Energy-Efficient Turbo Decoder for 3G Wireless Terminals

by

Ibrahim A. Al-Mohandes

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2005

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Since its introduction in 1993, the turbo coding error-correction technique has generated a tremendous interest due to its near Shannon-limit performance. Two key innovations of turbo codes are parallel concatenated encoding and iterative decoding. In its IMT-2000 initiative, the International Telecommunication Union (ITU) adopted turbo coding as a channel coding standard for Third-Generation (3G) wireless high-speed (up to 2 Mbps) data services (cdma2000 in North America and W-CDMA in Japan and Europe).

For battery-powered hand-held wireless terminals, energy consumption is a major concern. In this thesis, a new design for an energy-efficient turbo decoder that is suitable for 3G wireless high-speed data terminals is proposed. The Log-MAP decoding algorithm is selected for implementation of the constituent Soft-Input/Soft-Output (SISO) decoder; the algorithm is approximated by a fixed-point representation that achieves the best performance/complexity tradeoff. To attain energy reduction, a two-stage design approach is adopted.

First, a novel dynamic-iterative technique that is appropriate for both good and poor channel conditions is proposed, and then applied to reduce energy consumption of the turbo decoder. Second, a combination of architectural-level techniques is applied to obtain further energy reduction; these techniques also enhance throughput of the turbo decoder and are area-efficient. The turbo decoder design is coded in the VHDL hardware description language, and then synthesized and mapped to a 0.18μm CMOS technology using the standard-cell approach. The designed turbo decoder has a maximum data rate of 5 Mb/s (at an upper limit of five iterations) and is 3G-compatible. Results show that the adopted two-stage design approach reduces energy consumption of the turbo decoder by about 65%.

A prototype for the new turbo codec (encoder/decoder) system is implemented on a Xilinx XC2V6000 FPGA chip; then the FPGA is tested using the CMC Rapid Prototyping Platform (RPP). The test proves correct functionality of the turbo codec implementation, and hence feasibility of the proposed turbo decoder design.

Acknowledgements

I would like to first thank my thesis supervisor, Prof. Mohamed Elmasry. It was through his patience and invaluable guidance that this work was accomplished. I would also like to express my appreciation for Prof. Amir Khandani, Prof. Catherine Gebotys, Prof. Manoj Sachdev, and Prof. Mohamed Sawan for serving as my committee members and commenting on this work.

I would like to acknowledge the support of our VLSI lab system administrator, Phil Regier, especially his fast response to technical problems. I would also like to thank my colleagues in the VLSI lab for their help and advise. Special thanks go to Jonathan Lutz, now with Motorola Co.

Finally, I send my sincere thanks and great appreciation to my parents and my wife for their love and support during this work.

Contents

List of Tables

List of Figures

List of Acronyms

3G	Third-Generation
ACS	Add-Compare-Select
APP	A Posteriori Probability
ARQ	Automatic Repeat Request
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BM	Branch Metric
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CDMA	Code Division Multiple Access
cdma2000	3G CDMA standard (for year 2000) in North America
CRC	Cyclic Redundancy Check
DHDD	Decrease in Hard Decision bit-wise Difference
FEC	Forward Error Correction
FER	Frame Error Rate
HD	Hard Decision
HDA	Hard Decision Aided
HDD	Hard Decision bit-wise Difference
HSDPA	High Speed Downlink Packet Access
IMT-2000	International Mobile Telecommunications for year 2000
ITU	International Telecommunication Union

LLR	Log-Likelihood Ratio
MAP	Maximum A Posteriori
ML	Maximum Likelihood
RPP	Rapid Prototyping Platform
RSC	Recursive Systematic Convolutional
RTL	Register-Transfer-Level
SCR	Sign-Change Ratio
SISO	Soft-Input/Soft-Output
SM	State Metric
SNR	Signal-to-Noise Ratio
SOVA	Soft-Output Viterbi Algorithm
W-CDMA	3G (Wideband) CDMA standard in Europe and Japan

List of Symbols

L	A priori, or extrinsic, information
L_a	A priori information
β	Backward state metric
E_b	Bit energy
γ_i	Branch metric for bit $i \in \{0, 1\}$
B	Channel bandwidth
C	Channel capacity
L_c	Channel reliability factor
η	Channel spectral efficiency
R	Code rate
f_c	Correction function for Jacobian logarithm
L_e	Extrinsic information
α	Forward state metric
N	Frame (or block) length
$\mathbf{g}$	Generator polynomial for convolutional code
$\widehat{\mathbf{x}}$	Hard decision for original $\mathbf{x}$
$\widetilde{\mathbf{x}}$	$\mathbf{x}$ interleaved
Λ	Log-likelihood ratio (or probability)
N_0	One-sided noise power spectral density
E_s	Symbol energy

Chapter 1

Introduction

For many digital communication services, bandwidth and transmission power are limited resources, and it is well known that the use of Forward Error-Correction (FEC) codes plays a fundamental role in increasing power and spectrum efficiency. However, Shannon demonstrated in [1] that the development of error-correction techniques with increasing coding gain has a limit arising from the channel capacity.

Since then, FEC code designers have been looking for new codes that approach as close as possible the Shannon limit. However, each increased coding gain comes at the expense of decoder complexity, and its practical feasibility must be evaluated for the available technologies [2].

A new class of binary parallel concatenated Recursive Systematic Convolutional (RSC) codes, called *turbo codes* [3], are capable of achieving power efficiency close to the Shannon limit. Turbo codes have been adopted by the International Telecommunication Union (ITU) to effectively improve system capacity for Third-Generation (3G) wireless high-speed data services (cdma2000 and W-CDMA).

The goal of the ITU is to achieve a harmonized 3G wireless standard that would allow users to roam anywhere in the world without resorting to multimedia terminals. Despite being a small part of the overall system, turbo code specifications in the cdma2000 and W-CDMA systems are designed to have as much commonality as possible toward achieving this goal [4].

Now, communication system designers have a large spectrum of turbo-code decoders

at their disposal. However, performance and power are usually contradicting metrics; the decoder with an excellent decoding performance also has a very complex hardware architecture that results in a large amount of power consumption [5].

By applying high-performance power-reduction techniques in the design of the turbo decoder, energy consumption of the mobile terminal can be reduced in two ways. The first one is directly achieved by reducing energy consumption of the turbo decoder, thereby reducing energy consumption of the entire mobile terminal. The second way is indirectly derived from the fact that a high performance decoder can decode codes transmitted with low signal-to-noise ratios (SNR); this allows for the reduction of power emitted by the transceiver which results in a further energy reduction of the entire mobile terminal. Moreover, the reduction in energy consumption permits the reduction in mobile terminal size due to three factors. The first one is the use of small-size batteries, which is now possible because of the reduction in energy consumption. The other two factors include the reduction in both of the turbo decoder silicon area (due to lower complexity) and the antenna size (due to lower transmission power). In addition to decoding performance, energy, and area/size factors, throughput and latency are also considered when applying power-reduction techniques. When the optimizations of these factors are contradictory, the best possible tradeoff has to be chosen.

1.1 Thesis Contributions

In this thesis, a new energy-efficient design of a 3G-compliant turbo decoder is proposed. A two-stage design approach is adopted:

1. A novel low-complexity dynamic-iterative technique that reduces energy of the turbo decoder at both good and poor channel conditions is proposed.

2. A combination of architectural-level techniques is applied for further energy reduction; the techniques also enhance throughput of the turbo decoder and are area-efficient. One of the applied techniques is a novel operator reduction method that reduces power, area, and critical-path delay of the turbo decoder.

The new energy-efficient turbo decoder is coded in VHDL, and then synthesized into 0.18μm CMOS, achieving a maximum data rate of 5 Mb/s (with an upper limit of 5 iterations). Results show an energy consumption reduction of about 65% (compared to a basic implementation), with an energy efficiency of about 4.5 nJ/b/iteration.

To prove feasibility of the proposed turbo decoder design, a prototype is implemented for the turbo codec (encoder/decoder) on a Xilinx XC2V6000 FPGA. The Xilinx FPGA is tested using the CMC Rapid Prototyping Platform (RPP) and found to be functionally correct.

1.2 Organization

The thesis is organized into eight chapters. Chapter 2 provides an introduction to turbo codes as an efficient error-control coding for transmission over noisy channels. The role of turbo codes in 3G wireless systems is also presented. In Chapter 3, turbo decoder algorithms and design techniques are discussed, with an emphasis on iteration reduction and energy consumption. In Chapter 4, a novel dynamic-iterative technique (CRC-HDD) is introduced. A fixed-point approximation of the Log-MAP algorithm is also presented. Chapter 5 discusses architectural-level techniques suggested for improving area, throughput, and energy efficiency of the turbo decoder. In Chapter 6, key results from a 0.18μm CMOS implementation of the turbo decoder are detailed. In Chapter 7, the design and testing of an FPGA prototype for the new turbo codec are presented. Chapter 8 summarizes the results and concludes the thesis, along with suggestions for future work.

Chapter 2

Turbo Codes

Turbo coding is used as a FEC technique for transmission over noisy channels. Turbo codes have been used over the last decade in different wireless communication systems such as Code Division Multiple Access (CDMA), deep space, and satellite communication systems. In the IMT-2000 standard, turbo coding has been chosen by the ITU as a channel coding technique for 3G wireless high-speed (up to 2 Mbps) data services.

This chapter serves as a background for turbo coding as a channel coding technique. Also, the standardized turbo codes for 3G wireless communication systems are described.

2.1 Digital Communication Systems

To understand the role of turbo coding as an error-control technique, the general model of a digital communication system is shown in Figure 2.1. This system consists of three major parts: *transmitter*, *channel*, and *receiver*. Each of the transmitter and the receiver has its own components.

The task of the *transmitter* is to transform the information generated by a source into a form that can withstand the effects of noise over the communication channel. An *information source* generates message-bearing information (such as words and code symbols) to be transmitted. The *source encoder* converts the information source output to a sequence of binary digits with minimum redundancy. If the source encoder generates r_b bits per second (bps), r_b is called the *data rate*.

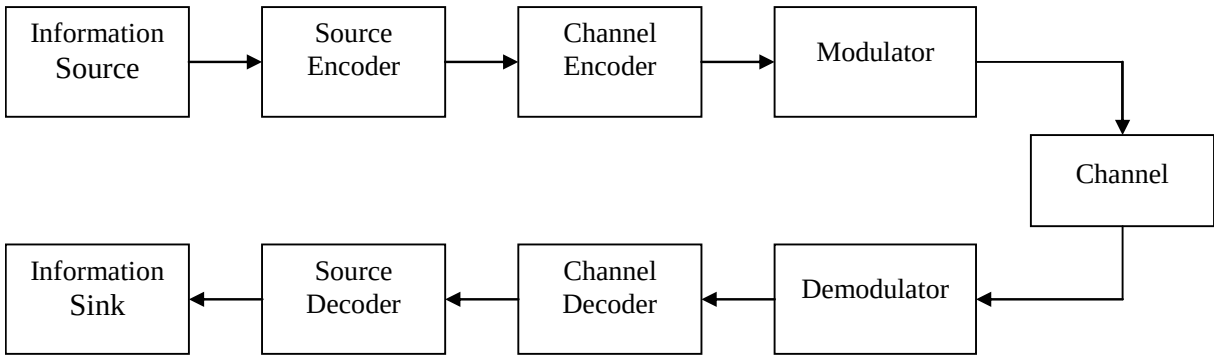


Figure 2.1: The general model of a digital communication system

Since channel impairments cause errors in the received signal, the *channel encoder* is incorporated to add redundancy to the information sequence. This redundancy is used to minimize transmission errors. The channel encoder assigns to each message of k bits a longer message of n bits called a *codeword*. A good error-control code generates codewords which are as different as possible from each other. This makes the communication system less vulnerable to channel errors. Each code word is characterized by a ratio $R = k/n < 1$, called the *code rate*. The data rate at the output of the channel encoder is $r_c = r_b/R$ bps. The primary goal of error-control coding is to maximize the reliability of transmission within the constraints of signal power, system bandwidth, and circuit complexity. It is achieved by introducing structured redundancy into transmitted signals. This usually results in a lowered data transmission rate, or an increased channel bandwidth, relative to an uncoded system [6].

The channel encoder output is not normally suitable for transmission. Therefore, the *modulator* is incorporated to match the signal to the channel, to enable simultaneous transmission of a number of signals over the same physical channel, and to increase the speed of information transmission. The modulator maps the encoded digital sequences into a train of short analog waveforms that are appropriate for propagation. An M-ary modulator maps a block of l binary digits from the channel encoder into one of M possible waveforms, where $M = 2^l$. The duration of the modulator output waveform is T sec and is referred to as the signaling interval, whereas $r_s = 1/T$ is called the *symbol rate*.

The minimum signal bandwidth is equal to r_s Hz, where $r_s = r_b/Rl$. Modulation can be performed by varying amplitude, phase, and/or frequency of a sinusoidal waveform called the *carrier*.

Channels are transmission media used to carry or store information. Channel examples include wire lines, microwave radio links over free space, satellite links, fibre optic channels, and magnetic recording media. Very often the term *channel* is used to refer to the frequency range allocated to a particular service such as television or phone channels. Two major limitations of real channels are *thermal noise* and *finite bandwidth*. In addition, mobile radio channels suffer from multipath propagation, fibre optic cables suffer from signal dispersion, and magnetic media are exposed to duct and physical damage.

In the *receiver*, the *demodulator* typically generates a binary or analog sequence at its output as the best estimates of the transmitted codeword or the modulated sequence respectively. The *channel decoder* makes estimates of the actually transmitted message. The decoding process is based on the encoding rule and the characteristic of the channel. The goal of the decoder is to minimize the effects of channel noise. The *source decoder* transforms the bit sequence generated by the decoder into an estimate of the source output sequence and delivers it to the user (*information sink*).

If the demodulator generates a binary sequence, subsequent channel decoding is called *hard decision decoding*. In this case, the three blocks of modulator, channel, and demodulator can be simplified by a *discrete channel*. The input and output of the discrete channel are binary sequences at r_c bits per sec. If the demodulator output in a given symbol interval depends on the current transmitted signal and not any previous transmission, the channel is said to be *memoryless*. If this memoryless channel has equal error probabilities for the binary symbols 0 and 1, it is called a Binary Symmetric Channel (BSC).

If the demodulator output is quantized into more than two discrete levels or samples are taken from the analog received baseband signal, the subsequent decoding process is called *soft decision decoding*. Hard decisions result in a more irreversible information loss than soft decisions.

If bandwidth efficiency is essential, combining coding and modulation into a single entity obtains a more effective signal design. This results in increased noise immunity of the signal without increasing the channel bandwidth. The combined coding and modulation is called

trellis coded modulation [6].

2.1.1 Channel Fundamental Limits

For a given channel there is an upper limit on the data rate related to the Signal-to-Noise Ratio (SNR) and the system bandwidth. Shannon has introduced the concept of *channel capacity*, C, as the maximum rate at which information can be transmitted over a noisy channel [1]. For an Additive White Gaussian Noise (AWGN) channel, channel capacity is given by the following formula:

$$C = B \log_2(1 + \frac{S}{N}) \quad \text{bits/sec} \tag{2.1}$$

where B is the channel bandwidth in Hz, and S/N is the average SNR, defined as

$$\frac{S}{N} = \eta \frac{E_b}{N_0} \tag{2.2}$$

where E_b/N_0 is the *bit energy to one sided noise power spectral density*, and η is the *spectral efficiency*, defined as

$$\eta = \frac{r_b}{B} \quad \text{bits/sec/Hz} \tag{2.3}$$

Another important parameter is the *power efficiency* defined as the required E_b/N_0 to achieve a specified *bit error probability*.

Shannon's channel coding theorem guarantees the existence of codes that can achieve an arbitrary small probability of error if the data transmission rate r_b is smaller than the channel capacity. Conversely, for a data rate $r_b > C$, it is not possible to design a code that can achieve an arbitrary small error probability.

This fundamental result shows that noise sets a limit on the data rate but not on the error probability, as widely believed before. Although the theorem does not indicate how to design specific codes that achieve maximum possible data rate at an arbitrary small error probability, it has motivated the development of a number of error-control techniques [6].

By substituting S/N from (2.2) into (2.1) and observing that $\eta_{\max} = C/B$, the minimum required E_b/N_0 for an error-free transmission is given by

$$\frac{E_b}{N_0} \geq \frac{2^\eta - 1}{\eta} \tag{2.4}$$

If the bandwidth is not limited (*i.e.*, $B \to \infty$ or $\eta_{\max} \to 0$), then

$$\lim_{\eta_{\max} \to 0} \frac{E_b}{N_0} = \ln 2 = -1.59 \quad \text{dB} \tag{2.5}$$

Hence, theoretically, the minimum required E_b/N_0 for error-free transmission is -1.59 dB [6, 7, 8].

2.2 Channel Coding Techniques

Various error-control codes have been used in the channel coding and decoding stages of wireless communication applications (*e.g.*, deep space, CDMA, and satellite communications). Two main categories are linear block codes and convolutional codes. Other codes are derived from these two categories, and include serial concatenated codes and turbo codes (also called parallel concatenated convolutional codes).

The strength of an error-control code is measured by its *coding gain*. For a coded system, coding gain is defined as the reduction in SNR over an uncoded system to achieve the same Bit Error Rate (BER).

2.2.1 Linear Block Codes

An (n, k) *block encoder* transforms a message of k bits into a message of n bits. The important feature of a block code is that a *codeword* depends only on the current input message and not on any previous message; *i.e.*, the encoder is a *memoryless* device. In an (n, k) block code, there are 2^k distinct messages. Since there is a one-to-one correspondence between a message and a codeword, there are also 2^k distinct codewords. The code rate $R = k/n$ determines the amount of redundancy.

An (n, k) block code is *linear* if

- the component-wise modulo-2 sum of two codewords is another codeword, and
- the code contains the all-zero codeword.

A linear *systematic* block code has the additional feature that the message itself is part of the codeword. In addition to the k-digit message sequence the codeword contains an

$(n-k)$-digit parity check sequence. This format allows the direct extraction of the message from the codeword.

The *hamming distance* between two codewords is defined as the number of places in which these codewords differ. The *minimum hamming distance* or *minimum distance* of a code is defined as the smallest hamming distance between any two different codewords in the code. This implies that for a linear block code, the minimum distance is the smallest *weight* (number of ones in a codeword) of the nonzero codewords in the code. The minimum distance parameter determines the error correction and detection capability of a code.

2.2.2 Convolutional Codes

An (n, k, m) *convolutional encoder* has k input bits, n output bits, and m memory elements (m-bit shift register). Each output bit is the modulo-2 sum (XOR operation) of the current input bit and some or all of the previous m input bits. Then, the n output bits are multiplexed to produce the codeword. A *systematic* convolutional encoder has the additional feature of producing the input message as part of the output codeword. The structure of a rate 1/2 non-systematic non-recursive convolutional encoder is illustrated in Figure 2.2.

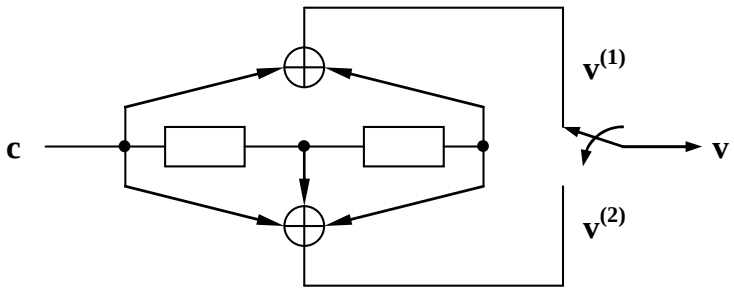


Figure 2.2: A rate 1/2 convolutional encoder

At time l, the input to the encoder is c_l and the output is a code block,

$$\mathbf{v}_l = (v_l^{(1)} v_l^{(2)})$$

The connections between the shift register elements and the modulo-2 adders can be described by generator sequences or generator polynomials

$$\mathbf{g}^{(1)} = (g_0^{(1)} g_1^{(1)} g_2^{(1)}) = (101)_2 = (5)_8 \quad \text{or} \quad \mathbf{g}_1(D) = 1 + D^2$$

$$\mathbf{g}^{(2)} = (g_0^{(2)} g_1^{(2)} g_2^{(2)}) = (111)_2 = (7)_8 \quad \text{or} \quad \mathbf{g}_2(D) = 1 + D + D^2$$

In other words,

$$\mathbf{g} = (\mathbf{g}^{(1)}, \mathbf{g}^{(2)}) = (5, 7)_8 \quad \text{or} \quad \mathbf{g}(D) = [\mathbf{g}_1(D) \;\; \mathbf{g}_2(D)] = [1 + D^2 \quad 1 + D + D^2]$$

The term *convolutional code* comes from the observation that the i^{th} output sequence, where $i = 1, 2$, represents the convolution of the input sequence and the i^{th} generator sequence

$$\mathbf{v}^{(i)} = \mathbf{c} * \mathbf{g}^{(i)}, \qquad i = 1, 2$$

where $*$ denotes the convolution operator.

The encoder state transitions can be represented graphically by a *trellis diagram*. A trellis diagram is derived from the encoder *state diagram* by tracing all possible input/output sequences and state transitions. The encoder is *trellis terminated* if the final state is the same as the initial state for a specific frame length N. Figure 2.3 shows the trellis diagram of the convolutional encoder described in Figure 2.2, assuming trellis termination with $N = 6$.

2.3 Turbo Codes

Turbo codes, also known as Parallel Concatenated Convolutional Codes (PCCC) [9], and serial concatenated codes [10] concatenate two codes to achieve a good tradeoff between coding gain and decoding complexity.

Serial Concatenated Convolutional Codes (SCCC) use two codes in series separated by an *interleaver*. This approach has been adopted in space communications, with convolutional code as the inner code and low redundancy Reed-Solomon block code as the outer code. The primary reason for using a concatenated code is to achieve a low error rate with an overall decoding complexity that is lower than the one required for a single code with

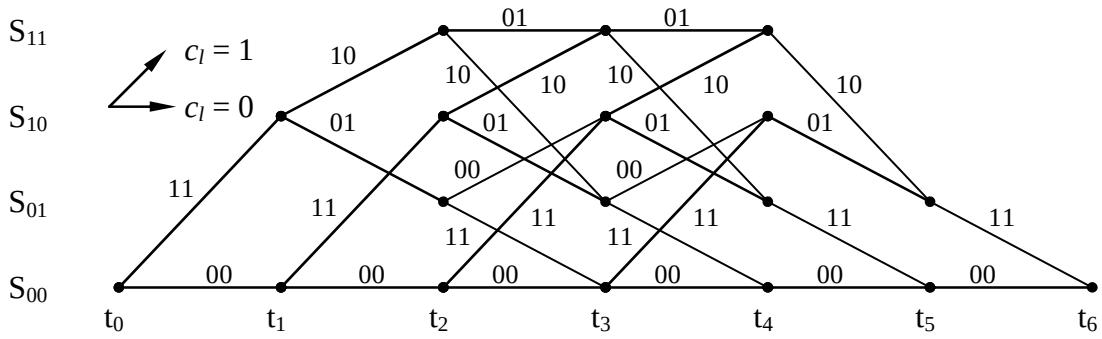


Figure 2.3: Trellis diagram for the 1/2 non-systematic encoder of Figure 2.2 (trellis terminated, frame length $N = 6$)

the same decoding performance. An interleaver is incorporated between the two codes to decorrelate the received symbols that are affected by burst errors generated by the inner decoder.

Turbo codes exploit a similar idea of connecting two codes and separating them by an interleaver [3]. The difference between turbo codes and serial concatenated codes is that, in turbo codes, two identical RSC codes are connected in parallel in the *turbo encoder*. Also, a long interleaver is used in turbo encoders to generate a concatenated code with a long block length, leading to a large coding gain. The *turbo decoder* consists of two RSC *component decoders* separated by *interleavers* and *deinterleavers*. The component decoders are based on a Soft-Input/Soft-Output (SISO) decoding algorithm, such as the Soft-Output Viterbi Algorithm (SOVA) or the Maximum A Posteriori (MAP) probability algorithm. A number of *iterations* are required by the turbo decoder to produce a BER as low as $10^{-5} - 10^{-7}$ at an SNR close to the Shannon capacity limit [6].

2.3.1 Turbo Encoder

A turbo encoder is formed by parallel concatenation of two RSC encoders separated by a random interleaver [3]. The encoder structure is called parallel concatenation because the two encoders operate on the same set of input bits, rather than one encoder encoding the output of the other. A block diagram of a rate 1/3 turbo encoder is shown in Figure 2.4.

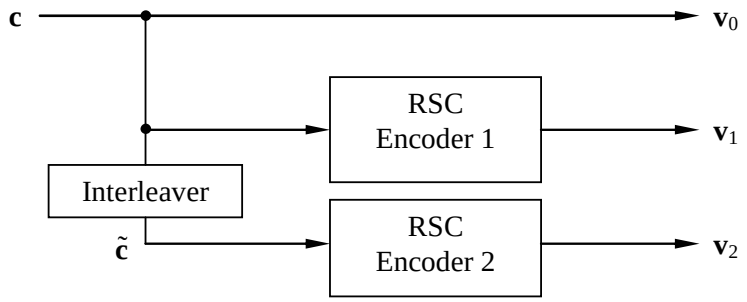


Figure 2.4: General diagram of a rate 1/3 turbo encoder

The generator matrix of a rate 1/2 constituent RSC code can be represented as

$$\mathbf{g}(D) = [1 \quad \frac{\mathbf{g}_1(D)}{\mathbf{g}_0(D)}]$$

where $\mathbf{g}_0(D)$ and $\mathbf{g}_1(D)$ are respectively feedback and feedforward polynomials with degree m. In the encoder, the same information sequence is encoded twice but in a different order. The first RSC component encoder operates directly on the input sequence $\mathbf{c}$, of length N, and has two outputs. The first output $\mathbf{v}_0$ is equal to the input sequence $\mathbf{c}$ since the encoder is systematic. The other output is the first parity check sequence $\mathbf{v}_1$. The second RSC encoder accepts the interleaved information sequence $\tilde{\mathbf{c}}$ as input. Only the parity check sequence $\mathbf{v}_2$ of the second encoder is transmitted. The information sequence and the parity check sequences of the two RSC component encoders are multiplexed to generate the turbo code sequence. Outputs from both RSC constituent encoders can be punctured and/or repeated to produce different code rates other than the direct 1/3 code rate.

For turbo encoders, either or both of the constituent RSC encoders is trellis terminated. Trellis termination means driving the encoder to the all-zero state. This is required at the end of each block to ensure that the initial state for the next block is the all-zero state.

2.3.2 Interleaver

The interleaver in turbo coding is a pseudo-random block scrambler defined by a permutation of N elements with no repetitions.

The first role of the interleaver is to generate a long block code from small memory convolutional codes. Secondly, the interleaver decorrelates the inputs to the two SISO component decoders so that an iterative suboptimum decoding algorithm based on information exchange between the two component decoders can be applied. If the input sequences to the two component decoders are decorrelated, there is a high probability that after correction of some of the errors by one decoder, some of the remaining errors become correctable by the other decoder.

In a pseudo-random interleaver, a block of N input bits is read into the interleaver and read out pseudo-randomly. The pseudo-random interleaving pattern must be available at the decoder as well [6].

2.3.3 Iterative Decoding of Turbo Codes

Turbo and serial concatenated codes can be decoded by either an A Posteriori Probability (APP) method or a Maximum Likelihood (ML) method. The practical importance of turbo and serial concatenated codes lies in the availability of a simple suboptimal decoding algorithm [3].

The iterative turbo decoder consists of two constituent SISO decoders serially connected via an interleaver, identical to the one in the encoder, and a corresponding deinterleaver, as shown in Figure 2.5.

The first SISO decoder takes as input the received information sequence $\mathbf{r}_0$ and the received parity sequence $\mathbf{r}_1$, both generated by the first RSC encoder. Then, the decoder generates a soft-output (extrinsic information) which is interleaved and used to produce an improved estimate of the intrinsic information sequence at the input of the second SISO decoder.

The other two inputs to the second SISO decoder are the interleaved received information sequence $\tilde{\mathbf{r}}_0$ and the received parity sequence produced by the second RSC encoder $\mathbf{r}_2$. The second decoder also produces a *soft output* (extrinsic information) which, after deinterleaving, is used to improve the estimate of the intrinsic information sequence at the input of the first SISO decoder. The decoder performance can be improved by this iterative operation, relative to a single operation of a serial concatenated decoder. The feedback loop is a distinct feature of this decoder and the term *turbo code* is derived from

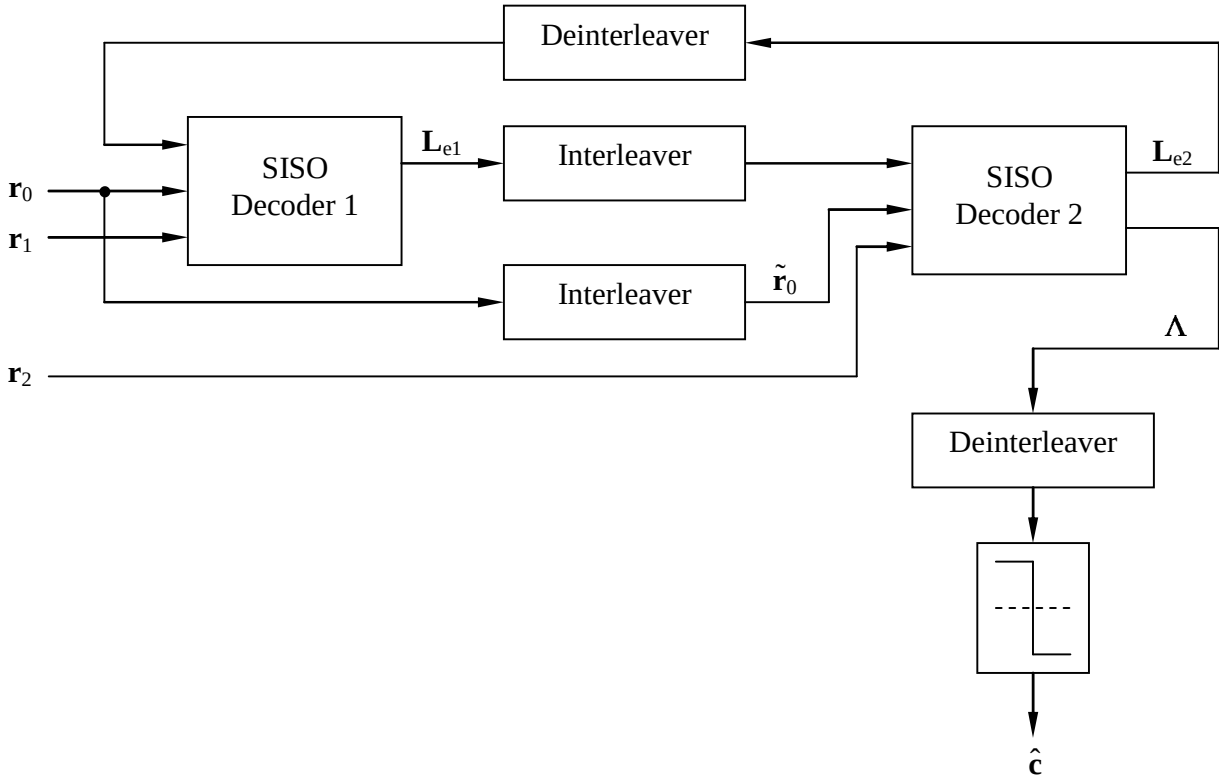


Figure 2.5: General diagram of an iterative turbo decoder

the principle of the turbo engine.

After a certain number of iterations, the soft outputs of both decoders stop making further performance improvements. Then, the second SISO decoder creates *hard decisions*, $\hat{\mathbf{c}}$, that are deinterleaved. The hard decision detector and deinterleaver can be swapped to reduce memory usage.

The two identical component decoders can be based on either the MAP, an APP decoding algorithm, or the SOVA, an ML decoding algorithm. Therefore, there are two general categories of turbo decoders in relation to component decoder type: MAP and SOVA iterative turbo decoders.

The SOVA decoder has a lower complexity than that of the optimum MAP decoder, but the MAP decoder has a better decoding performance. Modified versions of the MAP

algorithm have been developed to achieve a near-optimum performance with a much lower complexity than the original MAP algorithm.

The Max-Log-MAP replaces multiplications and exponentiations in the original MAP by additions, and comparisons by logarithmic approximations; however, this comes at the expense of degrading decoding performance. The Log-MAP algorithm employs a more accurate logarithmic approximation by using lookup tables; it gives a decoding performance better than that of the SOVA, with a small additional complexity compared to that of the Max-Log-MAP. The performance is not far from that of the original MAP [6].

Consequently, most VLSI implementations are based on the Log-MAP algorithm which achieves an excellent tradeoff between complexity (hence area and power consumption) and decoding performance.

2.4 Standardized Turbo Codes for 3G Wireless Systems

Figure 2.6 illustrates the structure of standardized turbo-codes for 3G wireless systems [4]. For both RSC encoders,

$$\mathbf{g} = (\mathbf{g}_0,\, \mathbf{g}_1,\, \mathbf{g}_2) = (1011, 1101, 1111)_2 = (13, 15, 17)_8$$

where $\mathbf{g}_0$ is the feedback polynomial, and $\mathbf{g}_1$ and $\mathbf{g}_2$ are the feedforward polynomials. The turbo code shown in Figure 2.6 is used in the cdma2000 standard [11], proposed by the TIA in USA, for high-speed (above 14.4 kbps) data services. Rate 1/2, 1/3, and 1/4 turbo codes are realized with appropriate puncturing patterns from Table 2.1.

For the UTRA/W-CDMA, proposed by ETSI in Europe and ARIB in Japan, the same constituent code is used for the rate 1/3 turbo code. Other code rates are obtained by a "rate matching" process, where coded bits are punctured or repeated accordingly [12]. The turbo code in Figure 2.6 is the result of an extensive simulation study [4].

For both of the 3G wireless systems, cdma2000 and UTRA/W-CDMA, turbo codes are terminated in a similar way. To enforce the trellis back to the all zero state, tail bits come from the contents of the shift registers, as shown by the dotted lines in Figure 2.6.

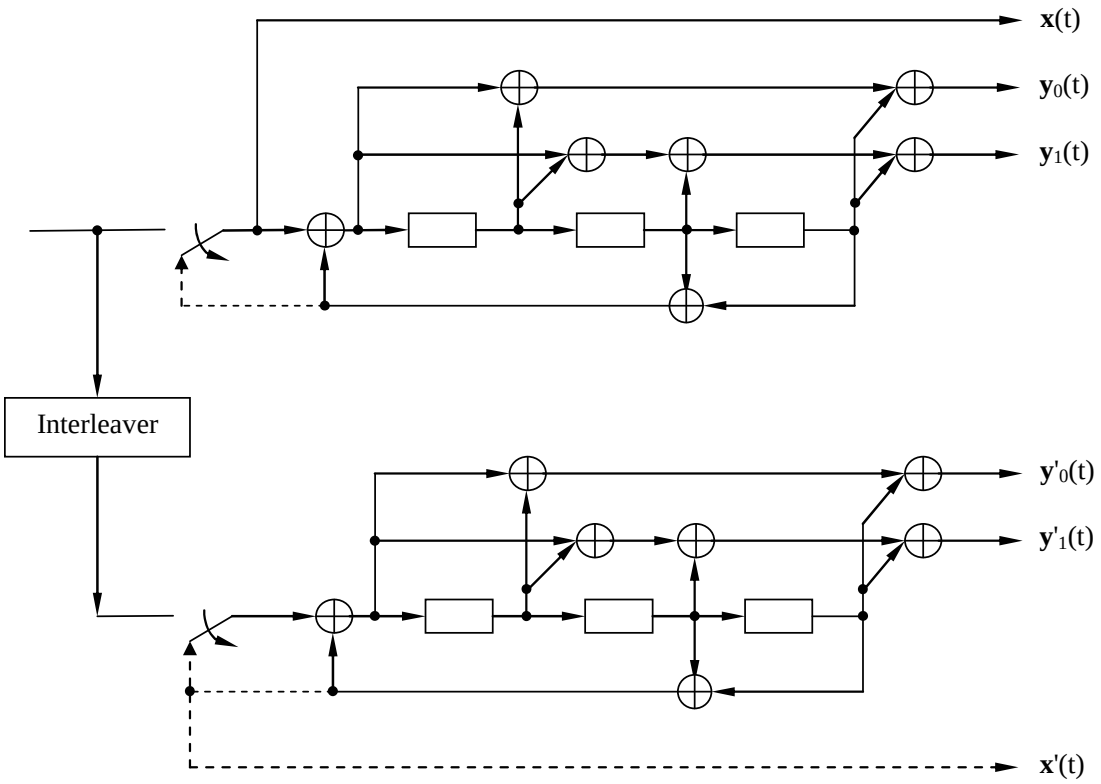


Figure 2.6: Standardized turbo code for 3G wireless systems (dotted lines effective for trellis termination only)

Moreover, because of the turbo interleaver, the contents of the shift register at the beginning of trellis termination are different for both constituent encoders. Therefore, for the standardized turbo code with eight state constituent codes, a total of $3 \times 2 = 6$ tail bits are required to terminate both encoders.

For the sake of description, it is assumed that the first three tail bits are used to terminate the upper constituent encoder, whereas the last three tail bits are used to terminate the lower constituent encoder [4].

The standardized turbo interleavers for the two systems belong to the same general class of interleavers in that they share the following properties:

1. A small number of "mother interleavers" are specified, from which interleavers of

Table 2.1: Puncturing patterns for the turbo code of Figure 2.6

Rate	1/2	1/3	1/4
$\mathbf{x}(t)$	11	11	11
$\mathbf{y}_0(t)$	10	11	11
$\mathbf{y}_1(t)$	00	00	10
$\mathbf{x}\prime(t)$	00	00	00
$\mathbf{y}_0\prime(t)$	01	11	01
$\mathbf{y}_1\prime(t)$	00	00	11

medium size are derived by *pruning* [13] unnecessary indexes. Pruning means ignoring an index that results in an invalid address because it exceeds the range of interest.

2. The mother interleavers can be viewed as two-dimensional matrices, where the entries are written into the matrix row by row and read out column by column.

3. Before reading out the entries, intra- and inter-row permutations are performed.

cdma2000 and UTRA/W-CDMA turbo interleavers differ from each other in the exact specifications of intra- and inter-row permutations and in the matrix dimensions of the mother interleavers [4].

There are several reasons why turbo codes are particularly suited for high-speed data services of 3G wireless systems:

1. At high speeds, sufficiently long blocks of data can be accumulated (*e.g.*, within a frame of 10 or 20 ms) without causing substantial delay in the system. Turbo codes become more and more effective as block (or interleaver) size increases because of *spectral thinning* (*i.e.*, the multiplicity of "neighbour" codewords becomes smaller as the interleaver size gets larger) [9].

2. For 3G high-speed data services, error-free data transmission is accomplished by Automatic Repeat Request (ARQ) protocol implemented in higher layers. As a

result, the more appropriate figure of merit is Frame Error Rate (FER), rather than BER. The performance difference between turbo and convolutional codes becomes even larger when the codes are compared in terms of FER as opposed to BER [14]. For turbo codes, however, the power of the code increases significantly as the frame size increases due to spectral thinning.

3. With fast power control, turbo codes are more effective as an FEC technique for 3G wireless systems. Indeed, without any power control, the performance advantage of turbo codes over convolutional codes decreases considerably. Power control is an important feature for the success of CDMA systems, which have been selected for 3G wireless technology.

Chapter 3

VLSI Design of Turbo Decoders

As stated in Chapter 2, turbo codes have been selected as a channel coding standard for 3G wireless high-speed data services. However, turbo decoding is a relatively complex task, hence consuming a considerable amount of area and energy of the entire mobile terminal. To obtain an efficient decoder implementation without degrading the required decoding performance, the system design space needs to be explored on multiple levels (*e.g.*, algorithmic, architectural, gate, and circuit).

3.1 Algorithmic-Level Design

As discussed in Chapter 2, there are two main algorithms that can be used in the component SISO decoders of the turbo decoder. These are the MAP decoding algorithm, based on APP probabilities, and the SOVA decoding algorithm, based on ML probabilities. Both algorithms use the iterative technique to enhance decoding performance. In addition, there are lower-complexity variations of the original MAP decoding algorithm such as Max-Log-MAP and Log-MAP.

3.1.1 MAP Decoding Algorithm

The Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm, also known as symbol-by-symbol MAP algorithm (MAP algorithm for short), is optimal for estimating the states or outputs of

a Markov process observed in white noise [15]. In the following, a brief overview of the algorithm is given[16].

Let the state of the encoder at time k be S_k, taking the values from 0 to 2^{M-1}, where M is the encoder memory order. The bit d_k is associated with the transition from step $k-1$ to step k. It is assumed that the data frame length is N, and that the encoder is trellis terminated. The goal of the MAP algorithm is to provide us with the ratio of the APP of each information bit d_k being 1 to the APP of it being 0. The following is obtained

$$\Lambda(d_k) = \ln \frac{\Pr\{d_k = 1|y_k\}}{\Pr\{d_k = 0|y_k\}} = \ln \frac{\sum_{S_k} \sum_{S_{k-1}} \gamma_1(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1}) \cdot \beta_k(S_k)}{\sum_{S_k} \sum_{S_{k-1}} \gamma_0(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1}) \cdot \beta_k(S_k)} \tag{3.1}$$

where the forward recursion of the MAP can be expressed as

$$\alpha_k(S_k) = \frac{\sum_{S_{k-1}} \sum_{i\in\{0,1\}} \gamma_i(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1})}{\sum_{S_k} \sum_{S_{k-1}} \sum_{i\in\{0,1\}} \gamma_i(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1})}$$
$$\alpha_0(S_0) = \begin{cases} 1 & \text{for} \quad S_0 = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

and the backward recursion as

$$\beta_k(S_k) = \frac{\sum_{S_{k+1}} \sum_{i\in\{0,1\}} \gamma_i(y_{k+1}, S_k, S_{k+1}) \cdot \beta_{k+1}(S_{k+1})}{\sum_{S_k} \sum_{S_{k+1}} \sum_{i\in\{0,1\}} \gamma_i(y_{k+1}, S_k, S_{k+1}) \cdot \alpha_k(S_k)}$$
$$\beta_N(S_N) = \begin{cases} 1 & \text{for} \quad S_N = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The branch transition probabilities are given by

$$\gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k] = p(y_k^s|d_k = i) \cdot p(y_k^p|d_k = i, S_k, S_{k-1}) \cdot q(d_k = i|S_k, S_{k-1}) \cdot \Pr\{S_k|S_{k-1}\}; \quad i \in \{0, 1\} \tag{3.4}$$

The value of $q(d_k = i|S_k, S_{k-1})$ is 1 if bit i is associated with the transition from state S_{k-1} to state S_k, and 0 otherwise. $\Pr\{S_k|S_{k-1}\}$ represents the a priori information of bit d_k: In case of no parallel transitions, $\Pr\{S_k|S_{k-1}\} = \Pr\{d_k = 1\}$ if $q(d_k = 1|S_k, S_{k-1}) = 1$, and $\Pr\{S_k|S_{k-1}\} = \Pr\{d_k = 0\}$ if $q(d_k = 0|S_k, S_{k-1}) = 1$.

3.1.2 Max-Log-MAP Decoding Algorithm

The MAP algorithm, in its original form, is difficult to implement because of numerical representation of probabilities and non-linear operations such as exponentiations and multiplications [16]. To avoid these problems, the logarithms of $\gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k]$, $\alpha_k(S_k)$, and $\beta_k(S_k)$ are taken instead.

By taking the logarithm of $\gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k]$ derived in (3.4) and inserting

$$\begin{aligned} p(y_k^s|d_k = i) &= \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{1}{N_0}[y_k^s - x_k^s(i)]^2} \\ p(y_k^p|d_k = i, S_k, S_{k-1}) &= \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{1}{N_0}[y_k^p - x_k^p(i, S_k, S_{k-1})]^2} \end{aligned} \tag{3.5}$$

we obtain the following expression for $q(\cdot) = 1$

$$\begin{aligned} \ln \gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k] = \frac{2y_k^s x_k^s(i)}{N_0} + \frac{2y_k^p x_k^p(i, S_k, S_{k-1})}{N_0} + \\ \ln \Pr\{S_k|S_{k-1}\} + K \end{aligned} \tag{3.6}$$

Since the constant K cancels out in the calculation of $\ln \alpha_k(S_k)$ and $\ln \beta_k(S_k)$, it can be ignored. Notice that N_0 must be estimated to correctly weigh the channel information with the a priori probability $\Pr\{S_k|S_{k-1}\}$.

For $\ln \alpha_k(S_k)$, we get

$$\begin{aligned} \ln \alpha_k(S_k) = \ln \left(\sum_{S_{k-1}} \sum_{i \in \{0,1\}} e^{\ln \gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k] + \ln \alpha_{k-1}(S_{k-1})} \right) - \\ \ln \left(\sum_{S_k} \sum_{S_{k-1}} \sum_{i \in \{0,1\}} e^{\ln \gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k] + \ln \alpha_{k-1}(S_{k-1})} \right) \end{aligned} \tag{3.7}$$

To simplify the solution, the following approximation is used:

$$\ln(e^{\delta_1} + \ldots + e^{\delta_n}) \approx \max_{i \in \{1 \ldots n\}} \delta_i \tag{3.8}$$

$\max_{i\in\{1\ldots n\}} \delta_i$ can be calculated by successively using $n-1$ maximum functions over only two values. From now on, we will use the notation: $\bar{\gamma}(\cdot) = \ln\gamma(\cdot)$, $\bar{\alpha}(\cdot) = \ln\alpha(\cdot)$, and $\bar{\beta}(\cdot) = \ln\beta(\cdot)$. Eventually, we obtain

$$\begin{aligned}\bar{\alpha}_k(S_k) = &\max_{(S_{k-1},i)}\{\bar{\gamma}_i[(y_k^s, y_k^p), S_{k-1}, S_k] + \bar{\alpha}_{k-1}(S_{k-1})\} - \\ &\max_{(S_k,S_{k-1},i)}\{\bar{\gamma}_i[(y_k^s, y_k^p), S_{k-1}, S_k] + \bar{\alpha}_{k-1}(S_{k-1})\}\end{aligned} \tag{3.9}$$

and similarly,

$$\begin{aligned}\bar{\beta}_k(S_k) = &\max_{(S_{k+1},i)}\{\bar{\gamma}_i[(y_{k+1}^s, y_{k+1}^p), S_k, S_{k+1}] + \bar{\beta}_{k+1}(S_{k+1})\} - \\ &\max_{(S_k,S_{k+1},i)}\{\bar{\gamma}_i[(y_{k+1}^s, y_{k+1}^p), S_k, S_{k+1}] + \bar{\alpha}_k(S_k)\}\end{aligned} \tag{3.10}$$

The second terms are a consequence of the derivation from (3.2) and (3.3); omitting them has no effect on the value of the output of the Max-Log-MAP algorithm since these normalization terms will cancel out in (3.11). Similarly, an approximation of the log-likelihood reliability of each bit d_k can be given as follows

$$\begin{aligned}\Lambda(d_k) \approx &\max_{(S_k,S_{k-1})}\{\bar{\gamma}_1[(y_k^s, y_k^p), S_{k-1}, S_k] + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)\} - \\ &\max_{(S_k,S_{k-1})}\{\bar{\gamma}_0[(y_k^s, y_k^p), S_{k-1}, S_k] + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)\}\end{aligned} \tag{3.11}$$

To be used in a turbo decoder, the output of the Max-Log-MAP algorithm, $\Lambda(d_k)$, is split into three terms (extrinsic, a priori, and systematic components). We begin with defining

$$\bar{\gamma}_i\prime(y_k^p, S_{k-1}, S_k) = \ln p(y_k^p|d_k = i, S_k, S_{k-1}) + \ln q(d_k = i|S_k, S_{k-1}) \tag{3.12}$$

By inserting this into (3.11), we obtain

$$\begin{aligned}\Lambda(d_k) \approx &[\max_{(S_k,S_{k-1})}\{\bar{\gamma}_1\prime(y_k^p, S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)\} + \\ &\ln p(y_k^s|d_k = 1) + \ln\Pr\{d_k = 1\}] - \\ &[\max_{(S_k,S_{k-1})}\{\bar{\gamma}_0\prime[y_k^p, S_{k-1}, S_k] + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)\} + \\ &\ln p(y_k^s|d_k = 0) + \ln\Pr\{d_k = 0\}]\end{aligned} \tag{3.13}$$

which can be written as

$$\begin{aligned}\Lambda(d_k) \approx & \max_{(S_k,S_{k-1})}\{\bar{\gamma}_1\prime(y_k^p, S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)\} - \\ & \max_{(S_k,S_{k-1})}\{\bar{\gamma}_0\prime[y_k^p, S_{k-1}, S_k] + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)\} + \\ & \frac{4y_k^s}{N_0} + L(d_k) \end{aligned} \tag{3.14}$$

The first two terms comprise the so-called extrinsic information, the third term is the systematic component, and the last term is the a priori component. The extrinsic information is easily obtained by subtracting the systematic and the a priori components from the output Log-Likelihood Ratio (LLR), $\Lambda(d_k)$. The extrinsic information of the current decoding stage will be used as the a priori information, $L(d_k)$, in the next decoding stage.

We need to determine the a priori information, $\ln \Pr\{S_k|S_{k-1}\}$, in (3.6). If $q(d_k = 1|S_k, S_{k-1}) = 1$, then

$$L(d_k) = \ln \frac{Pr\{d_k = 1\}}{Pr\{d_k = 0\}} = \ln \frac{Pr\{S_k|S_{k-1}\}}{1 - Pr\{S_k|S_{k-1}\}} \tag{3.15}$$

hence, $\ln \Pr\{S_k|S_{k-1}\} = L(d_k) - \ln(1 + e^{L(d_k)})$. Using (3.8), this can be approximated to

$$\ln \Pr\{S_k|S_{k-1}\} \approx L(d_k) - \max[0, L(d_k)] \tag{3.16}$$

If $q(d_k = 0|S_k, S_{k-1}) = 1$, then

$$L(d_k) = \ln \frac{Pr\{d_k = 1\}}{Pr\{d_k = 0\}} = \ln \frac{1 - Pr\{S_k|S_{k-1}\}}{Pr\{S_k|S_{k-1}\}} \tag{3.17}$$

hence, $\ln \Pr\{S_k|S_{k-1}\} = -\ln(1 + e^{L(d_k)})$. Similarly, it can be approximated to

$$\ln \Pr\{S_k|S_{k-1}\} \approx -\max[0, L(d_k)] \tag{3.18}$$

3.1.3 Log-MAP Decoding Algorithm

Because of the approximation in (3.8), the Max-Log-MAP algorithm is sub-optimal and yields an inferior soft-output than that of the MAP algorithm. The problem is to exactly

calculate $\ln(e^{\delta_1} + \ldots + e^{\delta_n})$. This problem can be solved by using the Jacobian logarithm [16]

$$\ln(e^{\delta_1} + e^{\delta_2}) = \max(\delta_1, \delta_2) + \ln(1 + e^{-|\delta_2 - \delta_1|}) = \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|) \tag{3.19}$$

where $f_c(\cdot)$ is a correction function. By using recursion, it can be proven that

$$\begin{aligned} &\ln(e^{\delta_1} + \ldots + e^{\delta_n}) = \ln(e^{\delta} + e^{\delta_n}) = \max(\delta, \delta_n) + f_c(|\delta_n - \delta|), \\ &\text{where} \quad \delta = \ln(e^{\delta_1} + \ldots + e^{\delta_{n-1}}) \end{aligned} \tag{3.20}$$

When deriving the Log-MAP algorithm, all maximizations over two values are augmented with the correction function. As a consequence, by correcting, at each step, the approximation made by the Max-Log-MAP, we have preserved the original MAP algorithm.

By calculating $f_c(\cdot)$, we lose some of the low complexity of the Max-Log-MAP algorithm. The correction function in (3.19) can be implemented using a lockup table. It is found in [16] that excellent results can be obtained with eight stored values and $|\delta_2 - \delta_1|$ ranging between 0 and 5.

3.1.4 SOVA Decoding Algorithm

The Viterbi algorithm (VA), in its MAP form, is described in [17]. It searches for the i^{th}-state sequence $\mathbf{S}^{(i)}$, and thus the desired information sequence $\mathbf{d}^{(i)}$, by maximizing over i the APP

$$P(\mathbf{S}^{(i)}|\mathbf{y}) = p(\mathbf{y}|\mathbf{S}^{(i)}) \cdot \frac{P(\mathbf{S}^{(i)})}{p(\mathbf{y})} \tag{3.21}$$

Since $\mathbf{y}$ is fixed, the following is equivalently maximized:

$$P(\mathbf{y}|\mathbf{S}^{(i)}) \cdot P(\mathbf{S}^{(i)}) \tag{3.22}$$

The maximization is realized in the code trellis, when for each state s and each time k, the path with the largest probability, *i.e.*, ML, $p(\mathbf{S}^{(i)}_{j \le k}, \mathbf{y}^{(i)}_{j \le k})$ is selected. This probability can be obtained by multiplying the branch transition probabilities associated to path i. They are $\gamma_j(s'^{(i)}, s^{(i)})$ for $1 \le j \le k$ and are similar to those defined by (3.4). Since

the maximum is not changed if the logarithm is taken, the same metric computation can be performed as described for the forward recursion of the Log-MAP algorithm. For the metric of the i^{th} path at time k,

$$M_k(s^{(i)}) = M_{k-1}(s'^{(i)}) + \frac{1}{2}L_c(y_k^s x_k^{s^{(i)}} + y_k^p x_k^{p^{(i)}}) + \frac{1}{2}L(d_k)x_k^{s^{(i)}}$$
$$\text{where} \quad L_c = 4\frac{E_s}{N_0} \quad \text{for AWGN channel} \tag{3.23}$$

This slight modification of the metric of the VA in (3.23) incorporates the a priori information about the probability of the information bits. The SOVA can be implemented in the register exchange mode or in the trace-back mode. It will be described now for the latter mode using the log-likelihood algebra [18].

It is desirable to obtain the soft output for bit $\hat{d}_k$, which the VA decides after a delay δ. The VA proceeds in the usual way by calculating the metrics for the i^{th} path using (3.23). For each state, the VA selects the path with the larger metric $M_k(s^{(i)})$. At each time $k+\delta$, the VA has selected the ML path with index i_δ and has discarded the other path with index $i_\delta{}'$ ending at this state. Along the ML path i_δ, which decides the bit $\hat{d}_k$, $\delta+1$ non-surviving paths $i_l{}'$ with indices $l = 0, \ldots, \delta$ have been discarded. The metric difference is defined as

$$\Delta_k^l = M_{k+l}(S^{(i_l)}) - M_{k+l}(S^{(i'_l)}) \geq 0 \tag{3.24}$$

It is shown in [19] that the L-value of the hard decision $\hat{d}_k$ is approximated by

$$L(\hat{d}_k) \approx \hat{d}_k \cdot \min_{l=0,\ldots,\delta} \Delta_k^l \tag{3.25}$$

Thus, we have the same hard decisions as the classical VA, and the reliability of the decisions is obtained by taking the minimum of the relevant metric differences along the ML path.

From (3.23) and (3.24), we find

$$\Delta_k^l = (M_{j<k}^{(1)} - M_{j<k}^{(2)}) + (M_{k<j<k+l}^{(1)} - M_{k<j<k+l}^{(2)}) + \frac{1}{2}L_c y_k^p (x_k^{p^{(1)}} - x_k^{p^{(2)}}) +$$
$$\frac{1}{2}L_c y_k^s [\hat{d}_k - (-\hat{d}_k)] + \frac{1}{2}L(d_k)[\hat{d}_k - (-\hat{d}_k)] \tag{3.26}$$

Therefore, the minimum value in (3.25) has the same structure. Thus, the SOVA output in its approximate version in (3.25) has the format

$$L_{\text{SOVA}}(\hat{d}_k) = L_c y_k^s + L(d_k) + \underbrace{\hat{d}_k \cdot \{\text{first 3 items in (3.26)}\}}_{L_e(\hat{d}_k)} \tag{3.27}$$

Similar to the Log-MAP algorithm, the output LLR consists of three components: systematic (first term), a priori (second term), and extrinsic information (last term). Also, extrinsic information of the current decoding stage, which is produced by subtracting the first two terms from the output LLR, is used as a priori information in the next decoding stage.

3.1.5 Comparison of MAP and SOVA Iterative Decoding Algorithms

The relationship between MAP, Log-MAP, Max-Log-MAP, and SOVA turbo decoding algorithms is illustrated in Figure 3.1 [16].

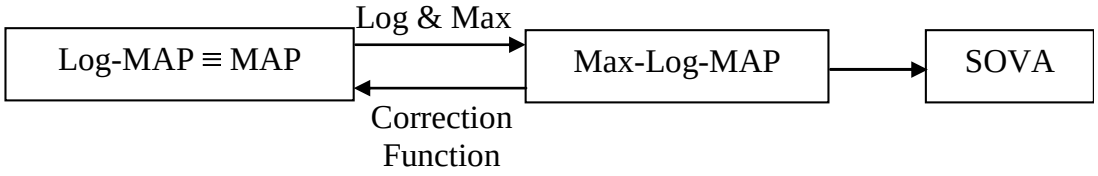


Figure 3.1: Relationship between MAP, Log-MAP, Max-Log-MAP, and SOVA

From the previous discussion, it can be seen that the Max-Log-MAP algorithm and the SOVA work with the same metric. If only the hard decisions are considered, the algorithms are identical; however, they behave in different ways in computing the information returned about the reliability of decoded bit d_k. The SOVA considers only one competing path per decoding step; *i.e.*, for each bit d_j, it does not consider all the competing paths but only the survivors of the Viterbi algorithm. To be taken into account in the reliability estimation, a competing path must join the path chosen by the Viterbi algorithm without being eliminated.

A comparison of (Log-)MAP, Max-Log-MAP, and SOVA is illustrated in Figure 3.2 [16]. The MAP includes all paths in its calculation but splits them into two sets: those that have an information bit one, at step j, and those that have an information bit zero; it returns the LLR of these two sets. The only thing that changes from step to step is the classification of the paths into the respective sets. Due to the Markov properties of the trellis, the computation can be done recursively. In contrast, the Max-Log-MAP looks at only two paths per step: the best with bit zero and the best with bit one at transition j; it then outputs the difference of the log-likelihoods. However, from step to step, one of these paths can change, but one will always be the ML path. The SOVA will always correctly find one of these two paths (the ML path), but not necessarily the other, since it may be eliminated before merging with the ML path. There is no bias on the SOVA output when compared to that of the Max-Log-MAP algorithm; only the former will be noisy.

Comparing decoding performance for the previously discussed decoding algorithms, it is found that [16]:

- The (Log-)MAP decoder is the best, followed by the Max-Log-MAP and SOVA.
- For a few iterations or for a small SNR, Max-Log-MAP and SOVA significantly degrade with respect to the (Log-)MAP.
- With an increasing number of iterations or with increasing SNR, the Max-Log-MAP approaches the (Log-)MAP.

Furthermore, for small memories the SOVA is roughly half as complex as the Log-MAP algorithm [16, 18].

Figures 3.3 and 3.4 compare the decoding performance of the four algorithmic implementations of the turbo decoder for both low and high number of iterations. Figure 3.3 shows BER vs. E_b/N_0 for the four algorithmic choices with both two and five iterations. Similarly, Figure 3.4 shows FER vs. E_b/N_0 for the four algorithmic choices of the turbo decoder with both two and five iterations. These simulations are based on a rate 1/3 turbo code with generator polynomial $\mathbf{g} = (13, 15)_8$, which is suitable for 3G wireless systems.

From the performance graphs, it is evident that the MAP and Log-MAP have better performance than the SOVA and Max-Log-MAP. The results are in agreement with the conclusions found in [16, 18], as previously discussed.

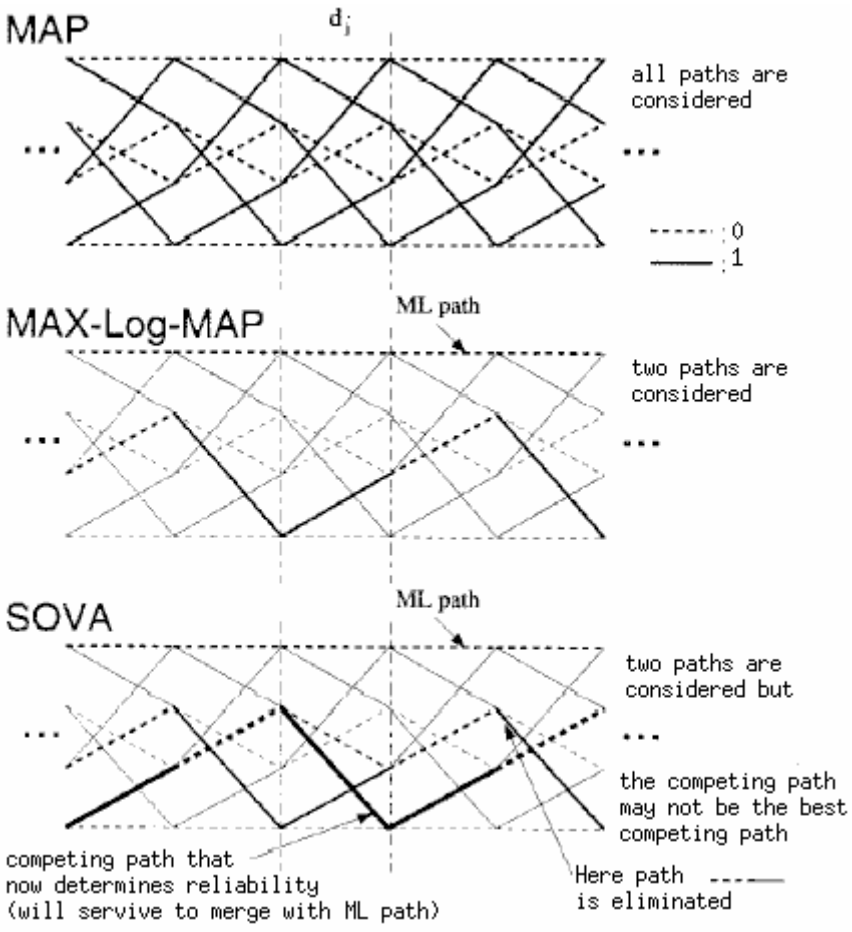
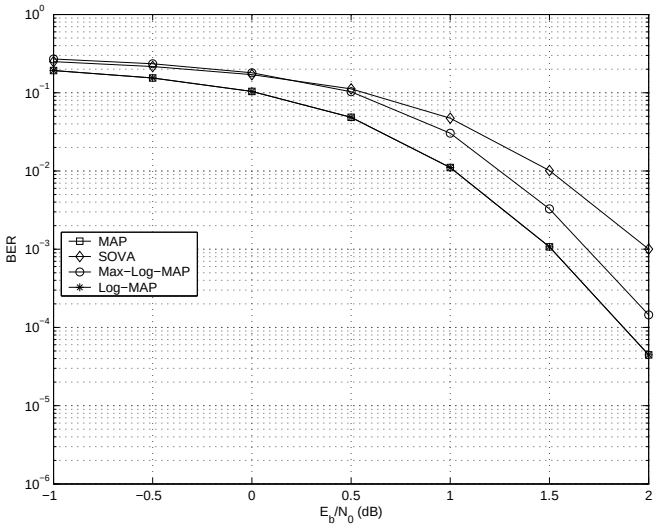
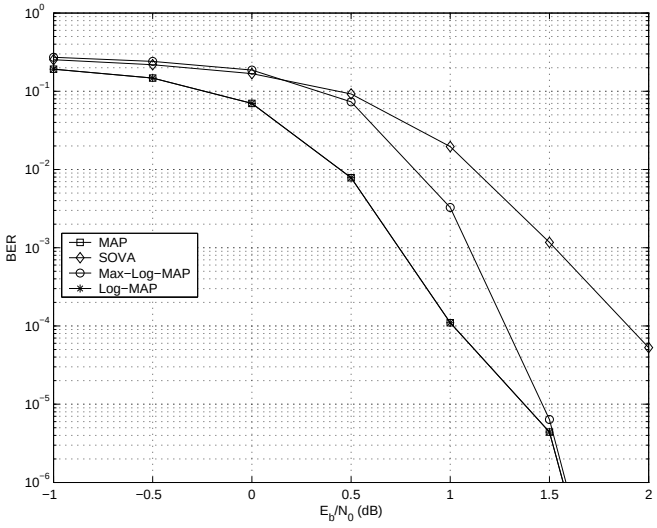


Figure 3.2: Comparing (Log-)MAP, Max-Log-MAP, and SOVA [16]

From the previous discussion, the Log-MAP algorithm is selected for implementation since it has the best tradeoff between decoding performance and decoder complexity. It has the same high decoding performance as the highly-complex original MAP; however, this comes at some additional complexity, compared to that of the low decoding performance Max-Log-MAP and SOVA.

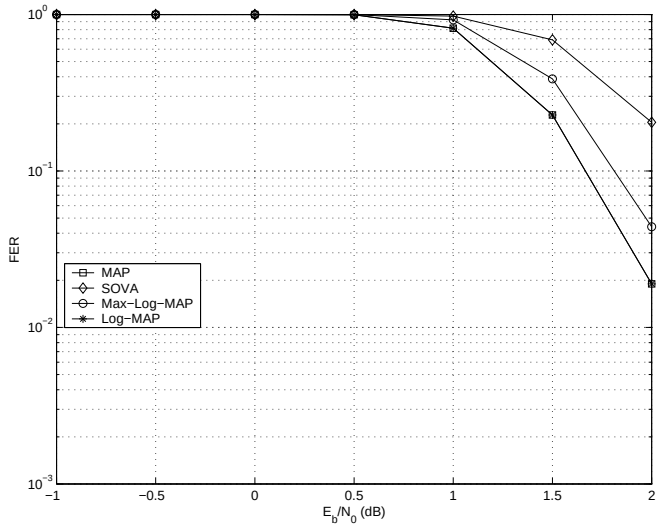


(a)

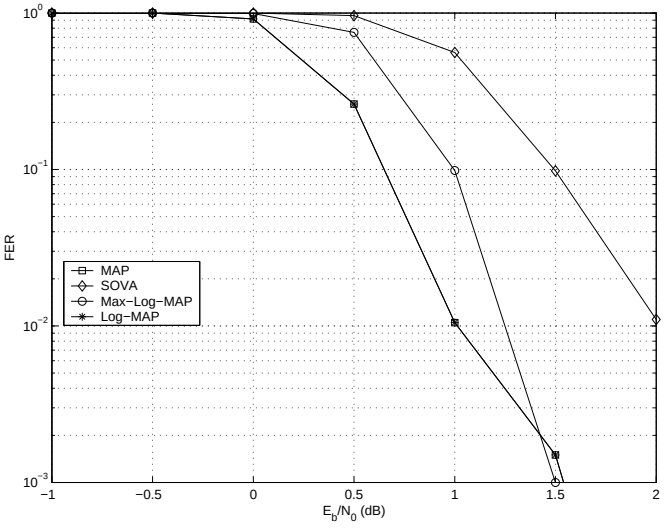


(b)

Figure 3.3: BER vs. E_b/N_0 for the MAP, SOVA, Max-Log-MAP, and Log-MAP turbo decoders with: (a) 2 iterations, (b) 5 iterations (rate 1/3, $\mathbf{g} = (13, 15)_8$, 1024 bits/frame, 2000 frames)



(a)



(b)

Figure 3.4: FER vs. E_b/N_0 for the MAP, SOVA, Max-Log-MAP, and Log-MAP turbo decoders with: (a) 2 iterations, (b) 5 iterations (rate 1/3, $\mathbf{g} = (13, 15)_8$, 1024 bits/frame, 2000 frames)

3.2 Exploration of System Design Space for Turbo Codecs

Although turbo codes have been employed in wide-band (3G) mobile radio systems, turbo decoders are relatively complex for implementation. To obtain efficient decoder implementations, the system design space needs to be explored on multiple levels. In this section, the system design space is explored with focus on the implementation-dependent part. The design decisions are rated regarding complexity, throughput, and power consumption.

Although iterative decoding is significantly less complex than optimal decoding, it remains a computationally complex task due to the iterative use of costly component decoders. Even the use of the sub-optimal Max-Log-MAP algorithm [16] for component code decoding results in considerably high computing performance needs. First order complexity estimations reveal approximately 1500 MOPS for a user data rate of 2 Mbps, assuming constraint length $K = 3$ codes and five iterations [20].

In order to achieve complexity reduction, simplifications can be attempted at different abstraction levels (*e.g.*, system, architecture, register-transfer, gate, and transistor levels). However, the optimization potential is, in general, closely related to the abstraction level. Application knowledge can be exploited to significantly simplify high-level specifications towards lower implementation complexity, where low-level design representations, in most cases, lack this opportunity. Thus, cost efficient turbo-decoder implementations require system design space exploration before mapping the algorithmic-level specification onto hardware or DSP.

Figure 3.5 depicts the system design space for turbo codecs. It is comprised of a service-dependent part and an implementation-dependent part. The components of the turbo-code encoder directly define the service dependent part of the system design space: component codes, puncturer, and interleaver. This is underlined by only the encoder being defined by standardization bodies. Although the required number of iterations is implementation-dependent, this number may also depend on the service to realize different qualities of service. The number of iterations is either static or dynamically determined during decoding after evaluation of some criteria [21].

Component RSC codes are decoded with the MAP algorithm or with the SOVA. When

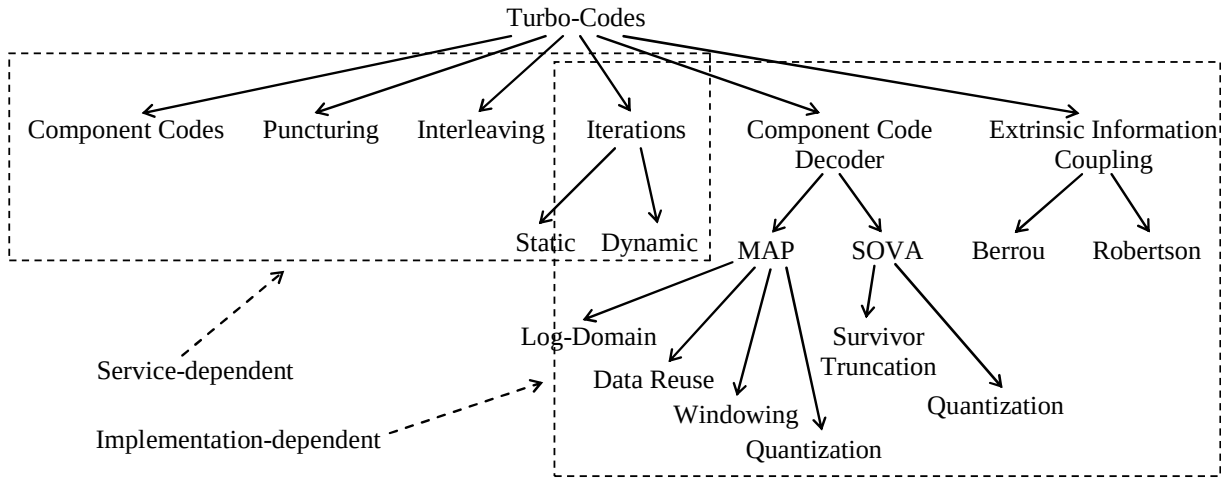


Figure 3.5: System design space for turbo codecs [20]

implementing MAP or SOVA, the designer has to choose among several implementation options which reduce computational complexity, increase throughput, and/or reduce power consumption. Extrinsic information coupling (for the feedback) is performed according to Berrou's original method [3] or rather directly as proposed by Robertson [21].

The FEC control has to sustain certain BERs for given SNRs: BER $= f(\text{SNR})$. However, implementation options influence this function. Due to the lack of a comprehensive turbo-code theory, the degradation has to be validated by simulation and traded off against implementation complexity [20]. Results of the influence of service-related parameters, such as component codes, puncturing, and interleaving, on the BER are beyond the scope of this discussion, and the focus is on implementation-dependent parameters.

3.2.1 Turbo Decoder Optimization

The complexity of the turbo decoder (TD) is a function of the component code decoder (CD) and the number of iterations (IT):

$$O_{\text{TD}} = f(\text{CD}, \text{IT}) \tag{3.28}$$

where the complexity of the component code decoder (CD) depends on operator strength (OS), amount of data reuse (DR), parallelism (P), and quantization (Q):

$$O_{\mathrm{CD}} = f(\mathrm{OS}, \mathrm{DR}, \mathrm{P}, \mathrm{Q}) \tag{3.29}$$

In this subsection, design tradeoffs with respect to these parameters are discussed. Also, extrinsic information coupling and intricacies of software implementations are briefly addressed [20]:

1. Component decoder (CD) optimization:

   (a) Operator strength (OS):

   Mainly two alternatives have been proposed for formulating the SOVA: trace-back and register exchange structures [18, 22]. These induce different implementation architectures. The superior performance of turbo-decoding with the MAP algorithm is clearly demonstrated in [23, 16]; hence, the SOVA is excluded from the following discussion.

   In addition, implementation complexities of the MAP and SOVA do not differ significantly. As per the discussion in Section 3.1 about the iterative decoding algorithms, it is found that the Log-MAP is equivalent in decoding performance to the original MAP, avoids its numerical problems, and is easier to implement due to operator strength reduction [20, 24]. Thus, from an implementation point of view, the MAP should always be implemented in the logarithmic domain. Further simplification yields the Max-Log-MAP by omitting the correction term of the Jacobian algorithm; however, this degrades the performance.

   As for the Max-Log-MAP, the inner loops of the forward and backward recursions of the Log-MAP comprise an add-compare-select (ACS) operation, but the add operation hereby additionally involves the evaluation of the correction term of the Jacobian algorithm. This correction term is best computed by using a lookup table, but this requires additional memory access and an extra logic for each ACS. The memory accesses can be traded for area (assuming a hardware implementation) if the lookup table is implemented as a combinational logic block. Either way, decoding speed decreases and power consumption increases for most target architectures.

(b) Data reuse (DR):

Calculations independent of the decoding iteration can be performed only once, and the intermediate results can be used in each MAP iteration: In case of a trellis transition, the corresponding branch metric of the (Max-)Log-MAP calculates as a sum of terms depending on the received symbols and the a priori (hence previous stage extrinsic) information [16]. Equation (3.6) is rewritten to clarify this point:

$$\bar{\gamma}_i[(y_k^s, y_k^p), S_{k-1}, S_k] = \frac{2y_k^s x_k^s(i)}{N_0} + \frac{2y_k^p x_k^p(i, S_k, S_{k-1})}{N_0} + \ln \Pr\{S_k | S_{k-1}\} \quad (3.30)$$

The a priori information $\ln \Pr\{S_k|S_{k-1}\}$ changes after each MAP iteration, whereas the terms stemming from the received symbols y_k^s and y_k^p remain constant during the entire decoding process of one data block. Therefore, pre-calculation of constant terms and performing, *e.g.*, five decoding iterations (with two SISO decoders each) saves approximately 20% of the computational complexity. The total memory size does not increase, as the pre-calculated intermediate terms replace the received symbols [20].

(c) Parallelism (P) and pipelining:

The inherent parallelism of turbo-decoders can be exploited to nearly arbitrarily tradeoff area for speed and power consumption.

On the top-most level, the component decoders can be arbitrarily pipelined (*i.e.*, functionally parallelized). The amount of additional buffer memory hereby depends on the pipeline depth.

On the component (SISO) decoder level, the functional units can be parallelized to some extent. For example, the functional units of the SISO decoder include the branch metric, forward and backward state metric, and soft-output (LLR) calculation. An obvious solution is to parallelize the branch metric with the forward state metric calculation, and the backward state metric with the soft-output calculation. Compared to a serialized solution, this approximately doubles the throughput. Adding a second branch metric calculation unit and a second soft-output calculation unit again doubles the throughput; this is described as follows. For the first half of the data block, each of the forward

and backward state metric calculation units is parallelized with a branch metric calculation unit. For the second half, each state metric calculation unit is parallelized with a soft-output calculation unit [20, 25].

Parallelism, on an additional level, is introduced by observing that the decoding of a data block in a component decoder can be divided into the decoding of a set of overlapping sub-blocks. This is called the "sliding window technique" or "windowing" [20, 25]. Windowing permits to further increase the throughput or to minimize the required memory size. For the window overlap being small compared to the window size, serial window processing reduces the required memory size by the ratio of block size to window size, while retaining throughput. In contrast, parallel window processing increases throughput by the same factor, while memory size remains constant.

On a lower level, the trellis butterflies (ACS sections) can obviously be processed in parallel during forward and backward recursion. It is quite evident that throughput, area, and power consumption increase according to the degree of parallelism.

When the parallelism of turbo-decoding is exploited, many tradeoffs are possible to obtain an optimal architecture. However, the impact of parallelism, at different levels, on memory size and structure is complex and should be carefully watched.

(d) Quantization (Q) and normalization

Quantization and normalization are ways to decrease the bit-width of a fixed-point approximation of a turbo-decoder. Each saved bit has a significant impact on area and power consumption of the decoder implementation. Memory can also be saved since the bit-width of the values is reduced. A good tradeoff between decoding performance and decoder complexity (hence area and power consumption) is desired.

2. Effect of the number of iterations (IT):

Depending on the quality of the channel and the demanded quality of service, the number of decoding iterations can be varied dynamically in order to save power.

However, the decoder has to be designed to handle the worst case. Many power saving techniques have been suggested in recent research: Some of these techniques are based on power-down mode [26] and some use voltage scaling [26] to save power consumption. Alongside one of these two techniques, there has to be a stopping criterion for further decoding iterations. Some of the stopping criteria compare soft-outputs and/or extrinsic information to some predefined thresholds, some compare hard decisions (decoded bits) of the current and previous iterations, and some use Cyclic-Redundancy-Check (CRC) checksums.

3. Regarding extrinsic information coupling, the Robertson's method is recommended over the original Berrou's method. Robertson's method is computationally less complex, performs as well, and does not require knowledge of the extrinsic information distribution parameters. For example, this saves up to 30% logic area for the FPGA implementation in [20].

4. For a software implementation of a turbo-decoder, it is essential to find a formulation of the algorithm that fits best to the given core and memory architecture, which are highly depending on the target device. This can be achieved by application of common data and control flow transformations. Since turbo-decoding is a data-flow dominated application, memory mapping and register assignment are important issues for the resulting performance. For multiple memory banks and multiport memory, parallel transfers highly increase the bandwidth. A skillful arrangement of the data within the memory simplifies the memory access order and allows easier addressing modes, leading to incremental addressing for example.

   The performance of pure software implementations is below the requirements for high data rate applications. Hardware implementations outperform software implementations, but they lack flexibility. Therefore, to achieve the high performance of hardware implementations and flexibility of software solutions, a hardware/software co-design can be used. In this mixed hardware/software implementation, performance-critical parts are implemented in hardware and flexibility-critical parts are implemented in software [20].

3.3 Architectures and Design Techniques for Turbo Decoders

In the last few years, a number of architectures and design techniques have been proposed by researchers to reduce power consumption, enhance decoding performance, reduce latency, and/or increase throughput of turbo decoders. The following is a brief overview of some of the work published to achieve such goals.

In [27], a suite of MAP-based turbo decoding algorithms with energy-quality tradeoffs for AWGN and fading channels is presented. These algorithms are derived by applying approximation techniques such as pruning the trellis, reducing the number of states, scaling the extrinsic information, applying sliding window, and early termination on the MAP-based algorithm.

In [28], a register-transfer-level (RTL) 12-bit fixed-point turbo decoder based on the Log-MAP algorithm is designed and simulated using VHDL as the hardware description language. The implemented RTL model is verified by comparing its parameters with those obtained from a C-language implementation of the same turbo decoder.

In [29], circuits and an IC implementation of a four-state, block length 16, three-metal one-poly $0.35\mu m$ CMOS analog turbo decoder with a fully programmable interleaver are presented. The IC is tested at 13.3 Mb/s, has a 1.2 μs latency, and consumes 185 mW on a single 3.3V power supply, resulting in an energy consumption of 13.9 nJ per decoded bit. The core area is $1131.2 \times 1257.9\ \mu m^2$. Mismatch simulations show that the circuits are viable for decoder lengths up to a few hundred information bits.

In [30], a radix-4 Log-MAP turbo decoder for high-speed 3G mobile data terminals is described. The Log-MAP core processes two received symbols per clock cycle using a windowed radix-4 architecture, doubling the throughput for a given clock rate over a similar radix-2 architecture. The chip is fabricated in $0.18\mu m$ CMOS, operates at a peak clock frequency of 145 MHz at 1.8V, and dissipates 956 mW when decoding continuous 10.8 Mb/s High Speed Downlink Packet Access (HSDPA) [31] data streams. Power is reduced using the 1/2-iteration hard decision assisted stopping criterion [32] (to as low as 189 mW for 10.8 Mb/s). The rate 1/3 decoder has an energy efficiency of 10 nJ/b/iteration when operating at 24 Mb/s (145 MHz). The decoder core is 14.5 mm^2 and contains 410k gates

of logic and 0.45Mb of SRAM.

In [33], the segmented sliding window approach and two other types of area-efficient parallel decoding schemes are proposed. The application of pipeline-interleaving technique to parallel turbo decoding architectures is also presented.

In [34], a 3GPP-compliant channel decoder chip supporting both data and voice calls in a unified turbo/Viterbi architecture is described. For voice services, the decoder can process over 128 voice channels encoded with rate 1/2 or 1/3, constraint length 9, convolutional codes. For data services, the turbo decoder is capable of processing any mix of rate 1/3, constraint length 4, turbo encoded data streams with an aggregate data rate of up to 2.5 Mb/s with 10 iterations per block (or 4.1 Mb/s with six iterations). The chip is fabricated in $0.18\mu m$ six-layer metal CMOS technology, has an active area of 9 mm^2, and has a peak clock frequency of 110.8 MHz at 1.8V (nominal). Power consumption is 306 mW when turbo decoding a 2-Mb/s data stream with ten iterations per block and eight voice calls simultaneously.

In [35], a fully synthesizable VHDL turbo decoder model is developed and logic synthesis is performed for a five-metal $0.25\mu m$ CMOS standard cell library with a supply voltage of 2.5V; allocated memories are dual-port synchronous static RAM. In addition, memory partitioning techniques for the interleaver memory are described.

In [36], a Log-MAP based turbo decoder for 3GPP W-CDMA mobile systems is designed and implemented in FPGA. The memory requirements are investigated and optimized from two aspects: (1) an alternative called state metric difference-storing method is proposed to save forward and backward state metric memory by up to one-third, (2) the interleaver address memory is reduced by repeating the address calculation at each iteration. The FPGA prototype operates at 42 MHz, has a decoding rate of up to 2.1 Mb/s with five iterations. The decoder is also synthesized for a $0.25\mu m$ CMOS standard technology; it has a maximum clock frequency of 92 MHz and the core logic is about 42k gates.

In [37], different memory optimizations for the MAP class of decoding algorithms are investigated. It turns out that it is not possible to present one decoder structure as being optimal. In fact, there are several tradeoffs which depend on the specific turbo code, the implementation target (hardware or software), and the selected cost function. The authors

end up with a parametric family of optimized algorithms from which the designer can choose.

In [2], several VLSI turbo decoder architectures are highlighted and compared in terms of complexity and performance. The impact on VLSI complexity of system parameters, such as state number, number of iterations, and code rate, is evaluated for different solutions. The results of this architectural study are then exploited for the design of a specific Log-MAP decoder, implementing a serial concatenation scheme with 2/3 and 3/4 codes; the designed circuit occupies 35 mm^2 and supports a 2 Mb/s data rate.

In [38], a low-power architecture of SOVA-based turbo decoder is proposed by adopting the Scarce State Transition (SST) scheme. Register Exchange Survival Memory Unit (RE-SMU) and Systolic block are used in the implementation of the SOVA decoder for high throughput and low latency.

In [39], a low-complexity multi-stage-pipeline turbo-code encoder and decoder architecture for wireless mobile communication applications is proposed. The turbo decoder is based on the Log-MAP algorithm and the complex channel statistic estimation process is simplified with minor performance degradation. Furthermore, a simple decision logic (based on the difference in hard decisions at previous and current iterations) is used for early iteration termination; then the turbo decoder goes into power-down mode.

In [40], finite precision effects on the performance of turbo decoders are analyzed and the optimal word length of variables is determined considering tradeoffs between performance and hardware cost. The paper shows that the performance degradation from the infinite precision is negligible if four bits are used for received bits and six bits for extrinsic information. A state metric normalization method (based on subtraction of a selected constant value) is also discussed. Furthermore, a power-down technique based on a group of decoding criteria is used for early termination of iterations. These criteria are (1) comparing extrinsic information against a preset value, (2) comparing LLR outputs against a preset value, (3) observing difference in the number of ones (in hard decisions) at previous and current iterations, and (4) observing sign similarities between extrinsic and LLR values. According to [41], this decoder is designed and fabricated using a 0.25μm CMOS standard-cell library and has a core area of 2.32×1.72 mm^2. The deocder is suitable for 3G W-CDMA systems with a 2 Mb/s data rate.

In [42], a power-efficient application specific VLIW (Very Long Instruction Set) processor for turbo decoding is designed. In this architecture, there are three levels of design: (1) a VLIW processor is generated from a C-language description with the A|RT[1] Designer tool, (2) a SISO decoder is embedded in the VLIW processor as an ASU (Application Specific Unit) generated by a high-level synthesis tool, (3) the data-path Processing Units (PU) in the ASU, used to perform forward, backward, and soft-output calculations, are specified in VHDL and synthesized by a logic-synthesis tool. The VLIW-based turbo decoder is implemented on an FPGA and runs at 2 Mb/s data rate.

Current communication systems can incorporate a high level of integration of multiple modules on a single chip, *i.e.*, System On Chip (SOC). Hence, a turbo codec (*i.e.*, turbo encoder and decoder) can be implemented as software or hardware Intellectual Property (IP) cores that can be embedded as building blocks, alongside other components, such as filters and demodulators, on a single-chip communication system. These IP cores have a high degree of design flexibility to suit various applications and customer needs. Examples of turbo encoder/decoder cores can be found in [43, 44, 45].

3.4 Dynamic-Iterative Techniques for Turbo Decoders

Many criteria have been suggested in the research to stop further decoding iterations when satisfying certain conditions, hence reducing power dissipation significantly. The stopping criteria can be categorized into two groups:

- Stopping techniques based on the improvement of channel quality. Some of these techniques are discussed, as follows:

[1] A|RT stands for Algorithm-to-Register-Transfer; the A|RT tools have been acquired by ARM®

- Cross-Entropy (CE) criterion, suggested in [18], as follows:

$$
\begin{aligned}
&T(i) \approx \sum_{k} \frac{\left|\Delta Le_2^{(i)}(\hat{u}_k)\right|^2}{e^{|\Lambda_1^{(i)}(\hat{u}_k)|}}, \quad \text{where} \\
&\Delta Le_2^{(i)}(\hat{u}_k) = \Lambda_2^{(i)}(\hat{u}_k) - \Lambda_1^{(i)}(\hat{u}_k) = Le_2^{(i)}(\hat{u}_k) - Le_1^{(i)}(\hat{u}_k), \\
&\Lambda_k^{(i)}, Le_k^{(i)} \quad \text{are the LLR and extrinsic info respectively} \\
&\text{of SISO decoder} \quad k \in \{1,2\} \quad \text{at iteration} \quad i.
\end{aligned} \tag{3.31}
$$

  In [18], it is shown that further iterations can be stopped if $T(i) \leq (10^{-2} \sim 10^{-4}) \cdot T(1)$. This method is complex to implement because of the non-linear operations involved and the use and storage of real numbers.

- Sign-Change-Ratio (SCR) criterion, suggested in [46], as a simplification of the CE method in [18] according to the relation:

$$
\begin{aligned}
&T(i) \approx \delta_i \cdot C(i), \quad \text{where} \quad \delta_i \quad \text{is a constant,} \\
&C(i) \quad \text{is the number of sign changes between} \\
&Le_2^{(i-1)}(\hat{u}) \quad \text{and} \quad Le_2^{(i)}(\hat{u})
\end{aligned} \tag{3.32}
$$

  Further iterations can be stopped when $C(i) \leq (0.01 \sim 0.06) \cdot C(1)$. This method is much simpler than the CE method but involves real number operations and memory storage of integers.

- Hard-Decision-Aided (HDA) criterion, also suggested in [46], this method compares the hard decisions (*i.e.*, decoded bits) of current and previous iterations. This method involves only binary operations and the storage of N bits (*i.e.*, hard decisions of previous iteration), where N is the frame length. This method is also used in [5, 39], as it was discussed in Section 3.3. From the same discussion, it can be seen that the stopping conditions suggested in [40, 41] are directly/indirectly extracted from the SCR and HDA criteria (*i.e.*, testing extrinsic information, LLR, and/or hard decisions).

- Stopping techniques based on the calculation of CRC checksum values are used in [47, 48, 49]. Since CRC checksums are part of the 3G wireless (Wide-band CDMA) standard, no overhead is required at the encoder part. The CRC checksum circuit adds a small overhead complexity to the turbo decoder, especially for long-frame sizes. At each decoding iteration up to the maximum number of iterations, if the CRC checksum of the decoded frame passes, further iterations are stopped; otherwise next iteration is executed.

A cancellation method is suggested in [48] to complement the CRC stopping method. In this method, further iterations are cancelled (*i.e.*, decoding is stopped) if the decoding results do not converge to the correct result. This method works as follows: Up to the maximum number of iterations, if the CRC checksum fails, the mean value of the LLRs of current iteration (μ) is calculated and the difference between this mean and that of the previous iteration ($\Delta\mu = \mu - \mu_{\text{old}}$) is calculated. If $\mu < \mu_{\text{th}}$ or $\Delta\mu < \Delta\mu_{\text{th}}$, cancel further iterations; otherwise execute next iteration.

Alongside these iteration stopping/cancellation techniques, there are two methods to save power at the idle time of the turbo decoder circuit or processor. These methods are power-down [26] and voltage scaling [26]:

- The power-down method is straightforward, and it means that the decoder circuit or processor (in case of software implementations) goes into a power-down mode when no further iterations are required. In this mode, voltage and frequency (thus time assigned for each iteration) are fixed. The turbo decoder implementations in [39, 40] apply the power-down mode alongside a stopping technique to save power consumption.

- The voltage scaling method uses different voltage supplies, and in some cases different frequencies, based on the expected number of decoding iterations. In [48], a voltage scaling (VS) heuristic is developed for programmable architectures. This technique changes the voltage and frequency of each iteration according to the remaining number of iterations. In [49], the VS heuristic is based on channel estimation which adds complexity overhead, hence more power consumption, to the turbo decoder.

According to [50], channel estimation can be eliminated with a small degradation to decoding performance; hence, the VS heuristic cannot be used for such a case.

3.5 Sources of Power and Energy Consumption

There are three major sources of power dissipation in digital CMOS circuits which are summarized by the following equation [26]:

$$P_{\text{tot}} = p_t \cdot (C_L \cdot V \cdot V_{\text{DD}} \cdot f_{\text{CLK}}) + I_{\text{sc}} \cdot V_{\text{DD}} + I_{\text{leakage}} \cdot V_{\text{DD}} \tag{3.33}$$

The first term represents the switching component of power, where C_L is the loading capacitance, f_{CLK} is the clock frequency, and p_t is the probability that a power consumption transition occurs (the activity factor). However, for conventional CMOS circuits, the voltage swing, V, is the same as the supply voltage, V_{DD}. The second term is due to the direct-path short circuit current, I_{sc}, which arises when both the NMOS and PMOS transistors are simultaneously active, conducting current directly from supply to ground [51, 52]. Finally, leakage current, I_{leakage}, which can arise from substrate injection and sub-threshold effects, is primarily determined by fabrication technology considerations [53]. The dominant term in a "well-designed" circuit is the switching component, and low power design thus becomes the task of minimizing p_t, C_L, V_{DD}, and f_{CLK}, while retaining the required functionality.

Power-delay product can be interpreted as the amount of energy expended in each switching event (or transition) and is thus particularly useful in comparing the power dissipation of different designs. If it is assumed that only the switching component of the power dissipation is important, then it is approximated by [26]

$$\text{Energy/Transition} \approx P_{\text{tot}}/f_{\text{CLK}} = C_{\text{effective}} \cdot V_{\text{DD}}^2 \tag{3.34}$$

where $C_{\text{effective}}$ is the effective capacitance being switched to perform a computation and is given by $C_{\text{effective}} = p_t \cdot C_L$.

The quadratic dependence of energy on voltage makes it clear that operating at the lowest possible voltage is most desirable for minimizing the energy consumption; unfortunately, reducing the supply voltage comes at the cost of a reduction in computational

throughput. Also in many cases, the designer has no control over the supply voltage (*e.g.*, when using a standard-cell CMOS library).

Energy can be reduced at the algorithmic, architectural, gate, and/or circuit levels. Most of the energy reduction comes at both the algorithmic and the architectural levels. Using a low complexity algorithm can reduce energy consumption dramatically. Parallelism and pipelining techniques, memory organization, and component-level design are examples of architectural level design. For the turbo decoder, quantization and normalization of soft data and metric parameters have important effects on energy computation. A great amount of energy consumption can be reduced if further decoding iterations are stopped upon satisfying some criteria that indicate decoding performance enhancement; these techniques are called dynamic-iterative techniques. The decoder goes into power-down mode for the unused iterations. Another choice used with dynamic iterative techniques is voltage scaling (VS), but the VS circuits (*e.g.*, DC-DC converters) have large area penalties, and are usually not available for standard cell CMOS designs.

Energy can be reduced at the gate level using techniques such as clock gating and toggle filtering to directly reduce switching activity, and hence total energy consumption. Low-power/low-energy algorithms and architectures reduce the switching activity indirectly (*e.g.*, by reducing the number of operations and/or memory transactions). Circuit-level techniques are not practical for complex designs such as turbo decoders, where the standard-cell design approach is usually used.

Chapter 4

A New Dynamic-Iterative Technique for Turbo Decoders

In this chapter, a novel low-complexity dynamic-iterative technique is proposed to reduce the energy consumption of the Log-MAP–based turbo decoder. The performance and complexity of the new technique are compared to those of other state-of-the-art dynamic-iterative techniques.

4.1 Quantization of the Log-MAP Turbo Decoder

As discussed in Section 3.1, the high-performance low-complexity Log-MAP decoding algorithm is chosen for implementation of the constituent SISO decoders of the turbo decoder.

The Log-MAP decoding algorithm cannot be implemented in VLSI without proper quantization and normalization:

- For the quantization part, it is found in [40, 41] that the following word-lengths give an optimum tradeoff between hardware cost, hence area and energy consumption, and decoding performance: 4:2 bits for received inputs, 6:2 bits for extrinsic information, 7:2 bits for branch metrics, and 9:2 bits for state metrics, where the q:f format refers to q total quantization bits of which f bits are used as a fraction. In addition, 10:2 bits are chosen for LLR soft outputs.

- For normalization, a simple method is utilized: At each time instant, all input state metrics are normalized by subtracting the minimum state metric calculated at the previous time instant, thus avoiding arithmetic overflow.
- A lookup table is used for the correction term of the Log-MAP algorithm, where 4:2 bits are used for the lookup table input (4-bit addresses) and 0:3 bits for lookup table outputs. Therefore, 16×3 bits are required for the lookup table implementation.

4.1.1 Decoding Performance of the Fixed-Point Approximation

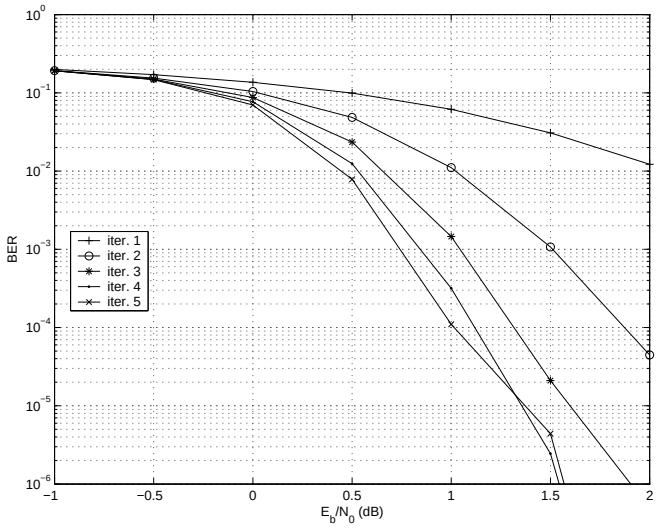
Figures 4.1 and 4.2 show comparisons of decoding performance of the original Log-MAP turbo decoder and the Quantized and Normalized Log-MAP turbo decoder (QN-Log-MAP), according to the previously mentioned parameters. The performance is measured in terms of BER or FER vs. E_b/N_0 (dB). The FER, rather than BER, is the performance figure-of-merit for high-speed data services of 3G wireless systems (W-CDMA and cdma2000).

For simplicity, these simulations and all subsequent simulations are based on a rate 1/3 turbo code with generator polynomial $\mathbf{g} = (13, 15)_8$, which is suitable for 3G wireless systems. The performance in terms of BER vs. E_b/N_0 is shown in Figure 4.1 for the Log-MAP and QN-Log-MAP turbo decoders. The performance in terms of FER vs. E_b/N_0 is shown in Figure 4.2 for the Log-MAP and QN-Log-MAP turbo decoders.

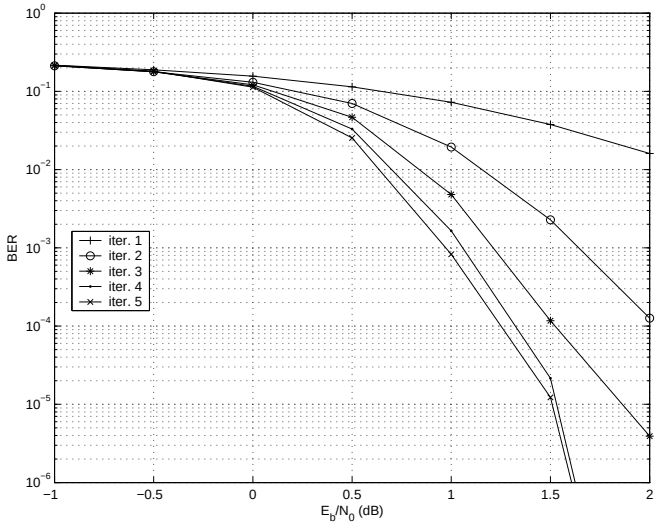
From Figures 4.1 and 4.2, it is evident that the fixed-point version of the Log-MAP (QN-Log-MAP) is close in performance to the original floating-point Log-MAP. At high SNRs, the QN-Log-MAP turbo decoder becomes more comparable in performance to the Log-MAP turbo decoder when increasing the number of decoding iterations (especially > 3).

4.2 Energy Reduction with Dynamic-Iterative Techniques

If the number of iterations is fixed regardless of channel conditions, then a large amount of energy is wasted by executing several redundant iterations. At good channel conditions

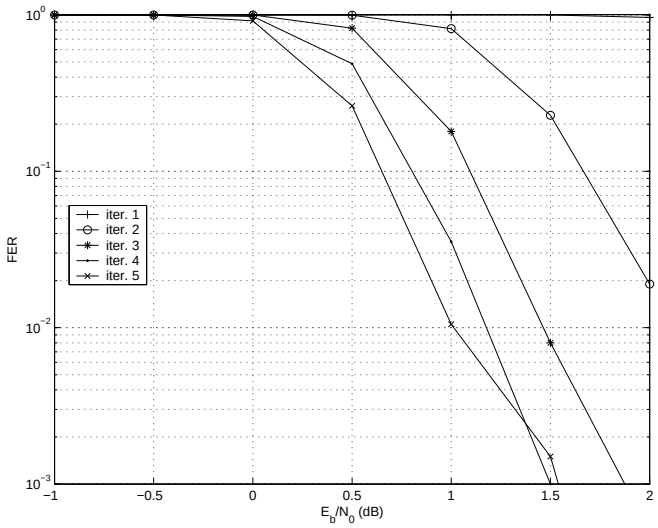


(a)

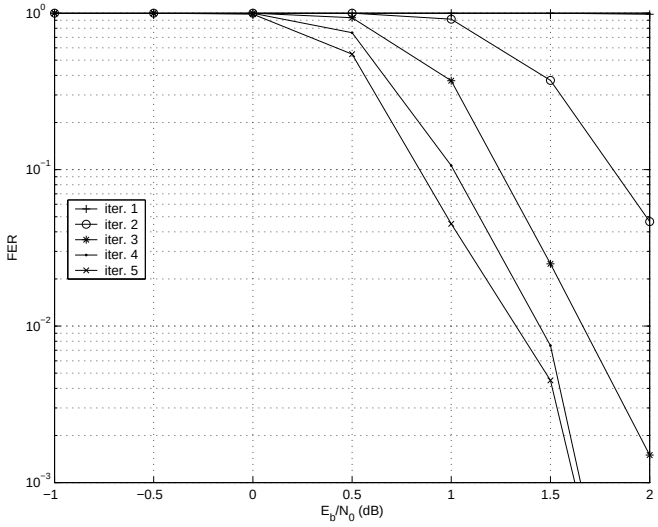


(b)

Figure 4.1: BER vs. E_b/N_0 for (a) Log-MAP (b) QN-Log-MAP turbo decoders (rate 1/3, $\mathbf{g} = (13, 15)_8$, 1024 bits/frame, 2000 frames, 5 iterations)



(a)



(b)

Figure 4.2: FER vs. E_b/N_0 for (a) Log-MAP (b) QN-Log-MAP turbo decoders (rate 1/3, $\mathbf{g} = (13, 15)_8$, 1024 bits/frame, 2000 frames, 5 iterations)

(*i.e.*, high SNRs), the correct frame can be obtained after a few iterations, and further iterations that increase energy consumption are redundant. On the other hand, if the channel conditions are poor (*i.e.*, at low SNRs), a correct frame cannot be reached after several iterations; hence, a large amount of energy is consumed without decoding the correct frame. The total energy consumed by the turbo decoder (over a specific period of decoding time or for a specific number of received frames), E_{tot}, can be defined by the following equation:

$$E_{\text{tot}} = E_{\text{it}} \cdot n_{\text{it}} \tag{4.1}$$

where E_{it} is the energy consumed per iteration, and n_{it} is the total number of iterations used by the turbo decoder. For fixed-iterative techniques, n_{it} is constant. For dynamic iterative techniques n_{it} is variable, hence

$$E_{\text{tot}} \propto n_{\text{it}} \tag{4.2}$$

From this relation, total energy consumption of the turbo decoder can be reduced by applying techniques that stop or cancel unnecessary iterations from being executed, and then forcing the turbo decoder into power-down mode for these unused iterations. Several techniques to stop redundant iterations have been discussed in the previous chapter. Among these methods are CE [18], SCR [46], HDA [46, 5, 39], CRC [47, 48, 49], and other statistical methods that depend on extrinsic and/or LLR soft outputs [5, 39]. Either the power-down [5, 39] mode or voltage-scaling [48, 49] mode is used to save power consumption during the unused time slots. Because voltage-scaling techniques require either specific programmable architectures or specific circuitry that cannot be mixed with the current single-supply standard-cell CMOS libraries, such techniques are not investigated in this thesis. Therefore, the power-down mode is used to save power consumption during the idle iteration periods; hence reducing energy consumption of the turbo decoder. Most of the current dynamic-iterative techniques use a complex circuitry and/or additional memory. Also, all these methods, except the method in [48], consider enhancing energy consumption at good channel conditions only; hence, a considerable amount of energy is still consumed if the channel conditions are poor. The cancellation method suggested in [48] for poor channel conditions (discussed in Section 3.4) involves calculation and storage of averages extracted from LLR soft outputs; hence, this method is not energy efficient. From this discussion

and the more detailed one in Section 3.4, it is found that the CRC and HDA techniques, along with power-down mode, are the lowest-complexity dynamic-iterative techniques.

4.2.1 CRC Stopping Method

CRC coding is part of the 3G wireless standard (W-CDMA and cdma2000) [11, 12]. CRC parity bits are added before turbo coding, leading to the frame structure in Figure 4.3.

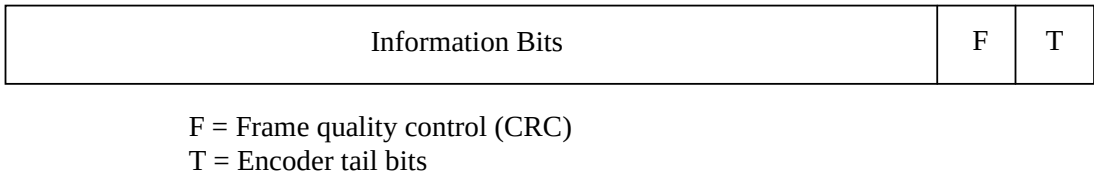


Figure 4.3: Frame structure used in 3G wireless CDMA standards

At the receiver, the CRC can be combined with turbo decoding to detect correct frames early and stop redundant iterations. The W-CDMA and cdma2000 3G standards specify CRC parity bit sizes to be 8, 12, 16, or 24 (W-CDMA only). For example, the 8-bit CRC generator polynomial used in both cdma2000 and W-CDMA is

$$\mathbf{g}(D) = D^8 + D^7 + D^4 + D^3 + D + 1 \tag{4.3}$$

For larger CRC bit sizes, cdma2000 and W-CDMA employ different generator polynomials. Figure 4.4 illustrates the 8-bit CRC encoder structure of (4.3).

The CRC decoder has a similar structure to that of the CRC encoder, but the switches are up for all the received $(N + F)$ bits. The CRC checksum is calculated by adding the last F bits stored in the shift register. If the CRC checksum is zero, a correct frame is reported. Therefore, after each turbo-decoding iteration, a CRC checksum is calculated for the resulting frame (*i.e.*, hard decisions). If the checksum is zero, further iterations are stopped and the turbo decoder goes into power-down mode. The circuit complexity of the CRC-checksum circuit is simple. It includes an F-bit shift register, a maximum of F XOR gates, and an F-bit modulo-2 addition circuit (*e.g.*, F-bit OR or NOR gate can be used to detect the all-zero CRC syndrome).

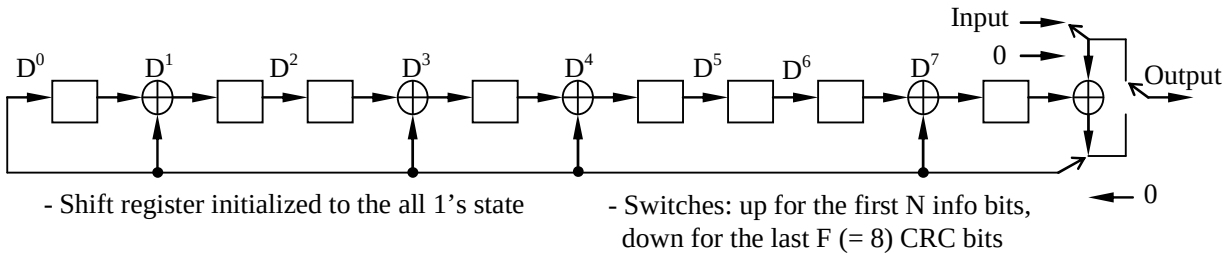


Figure 4.4: The 8-bit CRC-encoder for cdma2000 and W-CDMA standards [11, 12]

4.2.2 HDA Stopping Method

The HDA method was described in Section 3.4. At each decoding iteration, each decoded bit is compared to the corresponding bit resulting from previous iteration. The number of bit differences between current and previous iterations is counted. If it is less than or equal to a predefined threshold (small integer ≥ 0), further decoding iterations are stopped and the decoder goes into power-down mode. This technique is based on the observation that the number of differences between decoded bits from consecutive iterations decreases when the output converges to the correct frame. At some small threshold value of Hard Decision Difference (HDD), the output frame can be considered to be correct. It is found that 1% of the frame length is a good approximation for the HDD threshold (HDD_{th}). The complexity of the HDA circuit is simple. It includes bit-comparisons (XOR gates), bit-additions (using counter/incrementor), an integer-comparison, and an integer storage of the threshold-value. The obvious disadvantage of this method compared to the CRC method is that it usually involves an additional iteration, and thus a larger energy consumption. Also, the accuracy of this method depends on the threshold value set, which can be observed with simulation.

4.3 A New Dynamic-Iterative Technique: CRC-HDD

The low-complexity CRC and HDA methods, discussed in Section 4.2, can be used as stopping techniques at good channel conditions (*i.e.*, high SNRs) only. These methods cannot cancel unnecessary decoding iterations at poor channel conditions (*i.e.*, low SNRs), hence a large amount of energy is still consumed at such cases without decoding the correct

data frame. So far, there is only one cancellation technique that is suggested in [48]. As discussed in Section 4.2, this method has the complexity of calculating the average of all LLR soft outputs which increases power consumption per decoding iteration, and hence total energy consumption of the turbo decoder. In addition, the accuracy of this method depends on proper selection of threshold values required by the algorithm, and it is reported in [48] that this method implies a degradation in FER.

A novel dynamic-iterative technique is proposed in this work. The new technique uses the CRC checksum method to stop redundant iterations at good channel conditions and a novel method based on hard decision differences (HDD) to cancel unnecessary iterations at poor channel conditions. To save energy consumption of the turbo decoder, it goes into power-down mode for the idle iteration periods.

4.3.1 Iteration Stopping Using the CRC Method

The CRC method is used to stop redundant decoding iterations upon detection of correct data frame (when the CRC checksum is zero). This method is chosen for the stopping part due to its 3G compatibility and its lower number of iterations compared to that of the HDA method. The CRC method works well at good channel conditions, but, at poor channel conditions, the correct data frame cannot be detected and the maximum number of iterations is used. As a result, a large amount of the decoder energy is still wasted. In addition, if most of those unnecessary iterations can be cancelled, an Automatic Repeat Request (ARQ) can be sent earlier to the transmitter, hence reducing delays in processing received data. Therefore, a new low-complexity cancellation method is proposed to complement the CRC method (for poor channel conditions).

4.3.2 Introducing a Novel Cancellation Method: HDD

A novel low-complexity method is proposed to cancel further decoding iterations that do not converge to the correct data frame. Before explaining the new cancellation method, we will define some terms:

HD (Hard Decisions): The set of decoded bits (*i.e.*, frame) at each iteration.

HDD (Hard Decisions Difference): The number of bit differences between HD data from previous and current iterations, *i.e.*, $\sum_{i=0}^{N-1}(\mathrm{HD}_{\mathrm{prev},i} \oplus \mathrm{HD}_{\mathrm{curr},i})$

DHDD (Decrease in HDD): $\mathrm{HDD}_{\mathrm{prev}} - \mathrm{HDD}_{\mathrm{curr}}$

At each decoding iteration, if the CRC circuit cannot detect a correct data frame (*i.e.*, a non-zero checksum), a novel low-complexity cancellation method is used to detect divergence of the output from the correct data frame. At some threshold value, this method stops further decoding iterations. This cancellation method, like the HDA stopping method, depends on bit-differences between hard decisions (*i.e.*, decoded frames) resulting from consecutive decoding iterations.

This new cancellation method will be called HDD-aided (or HDD in short), derived from the fact that it depends on changes in HDD values (rather than changes in HD values, as in the HDA method). Consequently, the new combined stopping/cancellation technique will be called CRC-HDD (CRC for stopping and HDD for cancellation). Although the terms *stopping* and *cancellation* have the same meaning of halting further decoding iterations, stopping is used to indicate that further iterations are stopped based on a successful detection of the correct data frame (which usually happens at good channel conditions), and cancellation is used to indicate that further iterations that do not converge to the correct data frame are cancelled (which usually happens at poor channel conditions).

The new HDD cancellation method (as part of the suggested CRC-HDD dynamic-iterative technique) is based on observation of the following: firstly, the relation between averages of HDD values and SNR (shown in Figure 4.5), and secondly, the relation between averages of DHDD values and SNR (shown in Figure 4.6). For simplification, HDD values for the cancelled iterations are approximated to zero; this approximation has no significant effect on the final results.

From Figure 4.5, it can be seen that average HDD is inversely proportional to SNR. From this observation, we can possibly extract some HDD threshold and cancel iterations above this threshold, hence below some SNR. However, this method proves to be inaccurate because the variances of observed HDD values around their averages are found to be high. This means either degradation of FER at high SNRs or inefficient cancellation at low SNRs.

Instead of taking averages of HDD values, DHDD values from consecutive iterations are considered. Figure 4.6 shows that the relation between average DHDD and SNR is

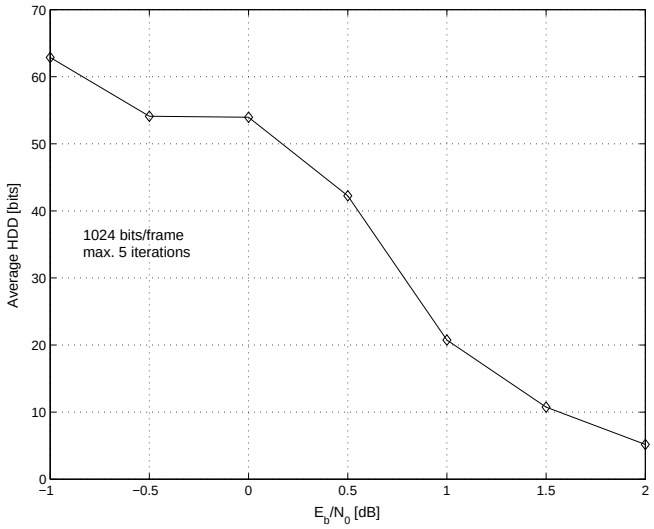


Figure 4.5: Average HDD vs. E_b/N_0 (1024 bits/frame, max. 5 iterations)

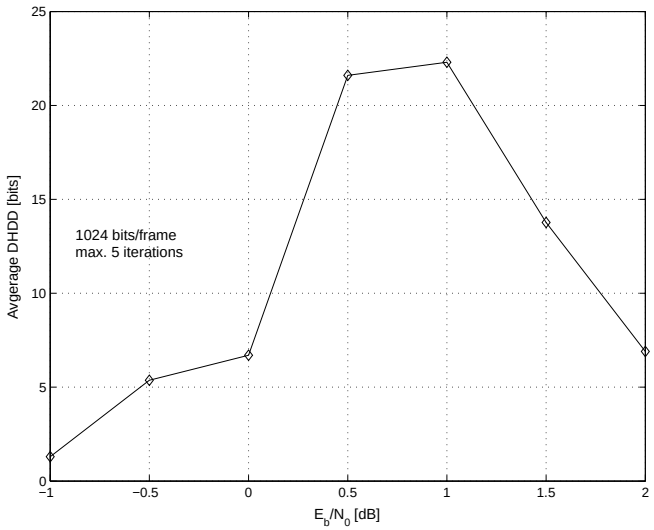


Figure 4.6: Average DHDD vs. E_b/N_0 (1024 bits/frame, max. 5 iterations)

proportional at low SNRs and inversely proportional at high SNRs. Also, it is found that the variances of observed DHDD values around their averages are small. From these observations, some DHDD threshold can be extracted to cancel iterations below that threshold. To avoid a sever performance degradation due to the decreasing phenomena of DHDD in the high SNR range, the CRC stopping method must be applied before the new cancellation method in the execution order of the dynamic-iterative algorithm. In addition, a new DHDD value cannot be available before the third iteration, thus further minimizing the chance of an incorrect cancellation (decoding performance of the new dyanmic-iterative technique is discussed in Section 4.3.4).

The newly developed method is efficient in cancelling iterations at low SNRs only. By combining this new method with the CRC method, the HDD method cancels a large fraction of the unnecessary iterations at low SNRs (below DHDD threshold), and the CRC method stops redundant iterations at high SNRs (at zero CRC checksums). In the new combined CRC-HDD technique, at each decoding iteration, the CRC zero-checksum is tested first, and if it fails the calculated DHDD value is compared to the DHDD threshold. Setting the DHDD threshold value at 1% of the frame length leads to a high cancellation efficiency. Note that a cancellation cannot occur before the third iteration, where the first DHDD value can be calculated.

4.3.3 The New CRC-HDD Dynamic-Iterative Algorithm

Figure 4.7 illustrates the CRC-HDD dynamic-iterative algorithm applied to turbo decoders. As shown in the flowchart, after calculation of LLR values, hard decision (HD) output bits are estimated. If the iteration number is less than the maximum, HDD, DHDD, and CRC checksum are calculated. If CRC checksum is equal to zero, decoding is stopped. Otherwise, the algorithm checks if DHDD is less than or equal to $\mathrm{DHDD}_{\mathrm{th}}$, and stops decoding if the condition is satisfied. It is noteworthy that the DHDD check cannot happen before the third iteration. If both checks fail, the algorithm stores the current HD and HDD values in special buffers, for use in the following calculations, before proceeding to the next iteration.

The CRC-HDD method requires a low-complexity logic to be added to the turbo decoder. The HDD section requires a maximum of N (= frame length) bit comparisons to

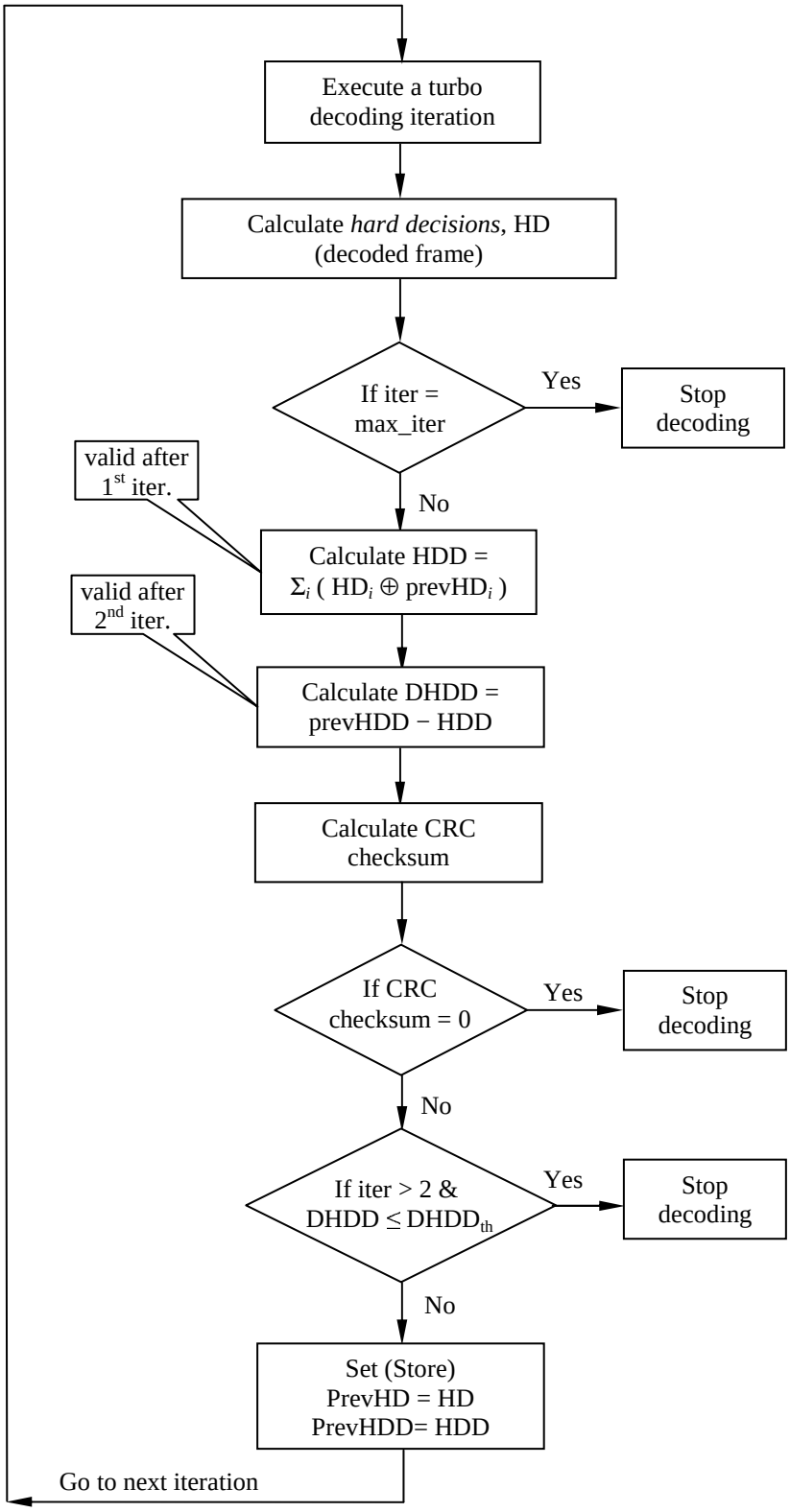


Figure 4.7: The CRC-HDD algorithm

calculate HDD, a signed subtraction to calculate DHDD, and a magnitude comparison between the calculated and threshold DHDD values. Also, memory storage is required for the following: previous HD and HDD values, and DHDD threshold. The number of operations and memory size used in the additional logic are marginal compared to those required for the SISO Log-MAP component decoders, interleavers, and deinterleavers.

4.3.4 Comparing Decoding Performance for CRC, HDA, and CRC-HDD

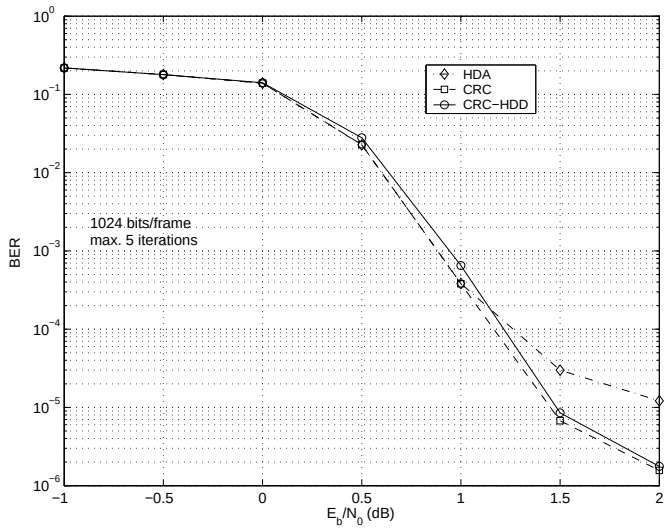
The CRC [47], HDA [46], and the proposed CRC-HDD dynamic-iterative methods are applied at the system-level to a rate 1/3 Log-MAP turbo decoder with generator polynomial $\mathbf{g} = (13, 15)_8$. Different frame lengths are used over the SNR range $\{-1, 2\}$ dB with a maximum number of five iterations and DHDD_{th} set to 1% of the frame length. The three techniques are applied to an encoder/decoder system modeled in the C language, with about 10M bits as encoder input and an AWGN channel model.

Decoding performance simulations are applied to turbo decoders using the three techniques to verify that the new CRC-HDD technique is close in performance to other low-complexity dynamic-iterative techniques (CRC and HDA). Figure 4.8(a) shows the BER vs. E_b/N_0 for the three techniques applied to a rate 1/3 Log-MAP turbo decoder with 1024 bits/frame and a maximum of five iterations; Figure 4.8(b) shows the FER vs. E_b/N_0 for the three methods applied to the same decoders.

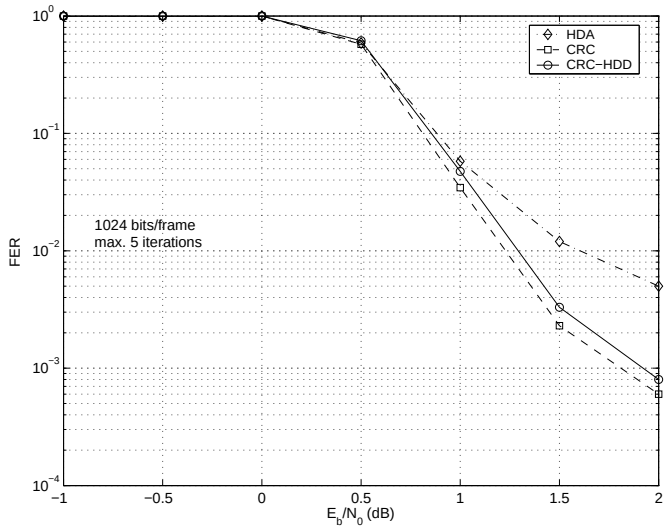
From Figure 4.8, it can be seen that the performance of the turbo decoder using the new technique is close to that of the decoder using the CRC method and better than that of the decoder using the HDA method.

4.3.5 Comparing Iteration Reduction for CRC, HDA, and CRC-HDD

The three methods are also compared in terms of percent reduction in iterations. Figure 4.9 shows the average number of iterations vs. E_b/N_0 for the CRC, HDA, and CRC-HDD Log-MAP turbo decoders for 1024-bit frames and a maximum of five iterations. As



(a)



(b)

Figure 4.8: (a) BER vs. E_b/N_0 (b) FER vs. E_b/N_0 for rate 1/3 Log-MAP turbo decoder with the CRC, HDA, and proposed CRC-HDD dynamic-iterative techniques

observed from the graph, the CRC-HDD technique is superior in iteration reduction to the CRC and HDA techniques.

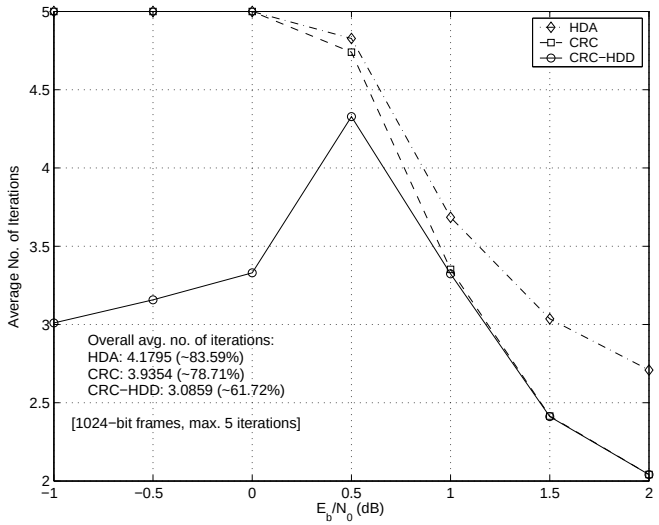


Figure 4.9: Average number of iterations vs. E_b/N_0 for rate 1/3 Log-MAP turbo decoders with the CRC, HDA, and proposed CRC-HDD dynamic-iterative techniques

Table 4.1 compares the percent reduction in iterations for turbo decoders using the CRC, HDA, and CRC-HDD dynamic-iterative techniques and employing different frame lengths. From Table 4.1, it is observed that the proposed CRC-HDD technique saves at least 35% of iterations compared to the static-iterative case (reduction increases for shorter frame lengths). It is also observed that the reduction given by the CRC-HDD method is more than 2.2 times the reduction given by the HDA method and more than 1.6 times the reduction given by the CRC method. If the effects of static power consumption and dynamic-iterative circuit complexity are ignored, a corresponding reduction in energy consumption is expected. For deep sub-micron technologies, the static power effects cannot be ignored, and more accurate figures for power/energy reduction has to be measured after hardware realization.

Table 4.1: Percent reduction in the number of iterations by the CRC, HDA, and proposed CRC-HDD dynamic-iterative techniques

Dynamic-iterative technique	Percent reduction in iterations				
	192 b/fr. 12b-CRC	384 b/fr. 16b-CRC	768 b/fr. 16b-CRC	1024 b/fr. 24b-CRC	5114 b/fr. 24b-CRC
HDA [46]	19.46%	17.81%	16.90%	16.41%	15.18%
CRC [47]	26.41%	23.96%	21.90%	21.29%	18.46%
Proposed CRC-HDD	43.18%	40.75%	39.02%	38.28%	35.69%

4.4 Hardware Complexity of the CRC-HDD Logic

Hardware complexity of the proposed CRC-HDD dynamic-iterative technique depends on frame length, N, and CRC-parity bit length, F. The complexity can be calculated in terms of memory bits and bit-level operations required.

For the CRC part, we need an F-bit register and a maximum of F 2-bit XOR gates for the CRC decoder (see Figure 4.4), and F-bit OR/NOR gate (or its equivalent) for the zero-checksum detection.

For the HDD part, we need N bits for the HD (decoded frame) buffer, 2-bit XOR gate(s) for N bit-wise comparisons of current and previous HD values, $\lceil \log_2 N \rceil$-bit counter (incrementor) and $\lceil \log_2 N \rceil$-bit buffer for current and previous HDD values respectively, $(\lceil \log_2 N \rceil + 1)$-bit signed subtractor to calculate DHDD, $\lceil \log_2(0.01N) \rceil$ bits to store DHDD_{th} value, and $(\lceil \log_2 N \rceil + 1)$-bit magnitude comparator to check if $\text{DHDD} \leq \text{DHDD}_{\text{th}}$.

We see that the CRC part has the same hardware complexity as that of the typical CRC technique, and the HDD part has a slightly larger complexity than that of the HDA technique because of the additional HDD buffering and subtraction. To reduce power consumption of the CRC-HDD circuit, a complexity reduction is suggested for the HDD part.

4.4.1 Complexity Reduction of the HDD Section

To reduce hardware complexity of the HDD circuit, hence complexity of the CRC-HDD algorithm, the following techniques are applied to the new design:

1. To avoid using an additional N-bit buffer to store the current (recently-calculated) HD value used for comparison with the previously-buffered HD value, an address latch can be used to store the deinterleaving address, hence allowing reading from then writing to the same HD memory location on two consecutive clock cycles. An additional $\lceil \log_2 N \rceil$-bit buffer is required for the deinterleaving address latch.

2. The number of memory bits and bit-operations of the HDD part can be reduced (thus reducing switching activity which contributes to power consumption) by extracting maximum values of HD and HDD through simulations. Using simulations for the 400 and 1024-bit frames, we find that the following values give the same accuracy as the full-length solution that was discussed earlier:

   Minimum $0.1N$ bit-wise comparisons of HD previous and current frames (*i.e.*, further comparisons are stopped if $0.1N$ bit-differences have been discovered), $\lceil \log_2(0.1N) \rceil$-bit counter (incrementor) and $\lceil \log_2(0.1N) \rceil$-bit buffer for current and previous HDD values respectively, $(\lceil \log_2(0.1N) \rceil + 1)$-bit signed subtractor to calculate DHDD, $\lceil \log_2(0.01N) \rceil$ bits to store DHDD_{th} value, and $\lceil \log_2(0.01N+1) \rceil$-bit magnitude comparator to check if $\text{DHDD} \leq \text{DHDD}_{\text{th}}$, where DHDD is range-limited to $\lceil \log_2(0.01N+1) \rceil$-bit unsigned values.

Table 4.2 summarizes the hardware complexity in terms of memory bits and bit-level operations for both typical (non-optimized) and low-complexity implementations of the HDD circuit used in the proposed CRC-HDD technique.

Table 4.2: Comparison between typical and low-complexity implementations of the HDD circuit for different frame lengths (= N bits)

		Typical HDD implementation			Low-complexity HDD implementation		
		General	$N =$ 400	$N =$ 1024	General	$N =$ 400	$N =$ 1024
Memory bits		$N + 2 \cdot \lceil \log_2 N \rceil + \lceil \log_2(0.01N) \rceil$	420	1048	$\lceil \log_2 N \rceil + 2 \cdot \lceil \log_2(0.1N) \rceil + \lceil \log_2(0.01N) \rceil$	23	28
Bit opers.	Comp.	$N + \lceil \log_2 N \rceil + 1$	410	1035	$(0.1 \rightarrow 1)N + \lceil \log_2(0.01N + 1) \rceil$	$43 \rightarrow 403$	$106 \rightarrow 1028$
	Subt.	$\lceil \log_2 N \rceil + 1$	10	11	$\lceil \log_2(0.1N) \rceil + 1$	7	8
	Total	$N + 2 \cdot \lceil \log_2 N \rceil + 2$	420	1046	$(0.1 \rightarrow 1)N + \lceil \log_2(0.1N) \rceil + 1 + \lceil \log_2(0.01N + 1) \rceil$	$50 \rightarrow 410$	$114 \rightarrow 1036$

Chapter 5

An Energy-Efficient Design of Turbo Decoder

5.1 Architectural-Level Techniques Applied to the Turbo Decoder

In this chapter, several architectural-level techniques applied to the turbo decoder design are explained. These VLSI techniques aim at reducing power/energy consumption, in addition to improving throughput and silicon area, of the turbo decoder.

5.1.1 Algorithm Selection and Quantization

As discussed in Chapter 3, the Log-MAP algorithm gives the best tradeoff in terms of decoding performance and implementation complexity. Therefore, it is chosen as the decoding algorithm for the constituent SISO decoders. Nonetheless, the floating-point Log-MAP algorithm is still complex and difficult to realize into hardware. Therefore, the fixed-point approximation (as discussed in Chapter 4) is used for sampled input data, arithmetic calculations, and intermediate results. For quantization values, the notation n:q is used, where n represents the total number of bits and q represents the number of bits in the fractional part. The quantized values are as follows: 4:2 for soft (sampled) inputs, 7:2 for branch

metrics, 9:2 for state metrics, and 6:2 for extrinsic information. These fixed-point values represent an optimal tradeoff between decoding performance and area/power of turbo decoder implementations [40].

5.1.2 Parallelism

Parallelism is done at two hierarchical levels:

- Parallel architectures are built for the low-level branch metric (BM), forward and backward state metric (SM), and log-likelihood-ratio (LLR) calculation units by assigning one unit to each state instead of sharing units among states in the trellis structure. The parallel execution by these units increases throughput by the number of states (eight for the 3G decoder), but this occurs at the expense of area and power consumption.

- To further increase throughput, an additional level of parallelism can be applied by simultaneously processing the two halves of the data block (frame) [20, 25]. For this purpose, double units of BM, SM (forward and backward), and LLR are introduced into the architecture. This further doubles the throughput, again at the expense of more area and power. Figure 5.1 illustrates the parallel processing of two half-frames (one begins with forward recursion while the other begins with backward recursion, then reversing rules at the next recursion).

By superimposing the two parallelism techniques, throughput is theoretically increased by 16 (when ignoring complexity of the SM normalization logic). The side effects of area and power consumption for the parallel architecture can be reduced by applying the following three architectural-level techniques.

5.1.3 A New Operator Reduction Method for the $\max^*$ Logic

There are three major operations in the Log-MAP turbo decoding algorithm: BM calculation, SM calculation, and LLR calculation. Each of these operations involves the $\max^*$ operation, defined by the following Jacobian approximation (see also (3.19)):

$$\max^*(a, b) = \max(a, b) + f_c(|a - b|)$$

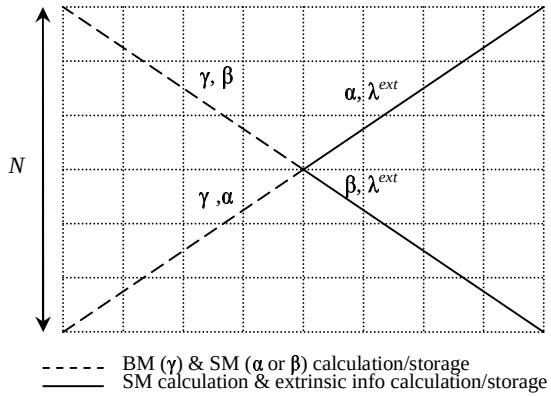


Figure 5.1: Parallel processing of two half-frames (one is in forward recursion while the other is in backward recursion)

where $f_c(x) = \ln(1 + e^{-x})$ can be represented by a small lookup table (with 16 entries); max* can be applied to more than two arguments by using recursion (similar to the max operation).

The calculation of absolute values in such operations increases the critical-path delay of the turbo decoder (also increases area and power consumption). This calculation of absolute values (abs logic) can be avoided by doubling the lookup table (LUT) size (to 32 entries) and adding an extra (sign) bit to the LUT input address (to account for both positive and negative values of $a - b$). Results show that the new design of the max* logic has shorter propagation delay, lower power consumption, and smaller silicon area. Figure 5.2 depicts both the classical (with abs logic) and the new (no abs logic) versions of the max* logic. In the diagram, the abs and LUT blocks (bounded by the dashed rectangular area) are replaced by a double-sized LUT (as explained above). In the suggested double-sized LUT, the LUT contents are repeated for positive and negative addresses that have the same magnitude.

A comparison of area, power, and delay for both classical and new implementations of the max* logic is summarized in Table 5.1. Three versions of the max* logic (with input lengths of 6, 10, and 11 bits) are required for BM, SM, and LLR calculation units, respectively. Results are produced using VHDL synthesis with Synopsys® Design Compiler.

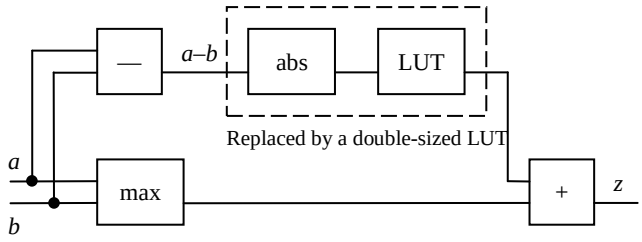


Figure 5.2: The max* operation (in the new design, the abs logic and the 16-entry LUT are replaced by a 32-entry LUT)

The minimum 0.47ns reduction in max* delay (as observed from Table 5.1) leads to about 1.4ns reduction in critical-path delay of the turbo decoder (max* appears three times in the LLR delay path); this increases the maximum clock frequency and hence increases the data rate, or throughput, of the turbo decoder. The small reductions in area and power are magnified throughout the turbo decoder due to the repetition of max* logic in the doubled parallel structures for BM, SM, and LLR calculation (max* duplication factor $= 2 \times (1\{\text{BM}\} + 8\{\text{SM}\} + 14\{\text{LLR}\}) = 46$).

5.1.4 Normalization of State Metrics

The calculation of current forward and backward state metrics, at each trellis stage, requires the addition of branch metrics to previous state metrics. Over time, this leads to an arithmetic overflow and produces erroneous results; hence, a normalization method is required. Two state-metric normalization methods are discussed and compared in terms of silicon area, power consumption, and propagation delay.

The first state-metric normalization method calculates the maximum SM value (using recursive calls of the max logic), stores that value in a buffer, and, on next trellis stage, subtracts it from all input SMs of the SM calculation logic. In [54], the SMs are normalized by subtracting the minimum value of the previous SMs. The serial call of max incurs a long delay in the critical-path of the turbo decoder. This delay can be reduced in the parallel implementation of state metric units by using a parallel max structure (for eight state metrics, only three stages of 2-input max units are needed instead of seven for the serial

Table 5.1: Results for classical and new implementations of the max* logic

Synthesis result	max* with abs logic	max* without abs logic
6-bit:		
Area (μm^2)	1594	1399
Power (mW)	3.85	3.33
Delay (ns)	3.45	2.83
10-bit:		
Area (μm^2)	2765	2289
Power (mW)	6.57	5.38
Delay (ns)	4.48	3.98
11-bit:		
Area (μm^2)	3009	2492
Power (mW)	7.17	5.93
Delay (ns)	4.82	4.35

max structure). Figure 5.3 illustrates the parallel structure for an 8-input max calculation logic.

Another method that proves more efficient in terms of area, propagation delay, and power consumption works by comparing all new SM values against a threshold value. If any SM exceeds the threshold, a constant value is subtracted from all SM values to prevent arithmetic overflow in next stages. The threshold value and the subtracted constant are chosen as 100% and 50%, respectively, of the highest value for SM quantization (*i.e.*, 2^{q-f-1} and 2^{q-f-2}, respectively, for a q:f quantization). The SM normalization method in [40] is similar, but both values are chosen as 50% of the highest value for SM quantization (*i.e.*, 2^{q-f-2}). Figure 5.4 shows the subtraction-based state metric normalization algorithm.

The two SM normalization methods based on the parallel-max architecture and the subtraction-based algorithm are VHDL-synthesized and compared in terms of area, power, and propagation delay, as shown in Table 5.2. From the table, we find that the subtraction-based SM normalization method has a much lower silicon area (about four times), power

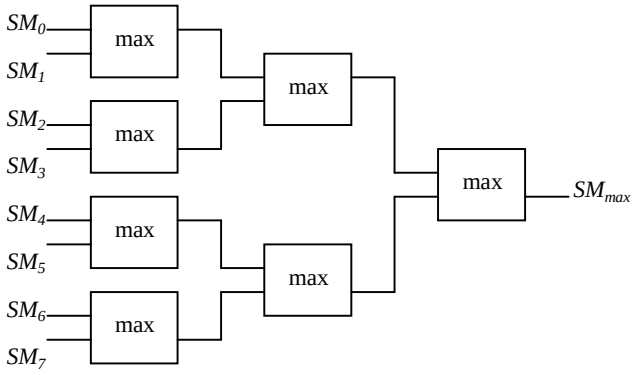


Figure 5.3: The parallel max normalization logic used in the first design of parallelized SM units (3-stage delay for eight inputs)

```
subt_flag := 0
FOR EACH SM(i)
   IF SM(i) >= SM_th THEN
      subt_flag := 1
   END IF
END FOR
IF subt_flag = 1 THEN
   FOR EACH SM(i)
      SM(i) := SM(i) - SM_subt
   END FOR
END IF
```

Figure 5.4: The subtraction-based state-metric normalization algorithm (used in the final design of parallelized SM units)

consumption (5.3 times), and propagation delay (2.8 times) than those of the other normalization method. Consequently, it is chosen for final implementation of the turbo decoder.

Table 5.2: Implementation results for the two SM normalization methods

Synthesis result	parallel max-based	subtraction-based
Area (μm^2)	11473	2858
Power (mW)	31.72	6.03
Delay (ns)	5.12	1.78

5.1.5 Resource Sharing

Another way to reduce area and power consumption of the turbo decoder is to apply resource sharing. Similar to the parallelism discussed in Section 5.1.2, resource sharing can also be applied at two levels of the design hierarchy:

- Both forward and backward SM units can be shared over the two subsequent recursions in each Log-MAP SISO decoder (in the parallel case of processing two half-frames, only two SM units, one forward and one backward, are required instead of four). Since those SM units are parallel structures of the basic one-state SM units, a large reduction in area and power consumption will result.

- On a higher level, instead of using two Log-MAP SISO decoders (one processing a half-iteration while the other one is idle), one Log-MAP SISO decoder is shared. The double Log-MAP scheme wastes a large silicon area of the decoder (almost double that of the shared scheme); the scheme also dissipates more power, even when applying low-power techniques (since leakage power consumption is always generated). Therefore, sharing one Log-MAP SISO decoder over two half-iterations is more area and power efficient.

This two-level resource sharing technique has no effect on throughput while reducing area and power consumption of the turbo decoder.

5.1.6 Interleaver Design

There are two popular categories of interleavers/deinterleavers: ROM-based and RAM-based. The former is usually based on patterns pre-generated by a random address generation software and programmed on a ROM. The latter calculates the addresses once (for every change of frame length) and stores the values in a RAM that is used throughout the decoding process (as long as received frames have the same length). A 3G-compliant interleaver should work with frame lengths between 40 and 5114 bits [12]. To allow for design flexibility, a RAM-based interleaver is implemented. Since the 3G interleaver uses a mother interleaver of at least 512 bits [4], a minimum initial latency of 512 clock cycles is introduced (even in the case of short frame lengths, *e.g.*, 128 bits). Initial latency is determined by the larger of mother interleaver length and the block length multiplied by the reciprocal of code rate; *e.g.*, for a rate 1/3 turbo decoder and frame length of 1024-bit, the initial decoding latency is 3072 (1024×3) cycles.

5.1.7 Double Buffering

In the final design of the turbo decoder, double buffering of soft-input data is employed to increase throughput of the turbo decoder. In other words, soft-input data memories are doubled to eliminate decoding delays between consecutively-received frames, thus increasing data throughput (or bit rate) of the turbo decoder. Double buffering works as follows: while one group of previously-stored soft-input data buffers is accessed by the turbo decoder for processing of the current data frame, a new set of soft-input data is stored in the other buffer group to allow for processing of the next frame immediately after the current frame is decoded. Hence, time delays are eliminated in the decoding of sequentially-received data frames.

5.2 Turbo Decoder Design Hierarchy

The turbo decoder is coded in VHDL, and then synthesized into a 0.18μm CMOS standard-cell based, or gate-level, design using Synopsys® Design Compiler. Figure 5.5 shows the design hierarchy for the VHDL-based turbo decoder. The Register-Transfer-Level (RTL) model of the turbo decoder is structural and VHDL-based. Each VHDL design unit is described by an ENTITY (describing interface) and an ARCHITECTURE (describing behaviour, dataflow, and/or structure). The VHDL design units, except Virage® behavioural RAM models, are Synopsys-synthesizable. The following is a brief description of each design unit in the turbo decoder structural RTL model.

td_chip

This is the top-level unit of the turbo decoder design. It contains the turbo decoder core logic (*td_core*) and the Virage-compiled RAMs. The I/O pads are added to the design after synthesis. Figure 5.6 shows the block diagram for the new turbo decoder (*td_chip*).

RAMs

The Virage® memory compiler generates both black-box RAM modules for synthesis and behavioural VHDL models for simulation.

td_core

The turbo decoder core (*td_core*) consists of the main turbo decoder logic (*turbo_dec*) and supporting logic for buffering and serialization of input/output data to/from the decoder (*io_logic*).

io_logic

The decoder support logic (*io_logic*) has three blocks: *recs_calc* for scaling input data using the channel reliability factor, L_c (defined in (3.23)), *buff_indata* for demultiplexing and double-buffering of soft-input data, and *ser_outdata* for serializing hard decision data from the HD buffer.

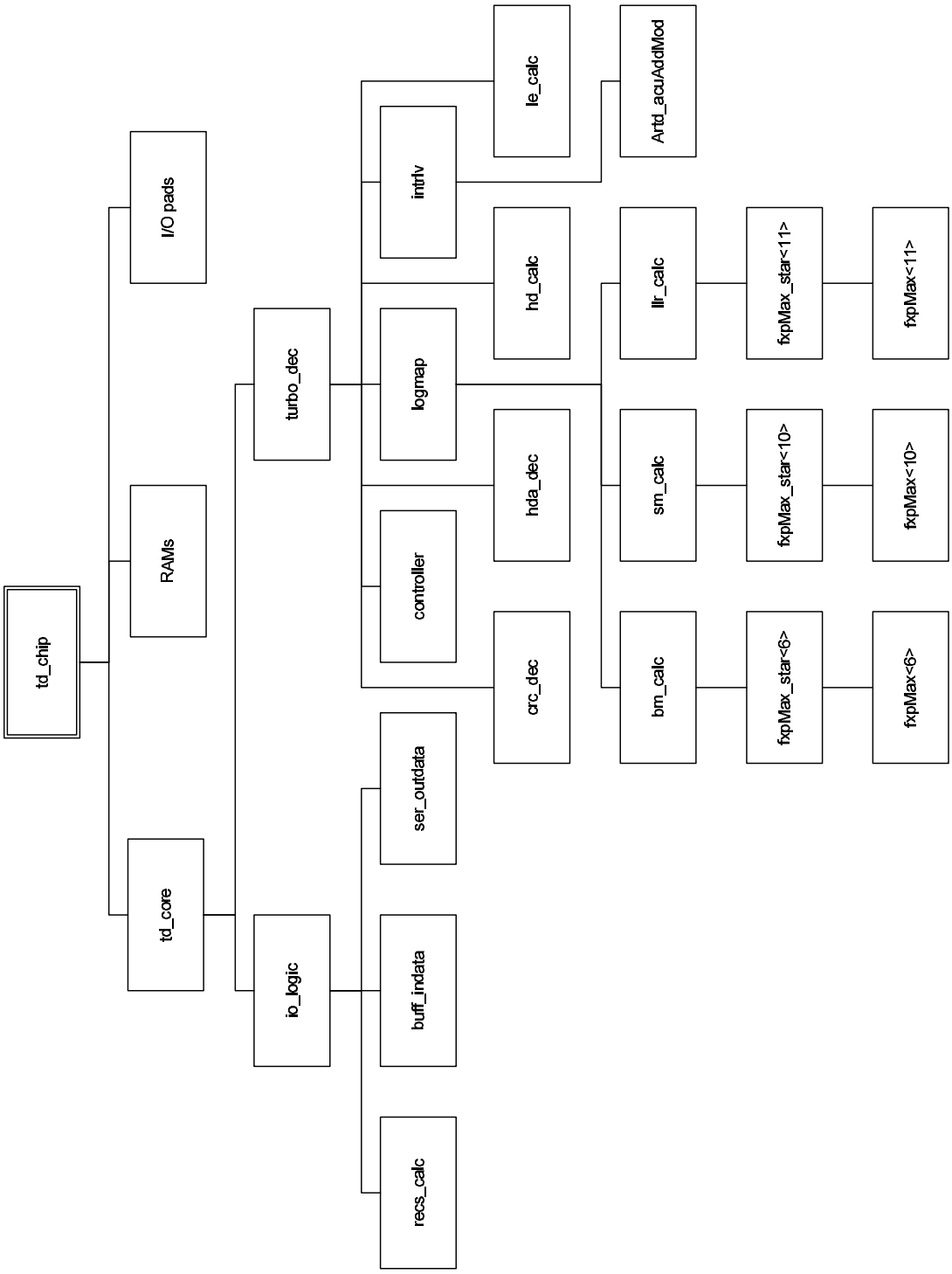


Figure 5.5: design hierarchy of the VHDL-based turbo decoder (Design units, except the black-box RAM modules, are equivalent to VHDL entities)

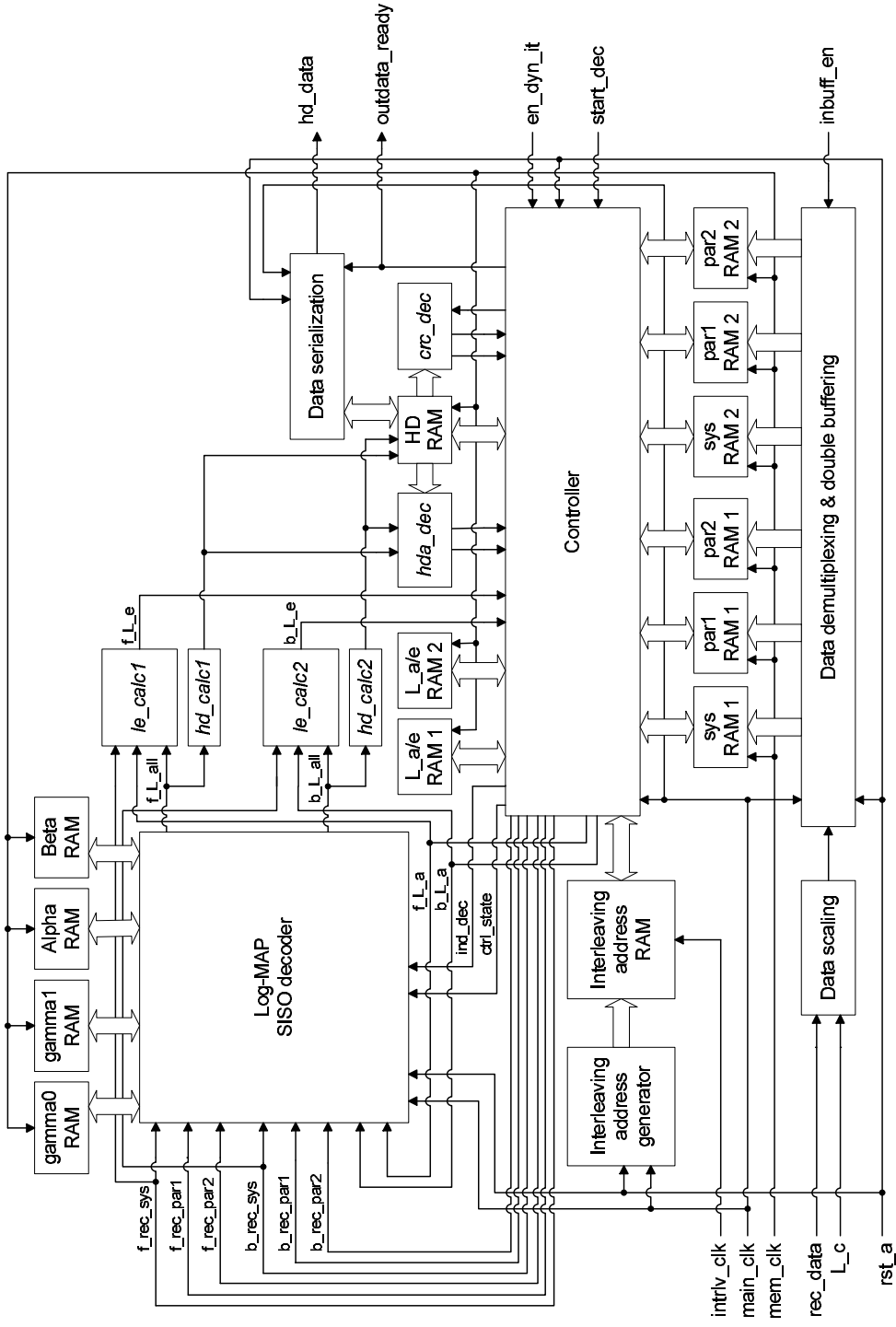


Figure 5.6: Block diagram of the new turbo decoder (*td_chip*)

turbo_dec

The main turbo decoder logic (*turbo_dec*) contains instances of the following design blocks: the control logic (*controller*), the Log-MAP SISO decoder (*logmap*), the 3G interleaving/deinterleaving address generator (*intrlv*), the hard decision calculation logic (*hd_clac*), the extrinsic info calculation logic (*le_calc*), the CRC decoding logic (*crc_dec*), and the hard decision bit comparison logic (*hda_dec*). The *logmap* unit is instantiated only once by applying resource sharing. There are two instances of both *hd_clac* and *le_calc* units, since the Log-MAP decoder processes two half-frames in parallel.

logmap

The Log-MAP SISO decoder logic (*logmap*) instantiates three metric calculation units: the branch metrics unit (*bm_calc*), the forward/backward state metrics unit (*sm_calc*), and the LLR estimation unit (*llr_calc*). Each of the three metric calculation units has instances of a suitable fixed-point version of the max* calculation logic (*fxpMax_star<xx>*), where *xx* equals 6, 10, or 11 bits and represents the fixed-point length of the max* inputs. Each of the *fxpMax_star<xx>* units instantiates a compatible fixed-point version of the max calculation logic (*fxpMax<xx>*). Since the Log-MAP decoder processes two half-frames in parallel, it has two instances of each of the three metric calculation units. Figure 5.7 shows the block diagram for the Log-MAP SISO decoder (*logmap*). Figure 5.8 illustrates the relation between trellis states and their next states for both 0 and 1 input bits of a 3G RSC encoder (code rate = 1/2, $\mathbf{g} = (13, 15)_8$); also shown are the corresponding output parity symbols (for BPSK modulation, -1 and $+1$ are equivalent to parity bits 0 and 1, respectively). State transitions and their output parity mappings are stored in LUTs and retrieved (by the Log-MAP decoder control logic) for state and branch metric calculations. Each *sm_calc* unit is shared between the forward and backward recursions.

bm_calc

Figure 5.9 shows the block diagram for the branch metrics calculation (*bm_calc*) unit. The *bm_calc* unit is a parallel implementation of the branch metrics described by (3.6) (see also supporting equations (3.16) and (3.18), where max is replaced by max*). In the *bm_calc* block diagram, *par_b0[state]* and *par_b1[state]* represent parity symbols associated

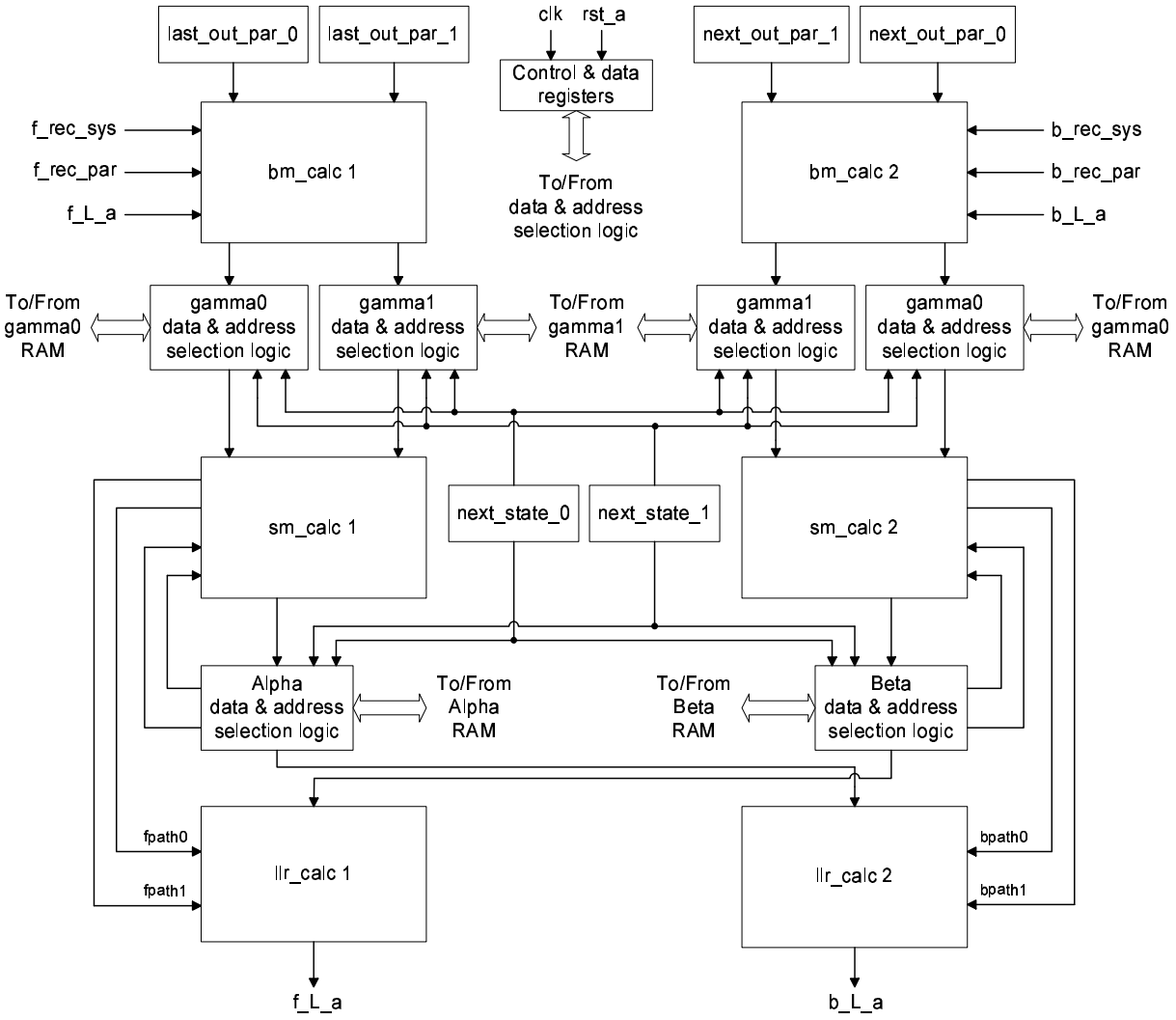


Figure 5.7: Block diagram of the Log-MAP SISO decoder (*logmap*)

with trellis state transitions from a state indexed by *state* with input bits of 0 and 1, respectively. These values (shown in Figure 5.8) are retrieved from LUTs in the Log-MAP decoder (as mentioned earlier).

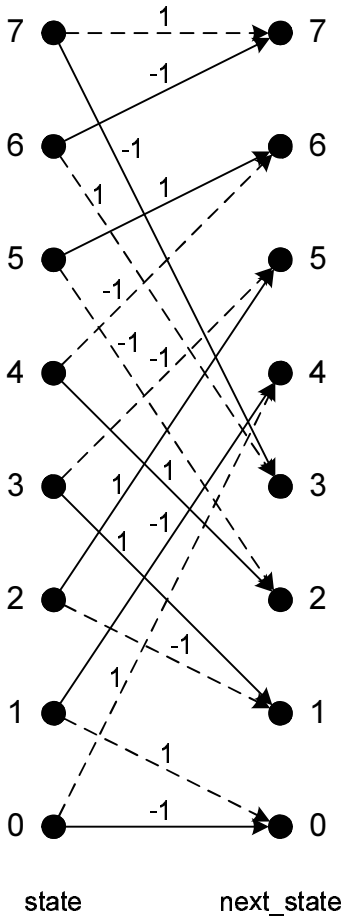


Figure 5.8: RSC encoder state transitions and their corresponding output parity symbols; $\mathbf{g} = (13, 15)_8$ (solid lines represent 0-input bit transitions and dashed lines represent 1-input bit transitions)

sm_calc

Figure 5.10 shows the block diagram for the state metrics calculation (*sm_calc*) unit. The *sm_calc* unit is a parallel implementation of both forward and backward static metrics described by (3.9) and (3.10), respectively, where max is replaced by max*. In addition, *sm_calc* generates forward or backward path metrics for the *llr_calc* unit.

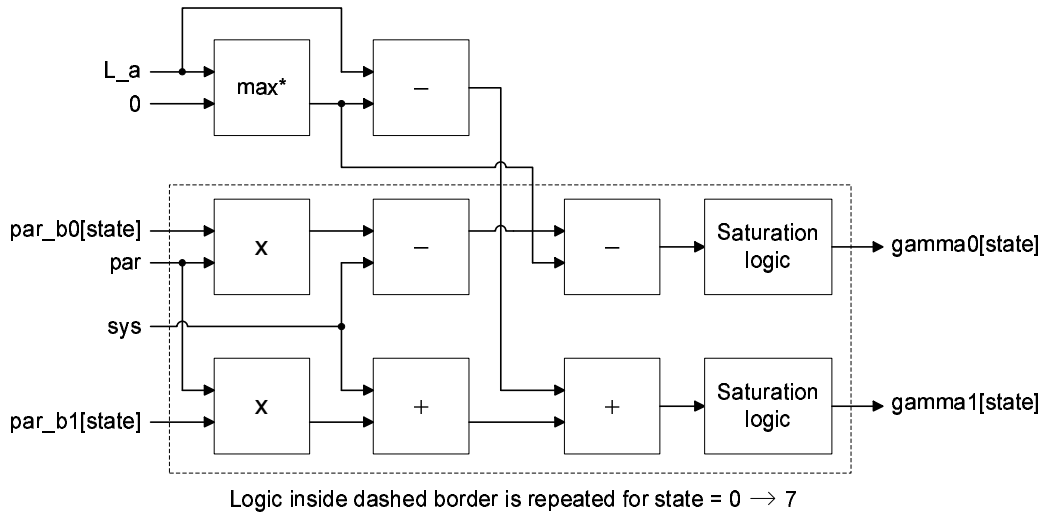


Figure 5.9: Block diagram of the branch metrics calculation (*bm_calc*) unit

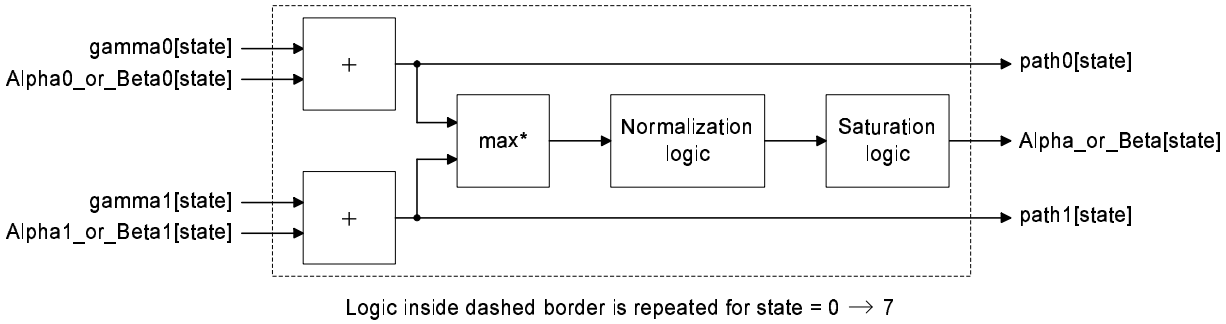


Figure 5.10: Block diagram of the state metrics calculation (*sm_calc*) unit

llr_calc

Figure 5.11 shows the block diagram for the LLR calculation (*llr_calc*) unit. The *llr_calc* unit is a parallel implementation of the log-likelihood ratio described by (3.11), where max is replaced by max*. The 8-input max* parallel structure, in the *llr_calc* block diagram, is similar to the parallel max structure in Figure 5.3, except that max units are replaced by max* units.

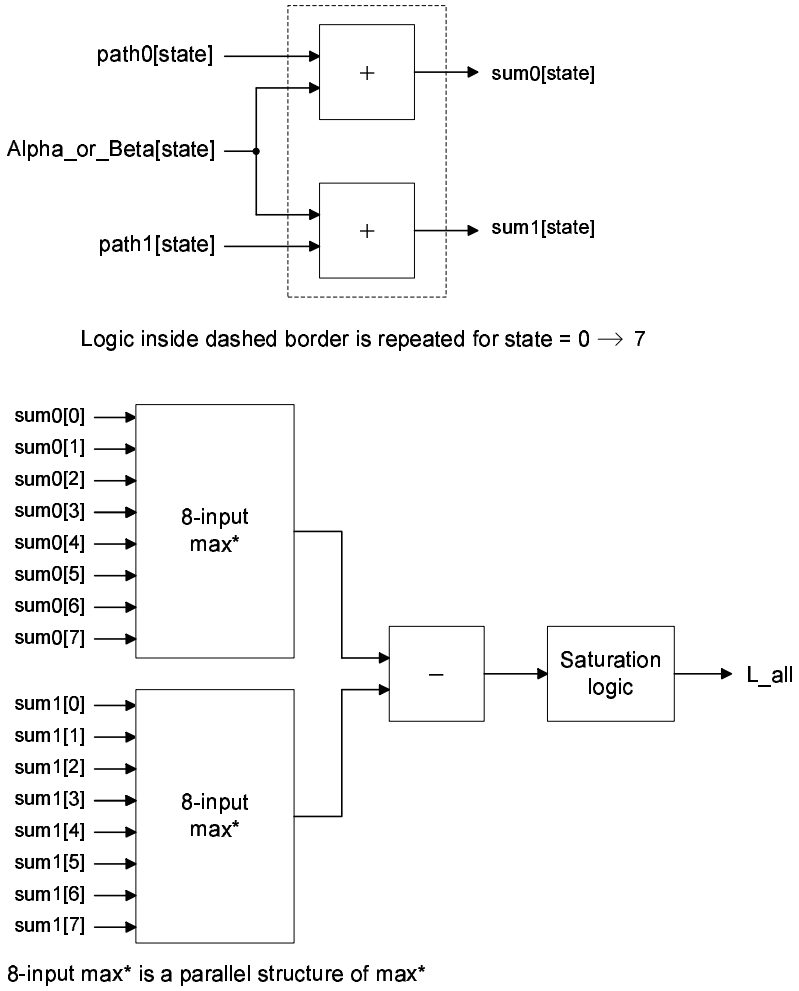


Figure 5.11: Block diagram of the LLR calculation (*llr_calc*) unit

hda_dec

The *hda_dec* unit compares two (current HD) bits generated by the two *hd_calc* units and two corresponding (previous HD) bits read from the HD buffer and generates two decision bits (0 if compared bits are the same and 1 otherwise). At each cycle of valid HD bits, the controller uses the two comparison bits to calculate HDD and DHDD values required by

the CRC-HDD algorithm.

crc_dec

At the end of each iteration, the *crc_dec* unit calculates CRC checksum for the buffered HD frame. After being instructed by the controller ($start\_crc\_dec = 1$), the *crc_dec* reads bit by bit from the HD buffer until the CRC checksum is generated. When the controller receives a valid signal ($crc\_out\_valid = 1$) from *crc_dec*, it compares the CRC checksum against zero (as required by the CRC-HDD algorithm).

intrlv

The interleaving address generation (*intrlv*) unit is based on the cdma2000 specification for turbo interleavers [11]. W-CDMA turbo interleavers [12] differ from cdma2000 interleavers in the exact specification of intra- and inter-row permutations and the matrix dimensions of the mother interleavers. The general properties of 3G interleavers were discussed in Section 2.4. For cdma2000, the mother interleavers have matrix dimensions of 32 rows by n columns, where n is an integer between 4 and 10. The following are the steps required by the 3G interleaving address generation algorithm [4]:

1. shuffle rows in bit reversal order (*i.e.*, 0, 16, 8, ..., 23, 15, 31)
2. for each row, permute elements according to: $x(j+1) = (x(j) + c) \mod 2^n$, where $x(0) = c$ and c is a row-specific value from a table lookup
3. read the new addresses column-by-column and exclude (prune) invalid addresses from the mother interleaver to get the desired interleaver size.

In our implementation of the 3G interleaver, these three steps are looped on a cycle-by-cycle basis rather than performing complete matrix operations in sequence; hence, a new mother interleaver address is calculated at every clock cycle (either passed to output or pruned) until all the mother interleaver addresses are generated. With this hardware implementation, the interleaver latency (in cycles) is equal to the mother interleaver size (*i.e.*, ≥ 512 for cdma2000 interleavers). The interleaving addresses are generated only once after starting (or restarting) the turbo decoder, and the addresses are stored by the *intrlv* unit in

the interleaving address RAM. As long as the turbo decoder receives frames of equal length, the interleaving address RAM can be used for all subsequent interleaving/deinterleaving operations of the turbo decoder. A variable-size 3G interleaving address generation unit, that can be set to any interleaver size less than 8192, is also implemented (compatible with the W-CDMA requirement that interleaver/frame length is between 40 and 5114 bits). The *intrlv* unit instantiates a modulo-2^n addition unit (*Artd_acuAddMod*) for intra-row permutations (step 2 of the 3G interleaving address generation algorithm).

controller

The *controller* unit includes the control logic for the turbo decoder units and memories. It controls memory access for the following: reading double-buffered soft input data, reading from and writing to extrinsic info buffers, storing hard decisions into HD buffer, and reading interleaving addresses from interleaving address memory. It also creates Log-MAP decoder control cycles and multiplexes/demultiplexes data from/to the shared Log-MAP decoder (*logmap*) unit. In addition, the controller includes control logic for the CRC-HDD dynamic-iterative algorithm including: checking CRC checksum, incrementing/buffering HDD values, calculating/checking DHDD value, halting decoding for an appropriate number of cycles if the dynamic-iterative technique is enabled and any one of the stopping/cancellation conditions is met. Figure 5.12 demonstrates how the *controller* handles interleaving and deinterleaving required by the turbo decoder. To allow for data read/write to occur on the same cycle as reading the interleaving address, three clocks are required: main clock (*main_clk*), interleaving address memory clock (*intlv_clk*), and data memories clock (*mem_clk*). Interleaving requires writing data into memory in a sequential order then reading it according to interleaving addresses, while deinterleaving requires writing data according to interleaving addresses then reading it in a sequential order.

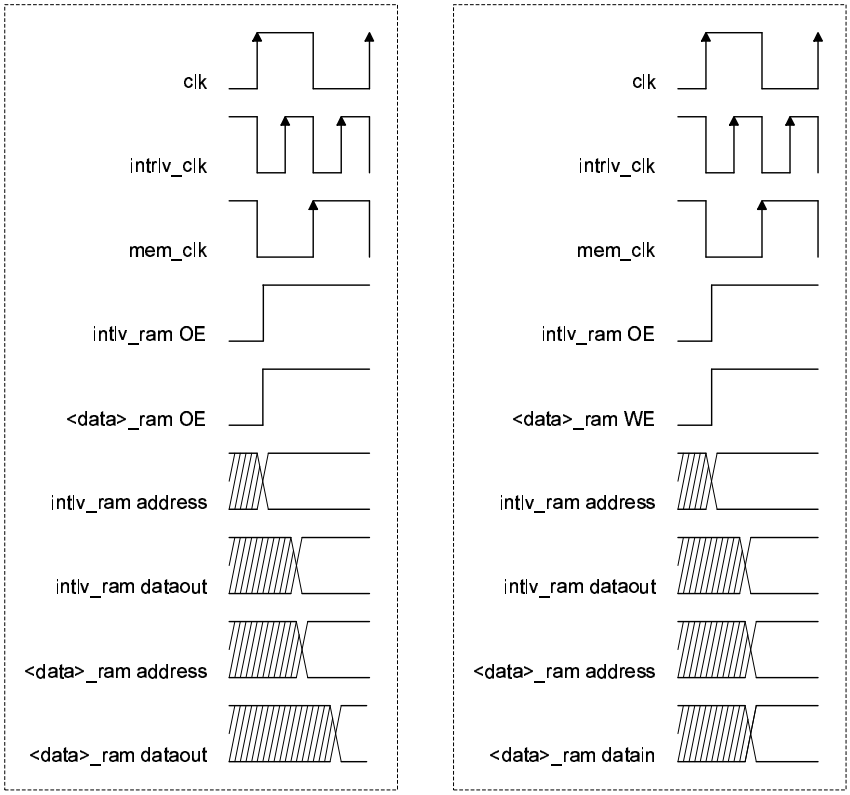


Figure 5.12: Timing diagram for data interleaving/deinterleaving by the *controller* unit

Chapter 6

Synthesis Results

6.1 Synthesis Results for a 0.18μm CMOS Standard-Cell Based Turbo Decoder

The proposed turbo decoder is coded in VHDL, and then synthesized and optimized into standard cells from the *Virtual Silicon® Technolgy* (VST) library, targeting the TSMC® 0.18μm six-metal CMOS process. The delay and area results are produced using *Synopsys® Design Compiler*. To calculate power consumption, back-annotated switching activity data are extracted from RTL simulation; then *Synopsys® Power Compiler* uses this data to calculate power consumption. Soft-input data are generated for RTL and gate-level simulations from an AWGN channel model with an SNR range of -1 to 2 dB, highlighting various channel conditions.

6.1.1 Applying the Architectural-Level Techniques to the Turbo Decoder

The new energy-efficient design of the turbo decoder is implemented in incremental steps, applying at each step some of the architectural-level techniques discussed in Chapter 5. The design steps are detailed as follow:

1. In the basic design, each of the low-level resources (BM, SM, and LLR units) deals

with one state at a time (each trellis bit stage needs eight states). In addition, the basic design uses two separate Log-MAP SISO decoders (each processing a half-iteration). In each half-iteration, forward and backward recursions are executed in a serial order. The state metrics are normalized by subtracting maximum SM (calculated at the previous bit stage) from each input state metric to the SM unit. A ROM-based interleaver is used in this basic design.

2. In this design step, the throughput is doubled by processing two half-frames in parallel (one is working in the forward direction while the other is working in the backward direction, and vice versa at the next recursion). However, this comes at the cost of doubling the number of BM, SM, and LLR units.

3. To further increase the throughput, BM, SM, and LLR units are extended to process all the eight states at one cycle using parallel elements of the *add-compare-select* (ACS) logic. For SM units, since all state metrics are generated in parallel, the maximum SM (used for SM normalization in the next bit stage) can also be calculated at the same clock cycle. A parallel structure for max calculation is introduced to reduce critical-path delay (using three stages of max calculation instead of seven in the serial structure).

4. To reduce propagation delay and area of the maximum-value calculation (max) logic introduced in the SM units, a simpler normalization method is used. In this normalization method, a constant value is subtracted from all state metrics if one of the state metrics exceeds a threshold. This method is area, power, and delay efficient. To further reduce critical-path delay, power, and area of the turbo decoder, a new design of the $\max^*$ logic (used in BM, SM, and LLR units) uses a double-length (32 entries) LUT (by including entries for both positive and negative inputs) to eliminate the complex absolute-value calculation (abs) logic. Double buffering is introduced in this stage to increase throughput of the turbo decoder.

5. In the last design step, resource sharing is introduced: at one level, two SM units are shared between both forward and backward recursions; at a higher level, one Log-MAP SISO decoder is shared between two half-iterations (instead of two SISO

decoders, one working while the other being idle). For flexibility, a 3G-compatible RAM-based interleaver is incorporated into the design.

Table 6.1 summarizes the turbo decoder design characteristics measured at each incremental step in the design process. By looking closely at the results, steps 1 and 2 do not achieve the minimum data rate of 2 Mb/s required for 3G wireless data applications. Step 3 achieves that goal (with a maximum data rate of 3.7 Mb/s) but at the expense of a much larger area and power consumption and even a longer critical delay (which limits the maximum clock frequency). After applying the techniques introduced in steps 4 and 5, when compared to step 3 (the highly-complex parallel structure), area is reduced by about 71% and power consumption by about 30% (energy by about 48%). Now, the decoder can operate at data rates up to 5 Mb/s (with five decoding iterations), improving the decoder energy efficiency by about 48% compared to the basic implementation (*i.e.*, step 1), at 2.9 nJ/b/iteration. With the 3G-compatible RAM-based interleaver, the turbo decoder can work with 3G block lengths (40–5114 bits).

Table 6.1: Turbo decoder characteristics at different stages of the design process

Design step	Area (mm^2)	Power (mW)	Max. clock freq. (MHz)	Data rate (Mb/s)	Energy (nJ/b/iter.)
1	0.155	8.44	100	0.3	5.63
2	0.263	17.24	100	0.6	5.75
3	2.250	103.62	88.5	3.7	5.60
4	1.137	76.68	100	5.0	3.01
5	0.658	72.39	100	5.0	2.90

6.1.2 Energy Reduction with the CRC-HDD Dynamic-Iterative Technique

The CRC-HDD dynamic-iterative method, proposed in Chapter 4, is applied to the architecturally-optimized design, discussed in Section 6.1.1, to further reduce power/energy consumption

by stopping or cancelling decoding at an earlier point. The CRC-HDD algorithm was shown in Figure 4.7. For high SNRs (*i.e.*, good channel conditions), the CRC logic is able to stop redundant iterations after detection of a correctly-decoded frame (CRC checksum = 0). For low SNRs (*i.e.*, poor channel conditions), the HDD logic (based on observing changes in hard Decision bit-wise differences) cancels execution of remaining iterations that never leads to correct frame decoding. The CRC-HDD method proves to have marginal effect on decoding performance (as discussed in Chapter 4). When there are no more active iterations for processing, the turbo decoder goes into a sleep (power-down) mode to reduce switching activity, and hence dynamic power.

Table 6.2 summarizes the turbo decoder design characteristics before and after applying the CRC-HDD dynamic-iterative technique. The first row shows results produced after applying the afore-mentioned architectural-level techniques, with a static number of five decoding iterations applied to all data frames. The second row shows results after applying the CRC-HDD dynamic-iterative logic, with an upper limit of five iterations. The results show an additional power/energy reduction by about 26% after adding the CRC-HDD dynamic-iterative logic, compared to the static-iterative design. The final design has an enhanced energy efficiency of 2.16 nJ/b/iteration (about 62% total energy reduction, compared to the basic implementation). From Table 6.2, we find that the additional dynamic-iterative logic has a small effect on the turbo decoder core area.

Table 6.2: Turbo decoder design characteristics before and after applying the CRC-HDD dynamic-iterative technique

Design phase	Area (mm^2)	Power (mW)	Max. clock freq. (MHz)	Data rate (Mb/s)	Energy (nJ/b/iter.)
Static-iterative	0.658	72.39	100	5.0	2.90
CRC-HDD dynamic-iterative	0.705	53.89	100	5.0	2.16

6.1.3 Effect of Memory Integration on the Turbo Decoder

The Virage® Custom-Touch Memory Compiler (CTMC) tool is used to generate customized 0.18μm CMOS RAM models for the turbo decoder. The compiled RAM models are used for both simulation and synthesis of the VHDL-based turbo decoder design. Behavioral and VITAL models are used for behavioral and gate-level simulations respectively, whereas Synopsys database files are used for synthesis (including area and power calculations). Table 6.3 lists the main characteristics of the Virage-complied RAMs used in the turbo decoder design.

Table 6.3: Memory characteristics for the turbo decoder design

Memory units	How many	bit-size (rows × cols.)	Access type
BM	2	$N \times 56$	dual-port
SM	2	$(N/2) \times 72$	single-port
Extrinsic	2	$N \times 6$	dual-port
Interleaving	1	$N \times \lceil \log_2 N \rceil$	dual-port
Soft-input	6	$N \times 4$	dual-port
HD	1	$N \times 2$	dual-port

From Table 6.3, it can be seen that all RAMs, except SM RAMs, are chosen as dual-port to allow for data parallelism (*i.e.*, processing two half-frames at a time, as discussed in Chapter 5). Also, we notice that the HD RAM is 2-bit wide although only one bit is used for hard decisions; this is due to the fact that Virage® CTMC has a limit of 2-bit minimum width for its complied memories. Nonetheless, the memory-reduction technique discussed in Section 4.4.1, part 1, is applied to prove that the additional buffer for storing previous HD data can be avoided. In addition, the logic reduction techniques discussed in Section 4.4.1, part 2, are applied to the design to reduce the HDD circuit complexity.

A standard-cell based 0.18μm CMOS version of the turbo decoder is synthesized with compiled memory modules for a block size of 128-bit. The synthesis results reveal that the design (before routing and adding pads) has an area of about 1.93 mm^2, out of which the memory blocks occupy about 63%. To reduce memory area of the turbo decoder, techniques

such as the sliding-window algorithm [55] can be used, but this thesis focusses on energy reduction (in addition to area and throughput optimizations) of the turbo decoder core. The sliding-window algorithm while reducing memory sizes for branch and state metrics, it sacrifices some decoding performance. The algorithm effect on power consumption can be studied only after implementation (we would still have the same number of memory accesses plus the additional complexity of the algorithm).

The new turbo decoder design has an additional feature of enabling or disabling its dynamic-iterative behaviour through an `en_dyn_it` signal, hence allowing for calculation of power consumption for both static (`en_dyn_it` = 0) and dynamic (`en_dyn_it` = 1) iterative cases. Table 6.4 shows the turbo decoder static and dynamic power consumptions (with a nominal 1.8V supply) for both logic states of the `en_dyn_it` signal.

Table 6.4: Power/Energy consumption for the static/dynamic-iterative turbo decoder

`en_dyn_it` =	Dynamic Power (mW)	Static Power (μW)	Total Power (mW)	Energy (nJ/b/iter.)
0 (static-iterative)	165.063	24.48	165.087	6.60
1 (dynamic-iterative)	111.223	24.48	111.247	4.45

From Table 6.4, it is evident that static, or leakage, power represents only 0.015% and 0.022% of total power for static and dynamic iterative decoders, respectively, and is independent of the dynamic-iterative technique being applied or not. This leads to the conclusion that the dynamic-iterative technique, affecting only dynamic power consumption, still plays an important role in the reduction of power, and energy, consumption of the turbo decoder in a deep sub-micron technology such as 0.18μm CMOS.

The numbers in table 6.4 indicate that the power/energy reduction produced by the CRC-HDD dynamic-iterative technique is about 33%, which is larger than the case when the memories are not taken into account (since the technique reduces the switching activities for both arithmetic calculations and memory accesses). For the 128-bit turbo decoder, results show that for both cases of the `en_dyn_it` signal, the memory consumes about 45% of total power consumption. Although the effects of memory access on power con-

sumption are not measured when applying the architectural-level techniques discussed in Chapter 5 (results shown in Table 6.1), a higher power/energy reduction at that design phase can be predicted if the effects of memory access are included. Taking this into consideration, the additional dynamic-iterative reduction of 33% would lead to a total energy reduction of at least 65%. This is analyzed as follows: 0.52 (energy ratio after applying the architectural-level techniques) $\times$ 0.67 (energy ratio after applying the proposed CRC-HDD dynamic-iterative technique) $\approx$ 0.35 (energy ratio between the basic implementation and the architecturally-optimized dynamic-iterative implementation).

6.2 CMOS Layout of the Turbo Decoder Including Memory Blocks

The layout is done for the 128-bit turbo decoder using Cadence® tools. The synthesized gate-level design is saved from Synopsys® Design Compiler as a Verilog netlist file and imported into Cadence design tools. The Cadence tools involved are: Encounter (floorplanning, power planning, and placement of memory blocks and pads); PKS (standard cell placement and clock tree synthesis); Silicon Ensemble (power and net routing); Design Framework II (hosting a set of tools including: Virtuoso for schematic/layout editing, Diva for DRC and LVS, and stream file (GDSII) generation). Mentor Graphics® Calibre DRC is also used for DRC.

The floorplan report for the chip layout, generated by Silicon Ensemble is shown in Figure 6.1. As shown in the report, the total cell area (standard cells + memory blocks + pads) has a utilization (total cell area / die area) of 77.82%; the remaining die area is occupied by power and net routing. Out of a total cell area of about 3,800,388 μm^2, there is a standard cell area of about 738,606 μm^2 (19.44%), and an area occupied by memory blocks of about 1,205,282 μm^2 (31.71%); the remaining cell area (48.85%) is occupied by power, I/O, and corner pads. The die area is 4,883,700 μm^2 (2,230μm $\times$ 2,190μm), or about 4.88 mm^2. The turbo decoder has an active area (*i.e.*, the area before adding pads) of about 3.03 mm^2. Figure 6.2 shows the chip layout of the turbo decoder.

```
VERIFY FLOORPLAN STATUSONLY ;
#. VERIFY FLOORPLAN STATUSONLY ;
BEGIN Floorplan Status Report:
core: 254 rows; length 169091340; area 1041602654400.
core: 25174 cells; length 119903520; area 738605683200; 71% of rows.
core: 25174 placed cells; length 119903520; area 738605683200; 71% of rows.
pad: 0 rows; length 0; area 0.
pad: 304 cells; length 6960000; area 1635600000000; 0% of rows.
pad: 304 placed cells; length 6960000; area 1635600000000; 0% of rows.
corner: 0 rows; length 0; area 0.
corner: 4 cells; length 940000; area 220900000000; 0% of rows.
corner: 4 placed cells; length 940000; area 220900000000; 0% of rows.
VLC_SITE_hdss1_64x72cm4: 0 rows; length 0; area 0.
VLC_SITE_hdss1_64x72cm4: 2 cells; length 355240; area 231268344800; 0% of rows.
VLC_SITE_hdss1_64x72cm4: 2 placed cells; length 355240; area 231268344800; 0% of rows.
VLC_SITE_hdss2_128x56cm4: 0 rows; length 0; area 0.
VLC_SITE_hdss2_128x56cm4: 2 cells; length 830880; area 493924924800; 0% of rows.
VLC_SITE_hdss2_128x56cm4: 2 placed cells; length 830880; area 493924924800; 0% of rows.
VLC_SITE_hdss2_128x2cm8: 0 rows; length 0; area 0.
VLC_SITE_hdss2_128x2cm8: 1 cells; length 283280; area 35143716800; 0% of rows.
VLC_SITE_hdss2_128x2cm8: 1 placed cells; length 283280; area 35143716800; 0% of rows.
VLC_SITE_hdss2_128x4cm8: 0 rows; length 0; area 0.
VLC_SITE_hdss2_128x4cm8: 6 cells; length 1699680; area 268923369600; 0% of rows.
VLC_SITE_hdss2_128x4cm8: 6 placed cells; length 1699680; area 268923369600; 0% of rows.
VLC_SITE_hdss2_128x6cm4: 0 rows; length 0; area 0.
VLC_SITE_hdss2_128x6cm4: 2 cells; length 830880; area 114711292800; 0% of rows.
VLC_SITE_hdss2_128x6cm4: 2 placed cells; length 830880; area 114711292800; 0% of rows.
VLC_SITE_hdss2_128x7cm4: 0 rows; length 0; area 0.
VLC_SITE_hdss2_128x7cm4: 1 cells; length 415440; area 61310635200; 0% of rows.
VLC_SITE_hdss2_128x7cm4: 1 placed cells; length 415440; area 61310635200; 0% of rows.
Total cell area: 3800387967200; die area: 4883700000000.
Utilization for all cells: 77.82%.
END Floorplan Status Report
```

Figure 6.1: Floorplan report for the turbo decoder chip generated with Silicon Ensemble; Length (Area) units are 0.001μm (0.000001μm^2)

6.3 Comparison with State-of-the-Art Turbo Decoders

To highlight the energy efficiency of the techniques we applied to the proposed turbo decoder (both static-iterative and new CRC-HDD dynamic-iterative), the turbo decoder implementation is compared to fabricated chips in 0.18μm CMOS that have recently been presented in the literature. Both works are published by teams from Bell Labs, Lucent

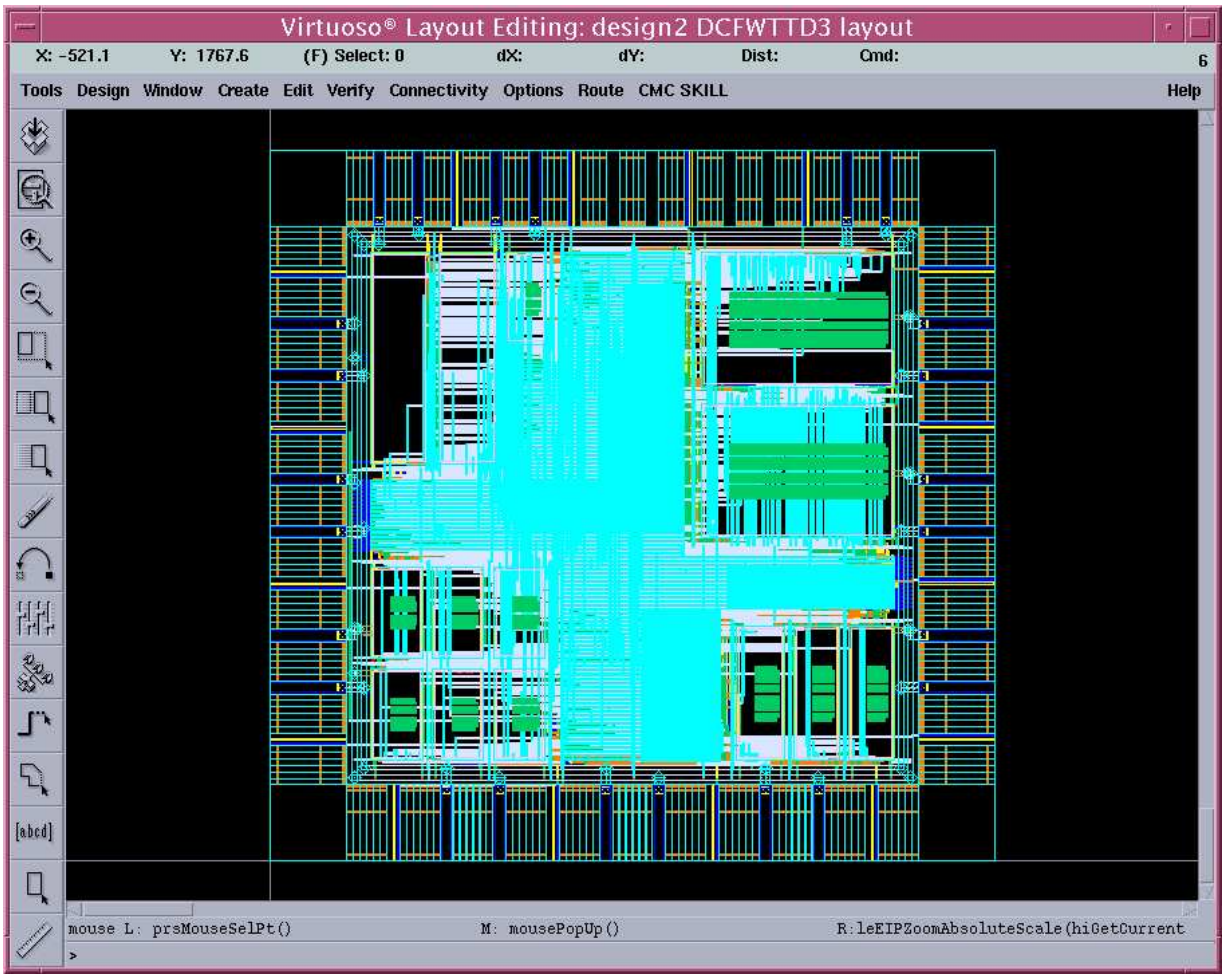


Figure 6.2: CMOS layout of the 128-bit turbo decoder (Snapshot from Virtuoso Layout Editor)

Technologies® [30, 34].

Table 6.5 presents the design characteristics for the turbo decoders in the literature and our proposed turbo decoder implementation. The different lines in the table for each chip refer to different energy consumptions and data rates resulting from changing clock frequency and/or number of iterations. The two chips work with different 3G block lengths by applying a sliding window of 40 bits, hence reducing memory size. The chip in [34]

supports both voice and data using a unified turbo/Vitebi architecture. As seen from the table, our proposed turbo decoder design is superior in energy efficiency (at both static and iterative modes), compared to other state-of-the-art turbo decoders (by about 34% or more compared to [30] and by about 54% or more compared to [34]). Furthermore, the active area (area without pads) of the proposed turbo decoder is smaller than the active areas of the two Bell Labs' chips.

Table 6.5: Key characteristics of turbo decoder chips from recent research work and the proposed turbo decoder implementation

Chip or design	Active area (mm^2)	Power (mW)	Clk. freq. (MHz)	Data rate (Mb/s)	No. of iters.	Energy (nJ/b/iter.)
[30]	14.5	1450	145	24	6	10.0
		956	86.4	10.8	8	11.1
		228.6	20	2	10	11.4
[34]	9	58	17.5	0.384	10	15.1
		292	88	2.048	10	14.3
Proposed design:						
a) static-iterative	3	165.1	100	5	5	6.6
b) CRC-HDD dynamic-iterative		111.2	100	5	≤ 5	4.5

The comparison between the new turbo decoder design (resulting from VHDL synthesis and measured using Synopsys Power Compiler) and the two fabricated chips from Bell Labs is approximate rather than accurate. The output pads are represented by 20pF loads, to have as close resemblance as possible to a real chip load when measuring power consumption. Assuming a 10–20% margin of error between results from synthesis/simulation and physical chip testing, the new design is still superior in terms of energy efficiency. The error margin is a function of different parameters including fabrication technology, testing environment, accuracy of synthesis tools, standard cell libraries, and memory models. In

addition to energy efficiency, data throughput requirement is another important parameter in the design of turbo decoders, *e.g.*, the new 3GPP-HSDPA standard [31] requires a high data throughput of 10.8 Mb/s.

Chapter 7

FPGA Design and Testing of a Turbo Codec Prototype

A prototype for the new turbo codec (encoder/decoder) design is implemented on a Xilinx® XC2V6000 FPGA chip. The Xilinx chip is tested for functionality using a Rapid Prototyping Platform (RPP) supplied by the Canadian Microelectronics Corporation (CMC).

7.1 The Turbo Codec Design

Figure 7.1 shows the block diagram of the new turbo codec system implemented on the Xilinx FPGA. A pseudo-noise (PN) sequence generator is implemented with a Linear-Feedback-Shift-Register (LFSR) to generate a pseudo-random sequence of input bits. A 3G-compatible CRC encoder adds CRC bits to each data frame. A 3G turbo encoder encodes each input frame sequence according to a generator polynomial $\mathbf{g} = (13, 15)_8$ and a code rate $R = 1/3$. The encoder outputs are BPSK-modulated (+1 for an encoded bit 1 and −1 for an encoded bit 0). To simplify testing of the codec system, a zero AWGN noise is assumed, and the channel reliability factor (L_c), used in soft-input data scaling, is calculated at $E_b/N_0 = 2$ dB. The encoder outputs are stored in three memories for systemic and parity data. A 3G-compatible interleaving address generator calculates at startup interleaving addresses for both encoder and decoder. The generated interleaving addresses are stored into two RAM modules (two identical copies: one for the encoder and

the other for the decoder). Both the encoder systemic data and the decoder soft-input data are double-buffered to eliminate latency between consecutively-decoded frames. A divide-by-two clock generator is used to generate two anti-phase clocks (one for encoder/decoder logic modules and the other for non-interleaving memories) from a faster clock (used to clock interleaving memories). A data synchronization unit is used to generate start signals for both encoder and decoder components.

The CRC-HDD dynamic-iterative turbo decoder decodes each frame and the output hard-decision (HD) data is stored in the HD memory. Other memories are used by the decoder for storing branch and state metrics and extrinsic information. For testing purposes, a special unit compares the data stored in the decoder HD memory to the corresponding data stored in the encoder systemic memory, and the number of correct bits is stored in a correct-bit count buffer. At the end of each frame-decoding operation, the number of correct bits is used to check for a correctly-decoded frame, and a correct-frame count is incremented by one if the number of correct bits is equal to the frame length. The system stops encoding/decoding after completing the pre-assigned number of frames, and the correct-frame count must stop counting after decoding the last frame. The turbo codec system is first written in VHDL, and then synthesized using Synopsys® FPGA Compiler II. Memory core generation, placement, routing, and FPGA bit-file generation are done using Xilinx tools.

7.2 FPGA Testing with the CMC RPP Environment

The turbo codec system is implemented on a Xilinx® Virtex-II XC2V6000 chip. The XC2V6000 FPGA is mounted on the ARM® Integrator/LT-XC2V6000 logic tile installed on the RPP V1.0 [56] that is supplied by CMC. The RPP V1.0 is shown in Figure 7.2 and comprises an ARM® Integrator/AP ASIC Development Motherboard as well as other supporting components. The turbo codec bit-file, generated using Xilinx tools, is downloaded to the XC2V6000 FPGA from a PC connected to the RPP system. The turbo codec correct-frame count is visually checked using LEDs on the ARM Integrator/IM-LT1 interface module and the ARM Integrator/LT-XC2V6000 logic tile. The turbo codec FPGA can be asynchronously-reset with a push-button on the logic tile.

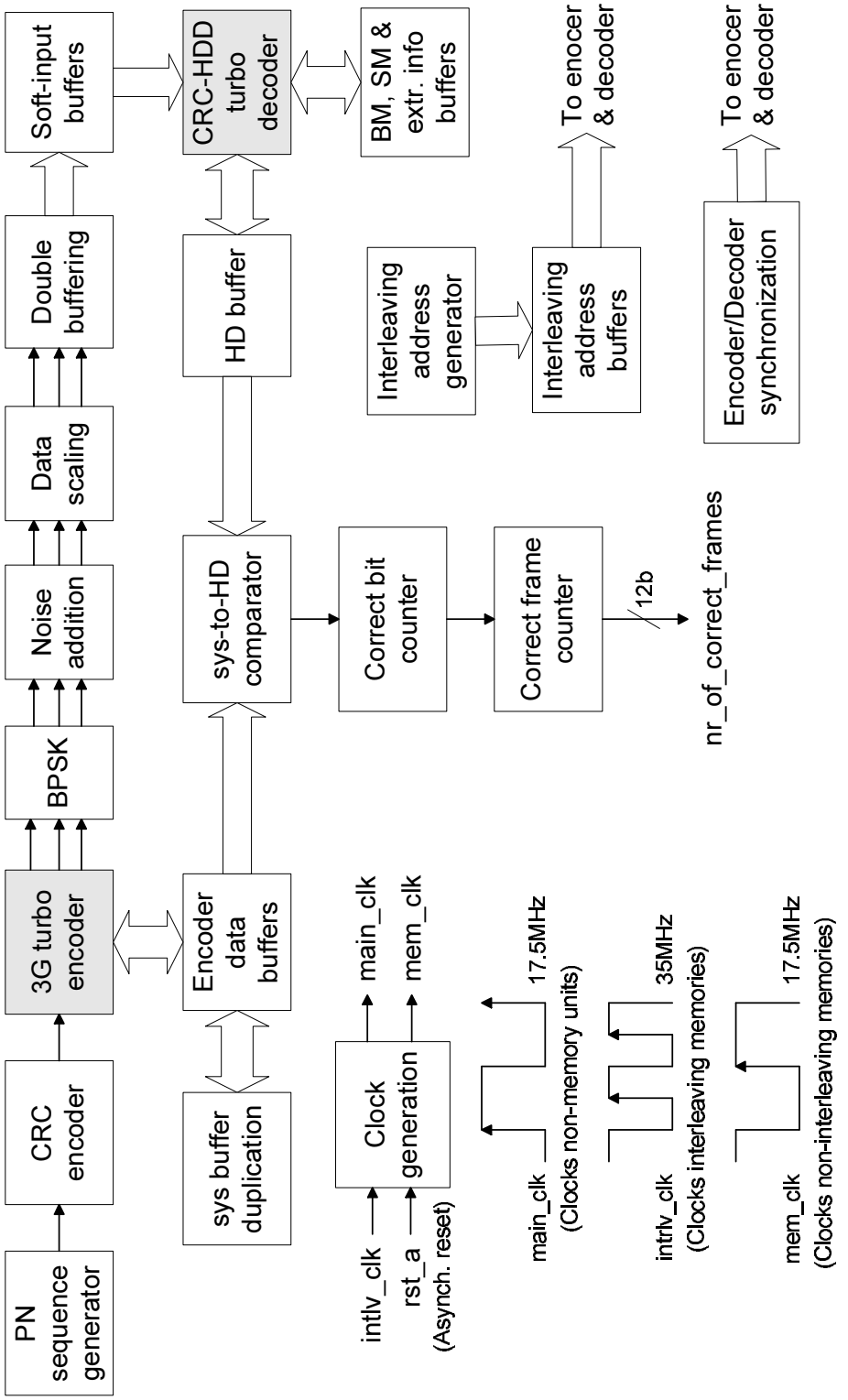


Figure 7.1: Block diagram of the turbo codec system

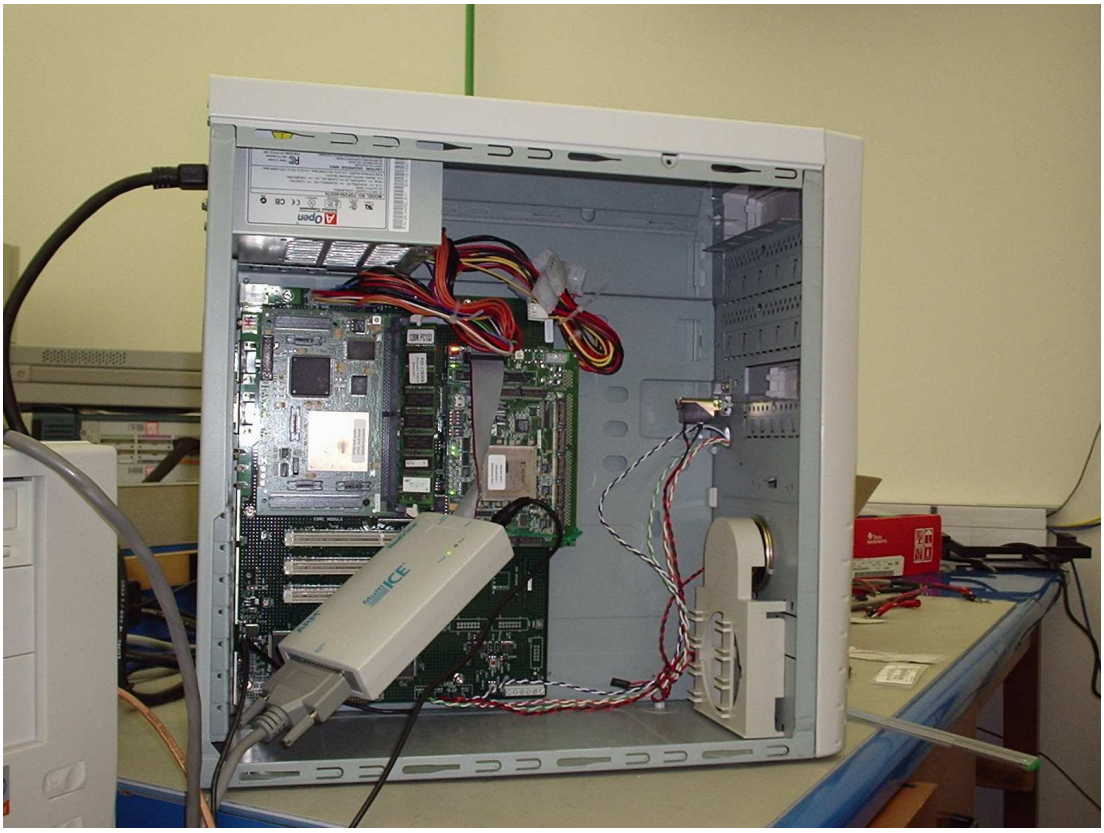


Figure 7.2: CMC Rapid Prototyping Platform

The turbo codec FPGA chip operates at a maximum clock frequency of 35MHz (the codec logic core works at 17.5MHz), hence providing a data rate of about 870 kb/s at a maximum of five decoding iterations. 4000 frames are encoded/decoded by the FPGA to check for correct functionality. The system is found to correctly decode all the encoded frames, for both 128 and 1024-bit block lengths, at an ideal noise environment (added noise = 0 and E_b/N_0 = 2 dB); hence proving feasibility and correct functionality of the proposed design for the 3G turbo decoder. Table 7.1 summarizes the key characteristics of the implemented turbo codec FPGA chip.

The goal of FPGA prototyping of the turbo codec is to test the feasibility of the new turbo decoder architecture by functional verification. The original goal of the thesis is

Table 7.1: Key characteristics of the turbo codec FPGA

Characteristic	Value
Interleaver	3G
Block length (bits)	128, 1024
CRC (bits)	12, 24 (3G)
Generator polynomial	$(13, 15)_8$
Code rate	1/3
E_b/N_0 (dB)	2
Quantization (bits)	Soft-input: 4, BM: 7, SM: 9, LLR: 10, Extrinsic: 6
Dynamic-iterative	CRC-HDD
Clock frequency (MHz)	Input: 35 {Core: 17.5, Memories: 17.5 & 35}
Number of iterations	≤ 5
Data rate (b/s)	870k
Number of frames	4000
Equivalent gate count	1,834,435

to study the effects of energy-efficient techniques for turbo decoders on a 0.18μm CMOS process, including the suggested architectural-level techniques (Table 6.1) and the new dynamic-iterative technique (Tables 6.2 and 6.4). Due to unresolved problems applying the new, and then partially-supported, digital design flow from CMC to fabricate a turbo decoder chip in 0.18μm CMOS along with memory blocks, another necessary alternative is to verify functionality of the new design using FPGA prototyping. Therefore, the power/energy study is not repeated during the FPGA design process. In addition, the CMC RPP, which is the testing environment for the FPGA turbo codec prototype, does not allow for physical measurement of power consumption.

Chapter 8

Conclusion

Turbo codes have generated a tremendous interest as a FEC technique due to its near Shannon-limit performance. Since 2000, turbo coding has been adopted as a channel coding standard for 3G wireless high-speed data services by the 3G standardization bodies (cdma2000 and W-CDMA). However, turbo decoders consume a considerable amount of energy, hence affecting their practical application to wireless terminals.

In this thesis, an energy-efficient turbo decoder was designed. A fixed-point version of the Log-MAP decoding algorithm was chosen for implementation of the constituent SISO decoder due to its optimized performance-complexity tradeoff. Two design approaches were applied to reduce energy consumption of the turbo decoder. The first approach involved the application of a novel low-complexity dynamic-iterative technique (CRC-HDD) to reduce the number of decoding iterations at both good and poor channel conditions. The majority of dynamic-iterative techniques in the literature are effective only at good channel conditions. In addition, many of these techniques involve complex operations, hence sacrificing power consumption. A turbo decoder applying the new technique was modelled in the C language, and then simulated and compared with decoders applying other low-complexity techniques. The new dynamic-iterative decoder proved to be superior in terms of iteration reduction.

The second design approach involved applying a combination of architectural-level techniques to reduce the turbo decoder energy, in addition to optimizing area and throughput. The turbo decoder was designed to meet both 3G CDMA wireless standards. The proposed

techniques were applied and results were taken at each design step to show efficiency of such techniques. Furthermore, the new CRC-HDD dynamic-iterative technique was applied to the turbo decoder implementation, adding additional improvement in energy efficiency.

The new turbo decoder was coded in VHDL, and then synthesized into a 1.8V 0.18μm CMOS standard-cell library using Synopsys Design Compiler. Power was calculated using Synopsys Power Compiler. Compiled memory modules were included for simulation and synthesis, and Cadence tools were used to generate CMOS layout of the turbo decoder. The combined design techniques (architectural-level and CRC-HDD) showed a reduction in energy consumption of the turbo decoder by about 65%. The new energy-efficient dynamic-iterative turbo decoder is 3G-compatible, and works at a maximum data rate of 5 Mb/s (with an upper limit of 5 iterations). It has an energy-efficiency of about 4.5 nJ/b/iteration, which is better than other implementations reported in the literature. A 128-bit core (standard cells plus memory blocks) occupies an area of about 3 mm^2 in a 0.18μm six-metal CMOS technology.

To prove feasibility of the proposed turbo decoder architecture, a prototype for the new turbo codec (encoder/decoder) was implemented on a Xilinx XC2V6000 FPGA chip. The turbo codec FPGA was tested using the CMC Rapid Prototyping Platform (RPP) and found to be functionally correct.

Future Work

The main goal of this work was to achieve energy efficiency for 3G-compliant turbo decoders. Some other research directions are suggested:

- Memory optimization techniques can be explored after applying the sliding window algorithm [55] to the turbo decoder.
- A dynamically-configurable (programmable) architecture can be investigated for 3G-compliant turbo decoders.
- The throughput of 5 Mb/s can be increased to higher data rates (*e.g.*, 10.8 Mb/s required by the new HSDPA standard [31]) by applying more parallelism and/or pipelining to the turbo decoder architecture (at the expense of more silicon area and energy consumption).

Bibliography

[1] C. E. Shannon, "A mathematical theory of communications—I & II," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423 & 623–656, 1948.

[2] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 369–379, 1999.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. ICC '93*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[4] L. N. Lee, A. R. Hammons Jr., F. W. Sun, and M. Eroz, "Application and standardization of turbo codes in third-generation high-speed wireless data services," *IEEE Trans. Veh. Technol.*, vol. 49, pp. 2198–2207, Nov. 2000.

[5] S. Hong and W. E. Stark, "Power consumption vs. decoding performance for low energy wireless communication system design," in *Proc. ICECS '99*, vol. 3, 1999, pp. 1593–1596.

[6] B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications.* Kluwer Academic Publishers, 2000.

[7] C. Heegard and S. B. Wicker, *Turbo Coding.* Kluwer Academic Publishers, 1999.

[8] P. Sweeney, *Error Control Coding: An Introduction.* Prentice Hall, 1991.

[9] S. Benedetto and G. Montorsi, "Unveiling turbo-codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409–428, 1996.

[10] G. D. Forney Jr., *Concatenated Codes*. Cambridge, MA: MIT Press, 1966.

[11] *Standards for CDMA2000 Spread Spectrum Systems*, EIA/TIA IS-2000.1-6.

[12] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)*, 3GPP TS 25.212 V3.4.0 (2000-09).

[13] M. Eroz and A. R. Hammons Jr., "On the design of prunable interleavers for turbo codes," in *Proc. VTC '99*, Houston, TX, May 16–19, 1999.

[14] L. Lee, M. Eroz, A. R. Hammons Jr., K. Karimullah, and F. W. Sun, "Third generation mobile telephone systems and turbo codes," in *Proc. 3rd Int. Symp. Multi-Dimensional Mobile Communications*, Melno Park, CA, Sept. 21–22, 1998, invited paper.

[15] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[16] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. Telecomm.*, vol. 8, no. 2, Mar.–Apr. 1997.

[17] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[18] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, Mar. 1996.

[19] J. Hagenauer, "Source-controlled channel coding," *IEEE Trans. Commun.*, vol. 43, no. 9, pp. 2449–2457, Sept. 1995.

[20] F. Berens, A. Worm, H. Michel, and N. Wehn, "Implementation aspects of turbo-decoders for future radio applications," in *Proc. IEEE Vihec. Tech. Conf. '99*, vol. 5, Sept. 1999, pp. 2601–2605.

[21] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," in *Proc. Globecom '94*, Dec. 1994, pp. 1298–1303.

[22] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft decision outputs and its applications," in *Proc. Globecom '89*, Nov. 1989, pp. 1680–1686.

[23] P. Jung, "Comparison of turbo-code decoders applied to short frame transmission systems," *IEEE J. Select. Areas Commun.*, vol. 14, no. 3, pp. 530–537, Apr. 1996.

[24] J. Sacha and M. Erwin, "The logarithmic number system for strength reduction in adaptive filtering," in *Proc. ISLPED '98*, Monterey, CA, Aug. 1998, pp. 256–261.

[25] H. Dawid, "Algorithmen und Schaltungsarchitekturen zur Maximum a Posteriori Faltungsdecodeierung," PhD thesis, RWTH Aachen, Shaker Verlag, Aachen, Germany, 1996.

[26] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[27] J. Kaza and C. Chakrabarti, "Design and implementation of low-energy turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 12, no. 9, pp. 968–977, Sept. 2004.

[28] Y. Tong, T.-H. Yeap, and J.-Y. Chouinard, "VHDL implementation of a turbo decoder with Log-MAP-based iterative decoding," *IEEE Trans. Instrum. Meas.*, vol. 53, no. 4, pp. 1268–1278, Aug. 2004.

[29] V. C. Gaudet and P. G. Gulak, "A 13.3-Mb/s 0.35-μm CMOS analog turbo decoder IC with a configurable interleaver," *IEEE J. Solid-State Circuits*, vol. 38, no. 11, pp. 2010–2015, Nov. 2003.

[30] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 LogMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *ISSCC 2003 Dig. Tech. Papers*, vol. 1, 2003, pp. 150–151.

[31] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; UTRA High Speed Downlink Packet Access*, 3GPP TR 25.950 V4.0.0 (2001-03).

[32] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo decoders," JPL TMO Progress Report 42-142, Aug. 15, 2000.

[33] Z. Wang, Z. Chi, and K. K. Parhi, "Area-efficient high-speed decoding schemes for turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 10, no. 6, pp. 902–912, Dec. 2002.

[34] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R.-H. Yan, "A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-μm CMOS," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1555–1564, Nov. 2002.

[35] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, "Architectural strategies for low-power VLSI turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 10, no. 3, pp. 279–285, June 2002.

[36] X.-J. Zeng and Z.-L. Hong, "Design and implementation of a turbo decoder for 3G W-CDMA systems," *IEEE Trans. Consumer Electron.*, vol. 48, no. 2, pp. 284–291, May 2002.

[37] C. Schurgers, F. Catthoor, and M. Engels, "Memory optimization of MAP turbo decoder algorithms," *IEEE Trans. VLSI Syst.*, vol. 9, no. 2, pp. 305–312, Apr. 2001.

[38] Y. Wang, C. Y. Tsui, and R. S. Cheng, "A low power VLSI architecture of SOVA-based turbo-code decoder using scarce state transition scheme," in *Proc. ISCAS 2000*, vol. 1, Geneva, Switzerland, 2000, pp. 283–286.

[39] S. Hong, J. Yi, and W. E. Stark, "VLSI design and implementation of low-complexity adaptive turbo-code encoder and decoder for wireless mobile communication applications," in *Proc. SiPS '98*, 1998, pp. 233–242.

[40] Z. Wang, H. Suzuki, and K. K. Parhi, "VLSI implementation issues of turbo decoder design for wireless applications," in *Proc. SiPS '99*, 1999, pp. 503–512.

[41] ——, "A K=3, 2Mbps low power turbo decoder for 3rd generation W-CDMA systems," in *Proc. CICC 2000*, May 2000, pp. 39–42.

[42] M. Bekooij, J. Dielissen, F. Harmsze, S. Sawitzki, J. Huisken, A. van der Werf, and J. van Meerbergen, "Power-efficient application-specific VLIW processor for turbo decoding," in *ISSCC 2001 Dig. Tech. Papers*, 2001, pp. 180–181.

[43] *PCE03 3GPP Turbo Encoder*, Version 0.3, Small World Communications, Australia, Dec. 19, 2000.

[44] *PCD03 3GPP Turbo Decoder*, Version 0.4, Small World Communications, Australia, Mar. 1, 2001.

[45] *Technical Description of Turbo Product Codes*, Version 4.0, Efficient Channel Coding, Inc., OH, USA, June 1999.

[46] R. Y. Shao, S. Lin, and M. P. C. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117–1120, Aug. 1999.

[47] K. R. Narayanan and G. L. Stüber, "A novel ARQ technique using the turbo coding principle," *IEEE Commun. Lett.*, vol. 1, no. 2, pp. 49–51, Mar. 1997.

[48] F. Gilbert, A. Worm, and N. Wehn, "Low power implementation of a turbo-decoder on programmable architectures," in *Proc. DAC 2001, Asia and South Pacific*, Jan. 30 – Feb. 2, 2001, pp. 400–403.

[49] O. Y.-H. Leung, C.-Y. Tsui, and R. S.-K. Cheng, "Reducing power consumption of turbo-code decoder using adaptive iteration with variable supply voltage," *IEEE Trans. VLSI Syst.*, vol. 9, no. 1, pp. 34–41, Feb. 2001.

[50] A. Worm, P. Hoeher, and N. When, "Turbo-decoding without SNR estimation," *IEEE Commun. Lett.*, vol. 4, no. 6, pp. 193–195, June 2000.

[51] H. J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 468–473, Aug. 1984.

[52] N. Weste and K. Eshragian, *Principles of CMOS VLSI design: A systems perspective.* MA: Addison-Wesley, 1988.

[53] R. K. Watts, Ed., *Submicron integrated circuits.* NY: John Wiley & Sons, 1989.

[54] S. S. Pietrobon, "Implementation and performance of a TURBO/MAP decoder," *Int. J. Satell. Commun.*, 1998.

[55] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," JPL TDA Progress Report 42-124, Feb. 1996.

[56] (2005) The CMC homepage on the Rapid Prototyping Platform. [Online]. Available: http://cmc.ca/prod_serv/des_fab_test/design/rapid_prototyping_system.html

Publications

[1] I. A. Al-Mohandes and M. I. Elmasry, "Iteration reduction of turbo decoders using an efficient stopping/cancellation technique," in *Proc. ISCAS 2002*, May 2002, Scottsdale, AZ, vol. 1, pp. 609–612.

[2] I. A. Al-Mohandes and M. I. Elmasry, "A new efficient dynamic-iterative technique for turbo decoders," in *Proc. MWSCAS 2002*, Aug. 2002, Tulsa, OK, vol. 3, pp. 180–183.

[3] I. A. Al-Mohandes and M. I. Elmasry, "Design of an energy-efficient turbo decoder for 3$^{\text{rd}}$ generation wireless applications," in *Proc. ICM 2003*, Dec. 2003, Cairo, Egypt, pp. 127–130.

[4] I. A. Al-Mohandes and M. I. Elmasry, "A low-power 5 Mb/s turbo decoder for third-generation wireless terminals," in *Proc. CCECE 2004*, May 2004, Niagara Falls, ON, vol. 4, pp. 2387–2390.

[5] I. A. Al-Mohandes and M. I. Elmasry, "Low-energy design of a 3G-compliant turbo decoder," in *Proc. NEWCAS 2004*, June 2004, Montreal, QC, pp. 153–156.

[6] I. A. Al-Mohandes and M. I. Elmasry, "An energy-efficient turbo decoder design for 3G wireless systems," *IEEE Trans. VLSI Syst.*, submitted for publication.