 36.212 Std., 2011.

[10] "5G: A technology vision," White paper, HUAWEI Technologies Co., Nov 2013.

[11] E. Dahlman, G. Mildh, S. Parkvall, J. Peisa, J. Sachs, and Y. Selén, "5G Radio Access," Ericsson, Stockholm, Sweden, Tech. Rep., June 2014.

[12] R. Ahmadi and F. N. Najm, "Timing analysis in presence of power supply and ground voltage variations," in *Proc. Int. Conf. Computer Aided Design, (ICCAD-2003)*, San Jose, CA., Nov 2003, pp. 176–183.

[13] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture, (MICRO-2003)*, San Diego, CA., Dec 2003, pp. 7–18.

[14] N. Ahmed, M. Tehranipoor, and V. Jayaram, "A novel framework for faster-than-at-speed delay test considering IR-drop effects," in *Proc. Int. Conf. Computer Aided Design, (ICCAD-2006*, San Jose, CA., Nov 2006, pp. 198–203.

[15] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks, "Understanding voltage variations in chip multiprocessors using a distributed power-delivery network," in *Proc. Design, Automation Test Europe Conf. Exhibition, (DATE-2007)*, Nice, France, April 2007, pp. 1–6.

[16] M. Alioto, G. Palumbo, and M. Pennisi, "Understanding the effect of process variations on the delay of static and domino logic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 697–710, May 2010.

[17] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM J. Research and Development*, vol. 6, no. 2, pp. 200–209, April 1962.

[18] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. New York, NY: Plenum, 1969, ch 2, pp. 37–172.

[19] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," in *Proc. Asia South Pacific Design Automation Conf. (ASPDAC-2005)*. Shanghai, China: ACM, 2005, pp. 2–7.

[20] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov 2004.

[21] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning DVS processor using delay-error detection and correction," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, April 2006.

[22] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "RazorII: In situ error detection and correction for PVT and SER tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, Jan 2009.

[23] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *IET Electron. Lett.*, vol. 39, no. 3, p. 299, April 2003.

[24] A. C. Rapley, C. Winstead, V. C. Gaudet, and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Proc. 3rd Int. Symp. Turbo Codes Related Topics*, Brest, France, September 2003, pp. 507–510.

[25] C. Winstead, V. C. Gaudet, A. C. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Proc. IEEE Int. Symp. Information Theory, (ISIT-2005)*, Adelaide, Australia, Sep 2005, pp. 1116–1120.

[26] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of LDPC Decoders," in *Conf. Rec. 39th Asilomar Conf. Signals, Systems Computers*, Pacific Grove, CA, Nov 2005, pp. 713–717.

[27] S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct 2006.

[28] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding," in *Proc. IEEE Workshop Signal Processing Systems (SiPS-2007)*, Oct 2007, pp. 255–260.

[29] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, Nov 2008.

[30] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. J. Gross, "Tracking forecast memories in stochastic decoders," in *Proc. IEEE Int. Conf. Acoustics, Speech Signal Processing, (ICASSP-2009)*, Taipei, Taiwan, April 2009, pp. 561–564.

[31] F. Leduc-Primeau, S. Hemati, W. J. Gross, and S. Mannor, "A relaxed half-stochastic iterative decoder for LDPC codes," in *Proc. IEEE Global Telecommun. Conf., (GLOBECOM-2009)*, Honolulu, HI, Nov 2009, pp. 1–6.

[32] G. Sarkis and W. Gross, "Reduced-latency stochastic decoding of LDPC codes over GF(q)," in *Proc. European Wireless Conf. (EW-2010)*, Lucca, Italy, April 2010, pp. 994–998.

[33] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, Sept 2010.

[34] A. Naderi, S. Mannor, M. Sawan, and W. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5617–5626, Nov 2011.

[35] C. Ceroici and V. Gaudet, "FPGA implementation of a clockless stochastic LDPC decoder," in *Proc. IEEE Workshop Signal Process. Syst., (SiPS-2014)*, Belfast, Northern Ireland, Oct 2014, pp. 1–5.

[36] S. Sharifi Tehrani, C. Jego, Z. Bo, and W. Gross, "Stochastic decoding of linear block codes with high-density parity-check matrices," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5733–5739, Nov 2008.

[37] M. Arzel, C. Lahuec, C. Jego, W. Gross, and Y. Bruned, "Stochastic multiple stream decoding of cortex codes," *IEEE Trans. Signal Process.*, vol. 59, no. 7, pp. 3486–3491, July 2011.

[38] C. Te-Hsuan and J. Hayes, "Design of stochastic Viterbi decoders for convolutional codes," in *IEEE 16th Int. Symp. Design Diagnostics Electronic Circuits Systems, (DDECS-2013)*, Karlovy Vary, Czech Republic, April 2013, pp. 66–71.

[39] Q. T. Dong, M. Arzel, C. Jego, and W. J. Gross, "Stochastic decoding of turbo codes," *IEEE Trans. Signal Process.*, vol. 58, no. 12, pp. 6421–6425, Dec 2010.

[40] Q. T. Dong, M. Arzel, C. Jego, and W. J. Gross, "Design and FPGA implementation of stochastic turbo decoder," in *Proc. IEEE 9th Int. New Circuits Systems Conf., (NEWCAS-2011)*, Bordeaux, France, June 2011, pp. 21–24.

[41] J. Hu, Y. Deng, J. Chen, and X. Ling, "High speed turbo decoder design based on stochastic computation," in *Proc. Int. Conf. Communications Circuits Systems (ICCCAS-2013)*, vol. 1, Chengdu, China, Nov 2013, pp. 235–239.

[42] J. Chen and J. Hu, "High throughput stochastic Log-MAP turbo-decoder based on low bits computation," *IEEE Trans. Signal Process.*, vol. 20, no. 11, pp. 1098–1101, Nov 2013.

[43] O. N. C. Yilmaz, Y.-P. Wang, N. Johansson, N. Brahmi, S. Ashraf, and J. Sachs, "Analysis of ultra-reliable and low-latency 5G communication for a factory automation use case," in *Proc. Int. Conf. Communication Workshop (ICCW-2015)*, London, UK, June 2015, pp. 1190–1195.

[44] M. Alles, T. Brack, and N. Wehn, "A reliability-aware LDPC code decoding algorithm," in *Proc. IEEE 65th Vehicular Technology Conf. (VTC-2007-Spring)*, Dublin, Ireland, April 2007, pp. 1544–1548.

[45] C. Winstead and S. Howard, "A probabilistic LDPC-coded fault compensation technique for reliable nanoscale computing," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 6, pp. 484–488, June 2009.

[46] V. C. Gaudet, "Low-power LDPC decoding by exploiting the fault-tolerance of the sum-product algorithm," in *Contemporary Mathematics*. American Mathematical Society, 2010, pp. 523:165–171.

[47] Y. Tang, C. Winstead, E. Boutillon, C. Jego, and M. Jezequel, "An LDPC decoding method for fault-tolerant digital logic," in *IEEE Int. Symp. Circuits Systems (ISCAS-2012)*, Seoul, Korea, May 2012, pp. 3025–3028.

[48] C. Winstead, Y. Tang, E. Boutillon, C. Jego, and M. Jezequel, "A space-time redundancy technique for embedded stochastic error correction," in *Proc. 7th Int. Symp. Turbo Codes Iterative Information Processing (ISTC-2012)*, Gothenburg, Sweden, Aug 2012, pp. 36–40.

[49] J. Geldmacher and J. Gotze, "EXIT-optimized index assignments for turbo decoders with unreliable LLR transfer," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 992–995, May 2013.

[50] C. Kameni Ngassa, V. Savin, and D. Declercq, "Faulty stochastic LDPC decoders over the binary symmetric channel," in *Proc. Int. Symp. Turbo Codes Iterative Information Processing (ISTC-2014),*, Bremen, Germany, Aug 2014, pp. 112–116.

[51] J. Andrade, A. Vosoughi, G. Wang, G. Karakonstantis, A. Burg, G. Falcao, V. Silva, and J. Cavallaro, "On the performance of LDPC and turbo decoder architectures with unreliable memories," in *Conf. Rec. 48th Asilomar Conf. Signals, Systems Computers*, Pacific Grove, CA, Nov 2014, pp. 542–547.

[52] C.-H. Huang, Y. Li, and L. Dolecek, "Belief propagation algorithms on noisy hardware," *IEEE Trans. Commun.*, vol. 63, no. 1, pp. 11–24, Jan 2015.

[53] M. May, M. Alles, and N. Wehn, "A case study in reliability-aware design: a resilient LDPC code decoder," in *Proc. Design, Automation Test Europe Conf. Exhibition, (DATE-2008)*, Munich, Germany, March 2008, pp. 456–461.

[54] R. Abdallah and N. Shanbhag, "Error-resilient low-power Viterbi decoder architectures," *IEEE Trans. Signal Process.*, vol. 57, no. 12, pp. 4906–4917, Dec 2009.

[55] E. Kim and N. Shanbhag, "Energy-efficient LDPC decoders based on error-resiliency," in *Proc. IEEE Workshop Signal Processing Systems (SiPS-2012)*, Quebec, Canada, Oct 2012, pp. 149–154.

[56] B. Sedighi, N. Anthapadmanabhan, and D. Suvakovic, "Timing errors in LDPC decoding computations with overscaled supply voltage," in *Proc. Int. Symp. Low Power Electronic Design, (ISLPED-2014).* La Jolla, CA, USA: ACM, 2014, pp. 201–206.

[57] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s turbo code decoder for LTE advanced base station applications," in *Proc. 7th Int. Symp. Turbo Codes Iterative Information Processing (ISTC-2012)*, Gothenburg, Sweden, Aug 2012, pp. 21–25.

[58] G. Wang, H. Shen, Y. Sun, J. Cavallaro, A. Vosoughi, and G. Yuanbin, "Parallel interleaver design for a high throughput HSPA+/LTE multi-standard turbo decoder," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 5, pp. 1376–1389, May 2014.

[59] Y. Sun and C. J. R., "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," *Integr. VLSI J.*, vol. 44, no. 4, pp. 305–315, Sept 2011.

[60] R. G. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2762–2775, Aug 2015.

[61] A. Li, L. Xiang, T. Chen, R. G Maunder, B. M. Al-Hashimi, and L. Hanzo, "VLSI implementation of fully-parallel LTE turbo decoders," *IEEE Access*, vol. 4, pp. 323–346, January 2016.

[62] I. Levi and A. Fish, "Dual mode logic-design for energy efficiency and high performance," *IEEE Access*, vol. 1, pp. 258–265, May 2013.

[63] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, no. 2, pp. 284–287, Mar 1974.

[64] Y. Wu, and B. Woerner, "The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes," in *49th IEEE Vehicular Technology Conf. (VTC-1999)*, vol. 2, Houston, TX, Jul 1999, pp. 1683–1687 vol.2.

[65] B. Brown and H. Card, "Stochastic neural computation. I. Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep 2001.

[66] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 92:1–92:19, May 2013.

[67] F. Kschischang, B. Frey, and H.-a. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[68] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571 –1580, nov. 2000.

[69] N. Bonello, C. Sheng, and L. Hanzo, ""low-density parity-check codes and their rateless relatives"," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 1, pp. 3–26, May 2010.

[70] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, March 2006.

[71] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

[72] Cadence Desing Systems Inc., "User manuals for cadence toolset," 2015.

[73] Synopsys Inc., "User manuals for synopsys toolset," 2015.

[74] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A circuits and systems perpsective*, 4th ed. Addison Wesley, 2011.

[75] M. Nourani and A. Radhakrishnan, "Power-supply noise in SoCs: ATPG, estimation and control," in *Proc. IEEE Int. Test Conf. (ITC-2005)*, Austin, TX, USA, Nov 2005, pp. 507–516.

[76] S. Pant, D. Blaauw, V. Zolotov, S. Sundareswaran, and R. Panda, "Vectorless analysis of supply noise induced delay variation," in *Proc. Int. Conf. Computer Aided Design,(ICCAD-2003)*, San Jose, CA, USA, Nov 2003, pp. 184–191.

[77] P. N. Whatmough, S. Das, D. M. Bull, and I. Darwazeh, "Circuit-level timing error tolerance for low-power DSP filters and transforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 989–999, June 2013.

[78] S. Purohit and M. Margala, "Investigating the impact of logic and circuit implementation on full adder performance," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 7, pp. 1327–1331, July 2012.

[79] Q. T. Dong, "Le principe de calcul stochastique appliqué au décodage des turbocodes: conception, implémentation et prototypage sur circuit FPGA," Ph.D. dissertation, Télécom Bretagne, Brest, France, December 2011.

[80] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Des. Test. Comput.*, vol. 28, no. 5, pp. 23–35, Sept 2011.

[81] J. Anderson and S. Hladik, "Tailbiting MAP decoders," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 297–302, Feb 1998.

[82] S. Beer, J. Cox, R. Ginosar, T. Chaney, and D. Zar, "Variability in multistage synchronizers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 12, pp. 2957–2969, December 2015.

[83] A. W. Eckford, and W. Yu, "Density evolution for the simultaneous decoding of LDPC-based Slepian-Wolf source codes," in *Proc. IEEE Int. Symp. Information Theory, (ISIT-2005)*, Adelaide, Australia, Sep 2005, pp. 1401–1405.

[84] G. Albertengo and R. Sisto, "Parallel CRC generation," *IEEE Micro*, vol. 10, no. 5, pp. 63–71, Oct 1990.

[85] Y. Huo, X. Li, W. Wang, and D. Liu, "High performance table-based architecture for parallel CRC calculation," in *Proc. IEEE Int. Workshop Local and Metropolitan Area Networks (LANMAN-2015)*, Beijing, China, April 2015, pp. 1–6.

[86] C. Janer, J. Quero, J. Ortega, and L. Franquelo, "Fully parallel stochastic computation architecture," *IEEE Trans. Signal Process.*, vol. 44, no. 8, pp. 2110–2117, Aug 1996.

[87] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE Int. Conf. Communications (ICC-95)*, vol. 2, Seattle, WA, USA, Jun 1995, pp. 1009–1013 vol.2.

[88] A. Nimbalker, Y. Blankenship, B. Classon, and T. Blankenship, "ARP and QPP interleavers for LTE turbo coding," in *Proc. IEEE Wireless Communications Networking Conf., (WCNC-2008)*, Las Vegas, NV, USA, Mar 2008, pp. 1032–1037.

[89] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *IET Electron. Lett.*, vol. 36, no. 23, pp. 1937–1939, Nov 2000.

[90] R. Dennard, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct 1974.

[91] Y. Liu, T. Zhang, and J. Hu, "Design of voltage overscaled low-power trellis decoders in presence of process variations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 439–443, March 2009.

[92] Taiwan Semiconductor Manufacturing Company Ltd, "TSMC 40nm core library databook," 2012.

Author Index