LUND UNIVERSITY

SOVA Based LTE Turbo Decoders

Performance and Architectures

ANG, LAY HONG
LIM, WEE GUAN

Master's Thesis at Ericsson AB
Supervisor: Matthias Kamuf, Ph.D

Lund, September 2009

Department of Electrical and Information Technology
Lund University
P.O. Box 118
SE-221 00 Lund, Sweden

This thesis is set in Computer Modern 10pt
with the LaTeX Documentation System

Printed in Sweden by Tryckeriet E-huset, Lund
September 2009

Abstract

The Max-Log-MAP algorithm is commonly used in a constituent decoder for turbo coding applications. In this thesis, the use of Soft-Output Viterbi Algorithms (SOVAs) as an alternative to Max-Log-MAP for use in the 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) standard is investigated. The decoding performance of the Battail Rule SOVA (BR-SOVA) was found to be comparable to that of the Max-Log-MAP although there is a high price to be paid in terms of hardware complexity. A Simplified Battail Rule SOVA (SB-SOVA) algorithm that is suitable for hardware implementation is proposed, and the decoding performance of SB-SOVA is similar to BR-SOVA. A proposed hybrid-SOVA architecture that combines both Hagenauer Rule SOVA (HR-SOVA) and BR-SOVA to reduce the computational complexity of SB-SOVA is described, and the decoding performance was found to be within 0.1 dB of SB-SOVA.

To improve throughput and reduce the latency, the input data block is divided into windows to allow for parallel processing. The α-stage warm-up method to determine the initial state of a window was found to give better performance as compared to the next iteration initialize (NII) method, with the performance of a windowed decoder performing within 0.1 dB of the ideal Max-Log-MAP.

Hardware architectures for the SB-SOVA and hybrid-SOVA decoders are presented and the memory requirements for SOVA were assessed to be around 10 % of that required for Max-Log-MAP on a per-window basis. The SB-SOVA and hybrid-SOVA decoders consume 87 % and 30 % more hardware resources per window respectively as compared to the Max-Log-MAP decoder. For the same degree of parallelization, the SOVA-based decoders provide 34 % higher throughput than a Max-Log-MAP decoder. The increased throughput of the SOVA architecture may enable the SB-SOVA based decoder to have lower hardware requirements as compared to the Max-Log-MAP as the the degree of parallelization required for a given data rate will be lower.

Contents

Acknowledgments

Abbreviations

Nomenclature

1 Introduction **1**
- 1.1 Turbo Decoding in LTE . . . 2
- 1.2 Algorithms for Decoding Turbo Codes . . . 4
  - 1.2.1 Log Likelihood Ratio . . . 4
  - 1.2.2 MAP Type Algorithms . . . 6
  - 1.2.3 SOVA Algorithm . . . 9

2 Improved SOVA Algorithm **13**
- 2.1 Modifications to SOVA . . . 13
  - 2.1.1 Merging and Updating Depths . . . 13
  - 2.1.2 Reliability Thresholding . . . 14
  - 2.1.3 Scaling of SOVA Outputs . . . 14
  - 2.1.4 Simplification of BR-SOVA Update Rule . . . 15
  - 2.1.5 Reliability Update Methods for SOVA . . . 17
- 2.2 Simulation Results . . . 20
  - 2.2.1 Simulation Environment . . . 20
  - 2.2.2 Optimum Reliability Thresholding and Scaling Factors for SOVA . . . 20
  - 2.2.3 Effects of Simplified BR-SOVA Update Rule . . . 26
  - 2.2.4 Comparison of Reliability Update Methods for SOVA . . . 27
  - 2.2.5 Results for Hardware Reliability Update Method . . . 28

3 SISO Decoder Hardware Architectures **33**
- 3.1 Hardware Architecture of Max-Log-MAP . . . 33
  - 3.1.1 Resource Utilization . . . 35
  - 3.1.2 Memory Requirements . . . 36
  - 3.1.3 Latency . . . 36

3.2 Hardware Architecture for HR-SOVA . . . 38
3.2.1 Trellis Stage . . . 38
3.2.2 Merge Stage . . . 41
3.2.3 Decode Stage . . . 41
3.2.4 Example of Hardware Architecture . . . 42
3.2.5 Latency and Throughput . . . 44
3.3 Simplified BR-SOVA Architecture . . . 46
3.3.1 Resource Utilization . . . 51
3.3.2 Memory and Register Utilization . . . 52
3.3.3 Latency and Throughput . . . 53
3.4 Hybrid-SOVA Architecture . . . 54
3.4.1 Resource Utilization . . . 56
3.4.2 Memory and Register Utilization . . . 56
3.4.3 Latency and Throughput . . . 57
3.5 Improved BPCU for SB-SOVA . . . 58
3.5.1 Resource Utilization . . . 60
3.5.2 Memory requirements . . . 62
3.6 Considerations for Parallel Windows . . . 62
3.6.1 Memory Organization for Multiple Windows . . . 64
3.6.2 Estimation of Window Initial State . . . 64
3.6.3 Inter-Bank Memory Access . . . 68
3.7 Comparison of SOVA and Max-Log-MAP . . . 68
3.7.1 Resource Utilization . . . 68
3.7.2 Memory Requirement . . . 69
3.7.3 Data Throughput . . . 69
3.7.4 Optimal SOVA Decoder . . . 70

4 Conclusions **77**

Bibliography **79**

Acknowledgments

We would first like to thank our supervisor Matthias Kamuf for providing all his help, assistance and guidance throughout the entire thesis work. We really appreciate the freedom you have given us to do carry out our own research and at the same time being there giving us advice and pointers all along the way.

Thanks go to all our friends and colleagues at Ericsson Research who have helped to make our stay fruitful and interesting. Special thanks go to Fredrik Nordström for all the delightful chats and tips, and of course to Jim Svensson for his synthesis tools, scripts and VNC/SVN server that have helped the work proceed much more smoothly and efficiently.

We are also very grateful to Joachim Rodrigues at LTH for giving us the opportunity to take on this challenging project at Ericsson Research. Our appreciation goes to all our teachers and friends at LTH who have made our stay a very enjoyable and fruitful one. Lastly, we would like to thank our classmates Ruiyi Zhang and Jiangfeng Cai for all the fun times that we had working together at Ericsson, and for the great dinner.

Abbreviations

3GPP	3rd Generation Partnership Project
ACS	Add-Compare-Select
AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Rate
BLER	Block Error Rate
BR-SOVA	Battail Rule SOVA
CRC	Cyclic Redundancy Check
FIFO	First In First Out
HR-SOVA	Hagenauer Rule SOVA
LIFO	Last In First Out
LTE	Long Term Evolution
LLR	Log-Likelihood Ratio
MAP	Maximum-a-Posteriori
ML	Maximum Likelihood
MSOVA	Modified Soft-Output Viterbi Algorithm
PCU	Path Comparison Unit
RE	Register Exchange
RSC	Recursive Systematic Convolutional
SB-SOVA	Simplified Battail Rule SOVA

SISO	Soft-Input/Soft-Output
SMU	Survivor Memory Unit
SNR	Signal-to-Noise Ratio
SOVA	Soft-Output Viterbi Algorithm
VA	Viterbi Algorithm

Nomenclature

$\hat{u}_k$	Estimated hard decision for bit u_k
ν	Memory depth of convolutional encoder
L	Merging depth for Viterbi Algorithm
$L(\hat{u}_k)$	A-posteriori value for bit u_k
$L(u_k)$	A-priori value for bit u_k
$L_c \cdot y_k$	Received channel values
$L_e(\hat{u}_k)$	Extrinsic value for bit u_k
M	Number of windows in a windowed decoder
N	Number of trellis states
n	Number of coded bits per information bit
S	Sequence of state transitions through a trellis
T_{crit}	Combinatorial delay of the critical path
U	Update depth for Soft-Output Viterbi Algorithm
u_k	Data bit to be transmitted at time $t = k$
W	Window size
Y	Received sequence of K_i symbols, with $Y = \{y_{k,1}, y_{k,2}, \ldots, y_{k,n}\}_{k=1}^{K_i}$
$y_{k,j}$	jth received bit for symbol at time $t = k$
$\Delta_{k,s}$	Metric difference at time $t = k$ for state s

Chapter 1

Introduction

In this current age where high speed portable wireless devices such as mobile telephones and wireless networks are commonplace, the market demands for smaller devices with longer battery life are ever increasing. This leads to stringent requirements for wireless terminals that can provide higher throughput, having smaller chip size, and yet consume as little power as possible. To meet this goal, an entire receiver chain will need to be designed with the above requirements in mind. This thesis looks in detail at the turbo decoder, which forms a part of the outer receiver chain.

The channel encoding used in the 3rd Generation Partnership Project (3GPP) E-UTRA Release 8 "Long Term Evolution (LTE)" standard [1] is a turbo code, which is computationally intensive to decode, and typically implemented as a hardware accelerator module within the receiver. The Maximum-a-Posteriori (MAP)-type algorithms are commonly used as Soft-Input/Soft-Output (SISO) decoders for turbo decoding. The Log-MAP and Max-Log-MAP based decoders do have some drawbacks, namely in that they require relatively large amounts of on-chip memory for the storage of intermediate values, and high latency due to the backward recursions that need the last symbols in the block to be received before decoding can begin [2].

An alternative decoder to the MAP-type decoders is one that is based on the Soft-Output Viterbi Algorithm (SOVA). There are existing hardware implementations of Hagenauer Rule SOVA (HR-SOVA) [3][4][5], but the performance of HR-SOVA based decoders in simulations is typically about 0.7 dB worse [6] than Max-Log-MAP based ones. Another updating rule proposed by Battail [7], known as Battail Rule SOVA (BR-SOVA) gives better decoding performance than HR-SOVA, but there is no known hardware implementation of BR-SOVA in the literature.

This thesis aims to investigate the performance of a SOVA based turbo decoder, whilst using a Max-Log-MAP decoder as a baseline for comparison. A hardware architecture for a parallel windowed SOVA based turbo decoder is developed. Fi-

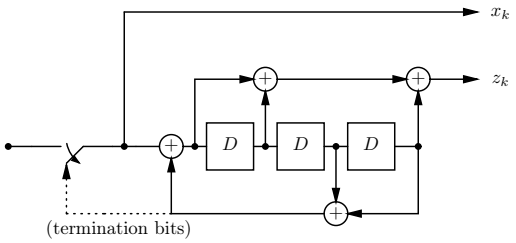


Figure 1.1: Block diagram of RSC encoder

nally, the proposed decoder is compared with a Max-Log-MAP based turbo decoder with respect to the decoding performance, hardware resource utilization, memory requirements and latency/throughput.

This chapter gives a brief introduction into the various topics covered within the thesis, so as to give context to the discussions in the chapters that follow. The channel coding algorithms used in the LTE standard are described briefly, followed by descriptions of the MAP and SOVA algorithms that can be applied to turbo decoding.

1.1 Turbo Decoding in LTE

The iterative turbo encoder used in LTE for data transport is made up of two 8-state, rate 1/2 Recursive Systematic Convolutional (RSC) constituent encoders that are connected in a parallel concatenated convolutional coding scheme [8].

The transfer function for a constituent convolutional encoder is given as

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right]$$

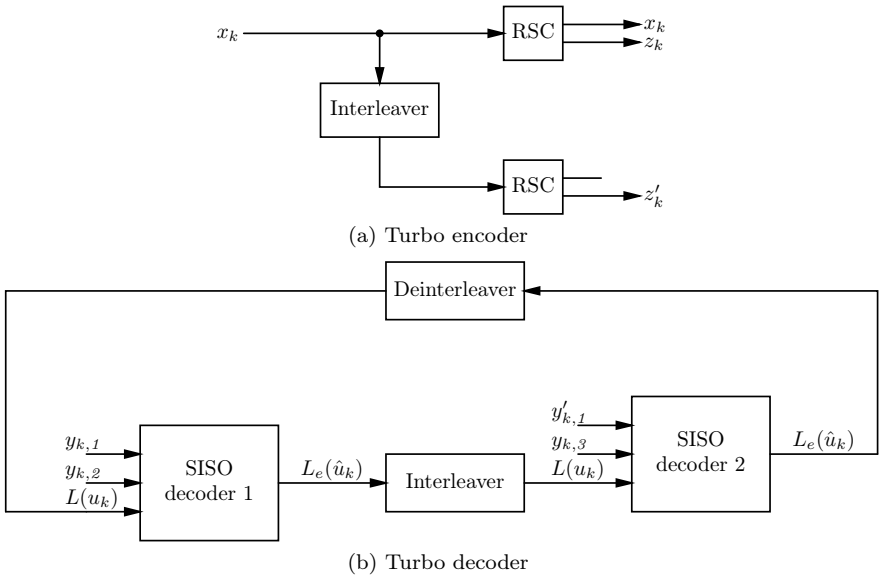
where the feed-forward and feedback generator polynomials are given as

$$g_0(D) = 1 + D^2 + D^3$$
$$g_1(D) = 1 + D + D^3$$

A block diagram of the resulting constituent encoder is shown in Figure 1.1. Each constituent encoder generates a rate 1/2 code, and it follows that the output of the turbo encoder (before puncturing) gives a code with a rate $R = 1/3$.

The turbo decoder in principle comprises of two SISO decoders that are separated by interleavers and deinterleavers. Block diagrams of the encoder and decoder are shown in Figure 1.2.

The format of the turbo code follows the 3GPP standard as described in [1]. It can be briefly described as follows. Consider an L-bit data block $\{c_0, c_1, \ldots, c_{L-1}\}$



(a) Turbo encoder

(b) Turbo decoder

Figure 1.2: Turbo encoder and decoder

that is checksummed by a 24-bit Cyclic Redundancy Check (CRC) with parity bits $\{d_0, d_1, \ldots, d_{23}\}$. The information bits to the turbo encoder will have a block length of $K_i = L + 24$ with the following sequence.

$$\boldsymbol{x} = \{c_0, c_1, \ldots, c_{L-1}, d_0, d_1, \ldots, d_{23}\}$$

The encoded output bits of the turbo encoder will then be

$$\{x_0, z_0, z'_0,\ x_1, z_1, z'_1, \ldots,\ x_{k-1}, z_{k-1}, z'_{k-1}\}$$

where z_i is the output of the first constituent encoder and z'_i is the output of the second constituent encoder, with the interleaved x_i as the input. z_i and z'_i are the *parity bits* of the first and second RSC encoders respectively.

The trellis of the turbo encoder will be terminated by feeding back the remaining bits in the shift register after all the data bits (including CRC) have been encoded using the dotted lines shown in Figure 1.1. The remaining 12 tail bits that are transmitted to terminate the trellis are given as

$$\{x_k,\ z_k,\ x_{k+1},\ z_{k+1},\ x_{k+2},\ z_{k+2},\ x'_k,\ z'_k,\ x'_{k+1},\ z'_{k+1},\ x'_{k+2},\ z'_{k+2}\}$$

which gives a total encoded output length of $3(K_i + 4)$ bit.

A block diagram of the turbo decoder is shown in Figure 1.2b, where it can be seen that two SISO decoders are connected together through an interleaver and a

deinterleaver. In the first iteration, there is no a-priori information, so the a-priori input $L(u_k)$ into SISO decoder 1 is zero. The extrinsic output $L_e(\hat{u}_k)$ of SISO decoder 1 is used (after interleaving) by SISO decoder 2 as a-priori input. The extrinsic output of SISO decoder 2 will once again (after deinterleaving) be used as the a-priori input of SISO decoder 1. This feedback of extrinsic output to SISO decoder 1 starts the next iteration of the turbo decoding process. To obtain the final output, the deinterleaved extrinsic output of the SISO decoders will need to be combined with the a-priori input as described in Section 1.2.1.

The LTE turbo code internal interleaver, as described in [1], is designed to shuffle the data between the two encoders/decoders. The interleaver helps spread out the burst errors and thus improves the performance of the decoder.

The algorithms that can be used to perform turbo decoding are described in greater detail in Section 1.2.

1.2 Algorithms for Decoding Turbo Codes

This section presents two classes of decoders that can be used for turbo decoding, namely the MAP-type algorithms and the SOVA. Section 1.2.1 describes the Log-Likelihood Ratio (LLR) that are used to exchange information between the two constituent decoders. The details of MAP algorithm and SOVA algorithm are covered in Sections 1.2.2 and 1.2.3 respectively.

1.2.1 Log Likelihood Ratio

To ease the complexity of performing multiplications, operations are typically performed in the log-domain, and the Log-Likelihood Ratio (LLR) of a binary random variable u_k, $L(u_k)$ is defined as

$$L(u_k) = \log \frac{\mathrm{P}(u_k = +1)}{\mathrm{P}(u_k = -1)} \tag{1.1}$$

The LLR of each bit is passed between the two constituent decoders.

Since u_k is in GF(2) with the elements $\{+1, -1\}$,

$$\mathrm{P}(u_k = +1) = 1 - \mathrm{P}(u_k = -1)$$

and

$$L(u_k) = \log \frac{\mathrm{P}(u_k = +1)}{1 - \mathrm{P}(u_k = +1)}$$

Conditioning the random variable u_k on a different random variable y_k, the conditioned log-likelihood ratio $L(u_k|y_k)$ is given by

$$\begin{aligned} L(u_k|y_k) &= \log \frac{\mathrm{P}(u_k = +1|y_k)}{\mathrm{P}(u_k = -1|y_k)} \\ &= \log \frac{\mathrm{p}(y_k|u_k = +1) \cdot \mathrm{P}(u_k = +1)}{\mathrm{p}(y_k|u_k = -1) \cdot \mathrm{P}(u_k = -1)} \\ &= L(y_k|u_k) + L(u_k) \end{aligned} \tag{1.2}$$

After transmitting over a channel with a fading factor a and Additive White Gaussian Noise (AWGN),

$$\begin{aligned} L(u_k|y_k) &= \log \frac{\mathrm{p}(y_k|u_k = +1) \cdot \mathrm{P}(u_k = +1)}{\mathrm{p}(y_k|u_k = -1) \cdot \mathrm{P}(u_k = -1)} \\ &= \log \frac{\exp(-E_s/N_0(y_k - a)^2)}{\exp(-E_s/N_0(y_k + a)^2)} + \log \frac{\mathrm{P}(u_k = +1)}{\mathrm{P}(u_k = -1)} \\ &= 4 \cdot a \cdot \frac{E_s}{N_0} \cdot y_k + L(u_k) \\ &= L_c \cdot y_k + L(u_k) \end{aligned} \tag{1.3}$$

where $L_c = 4 \cdot a \cdot (E_s/N_0)$ is known as the reliability value of the channel. For fading channels, a denotes the fading amplitude, but for Gaussian channels, $a = 1$.

The detailed derivation of the LLR values output by the constituent decoders is covered in [9]. In a similar fashion, the conditional probability can be derived, and for brevity it is stated here as follows

$$\begin{aligned} \mathrm{p}(y_k|u_k = \pm 1) &= \left(\frac{\mathrm{P}(y_k) \cdot (1 + \exp(-L(u_k)) \cdot \exp(-L_c \cdot y_k/2)}{1 + \exp(-(L(u_k) + L_c \cdot y_k))} \right) \cdot \exp(u_k \cdot L_c \cdot y_k/2) \\ &= B_k \cdot \exp(u_k \cdot L_c \cdot y_k/2) \end{aligned} \tag{1.4}$$

where

$$B_k = \frac{\mathrm{P}(y_k) \cdot (1 + \exp(-L(u_k)) \cdot \exp(-L_c \cdot y_k/2)}{1 + \exp(-(L(u_k) + L_c \cdot y_k))}$$

The output of the decoder is the logarithm of the ratio of the bit probability being "+1" or "−1" for a given observation of y, or

$$L(\hat{u}_k) = L(u_k|y) = \log \frac{\mathrm{P}(u = +1|y)}{\mathrm{P}(u = -1|y)} \tag{1.5}$$

A block diagram of a SISO decoder is shown in Figure 1.3. The decoder uses the a-priori values $L(u_k)$ for all information bits u_k and the received channel values

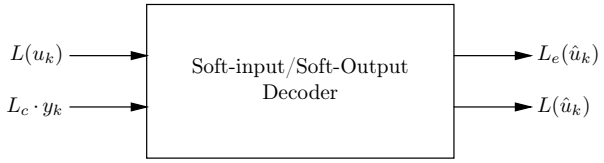


Figure 1.3: Block diagram of SISO decoder

$L_c \cdot y_k$ for the coded bits as inputs and returns the soft output bits of the decoder, $L(\hat{u}_k)$ and extrinsic information $L_e(\hat{u}_k)$. For systematic codes, the resulting soft output information bit is given by

$$L(\hat{u}_k) = L_c \cdot y_k + L(u_k) + L_e(\hat{u}_k) \tag{1.6}$$

As shown in Figure 1.2b, the extrinsic output from the first constituent decoder, $L_e(\hat{u}_k)$, is passed via an interleaver/deinterleaver to the next constituent decoder as the a-priori input. The a-priori input to the first decoder in the first iteration is typically set to zero, as there is usually no a-priori information at this stage. By increasing the number of iterations, better decoding performance can be obtained at the expense of latency and computational cost.

1.2.2 MAP Type Algorithms

The traditional approach to decoding RSC codes is to use the "symbol-by-symbol" Maximum-a-Posteriori (MAP) decoding that minimizes the Bit Error Rate (BER) of the decoded bits. MAP decoding is done using the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [10] which is optimal for estimating the outputs of a Markov process.

The MAP algorithm is not practical for implementation because of the numerical representation of probabilities, non-linear functions and the large numbers of multiplication and division operations. Log-MAP is equivalent to true MAP, but because it operates in the logarithmic domain, it avoids the implementation pitfalls associated with MAP. The Max-Log-MAP uses an approximation to simplify the Log-MAP further at the expense of some performance degradation. The MAP, Log-MAP and Max-Log-MAP algorithms will be described in the sections that follow.

MAP

Let $y_{j<k}$ denote the sequence of received symbols y_j from the start of the trellis up to and including $t = k - 1$, and $y_{j>k}$ be the corresponding received sequence from $t = k + 1$ to the end of the trellis. Using the LLR defined in (1.5),

$$L(\hat{u}_k) = \log \frac{\mathrm{P}(u_k = +1|y)}{\mathrm{P}(u_k = -1|y)} = \log \frac{\sum_{u_k=+1}^{(s',s)} \mathrm{P}(s', s, y)}{\sum_{u_k=-1}^{(s',s)} \mathrm{P}(s', s, y)} \tag{1.7}$$

where

$$\begin{aligned} \mathrm{P}(s', s, y) &= \mathrm{P}(s', y_{j<k}) \cdot \mathrm{P}(s, y_k|s') \cdot \mathrm{P}(y_{j>k}|s) \\ &= \underbrace{P(s', y_{j<k})}_{\alpha_{k-1}(s')} \cdot \underbrace{\mathrm{P}(s|s')}_{\gamma_k(s',s)} \cdot \underbrace{\mathrm{P}(y_{j>k}|s)}_{\beta_k(s)} \end{aligned} \tag{1.8}$$

therefore

$$L(\hat{u}_k) = \log \frac{\sum_{u_k=+1}^{(s',s)} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)} \tag{1.9}$$

The BCJR algorithm uses both forward and backward recursions through the trellis. The branch transition probability from sequence s' to sequence s is given by the $\gamma_k(s', s)$ term. The forward recursion is given by the $\alpha_k(s)$ term and depends on the previous bits in the trellis. The backward recursion is given by the $\beta_{k-1}(s')$ term, which as expected depends on the next/future bits in the trellis. Mathematically, these three terms can be written as follows

$$\begin{aligned} \gamma_k(s', s) &= \mathrm{P}(s|s') \cdot \mathrm{p}(y_k|s', s) \\ &= \mathrm{P}(y_k|u_k) \cdot \mathrm{P}(u_k) \end{aligned} \tag{1.10}$$

$$\alpha_k(s) = \sum_{(s',s)} \gamma_k(s', s) \cdot \alpha_{k-1}(s') \tag{1.11}$$

$$\beta_{k-1}(s') = \sum_{(s',s)} \gamma_k(s', s) \cdot \beta_k(s) \tag{1.12}$$

As the index pair of previous and current state (s', s) determines the information bit u_k and the parity bits $x_{k,v}$ for $v = 2, \ldots, n$ in a systematic code, the probability of the received symbol given a specific data bit can be written as follows

$$\mathrm{P}(y_k|u_k) = \mathrm{P}(y_{k,1}|u_k) \cdot \prod_{v=2}^{n} \mathrm{P}(y_{k,v}|x_{k,v})$$

and using (1.4),

$$\begin{aligned} \mathrm{P}(y_k|u_k) &= B_k \cdot \exp\left(\frac{1}{2}L_c \cdot y_{k,1} \cdot u_k\right) \cdot \left(\prod_{v=2}^{n} \mathrm{P}(y_{k,v}|x_{k,v})\right) \\ &= B_k \cdot \exp\left(\frac{1}{2}L_c \cdot y_{k,1} \cdot u_k + \frac{1}{2}\sum_{v=2}^{n} \mathrm{P}(y_{k,v}|x_{k,v})\right) \end{aligned} \tag{1.13}$$

Log-MAP

The Log-MAP algorithm is a transformation of the MAP algorithm to the logarithmic domain, and in doing so, multiplications are converted into additions. Since it is just a change in domain, the Log-MAP algorithm gives equivalent performance to the MAP algorithm.

For brevity, the LLR expression for Log-MAP is stated as follows,

$$\begin{aligned} L(\hat{u}_k) &= \log \frac{\sum_{u_k=+1}^{(s',s)} \exp\left(\alpha_{k-1}^{LM}(s')\right) \cdot \exp\left(\gamma_k^{LM}(s',s)\right) \cdot \exp\left(\beta_k^{LM}(s)\right)}{\sum_{u_k=-1}^{(s',s)} \exp\left(\alpha_{k-1}^{LM}(s')\right) \cdot \exp\left(\gamma_k^{LM}(s',s)\right) \cdot \exp\left(\beta_k^{LM}(s)\right)} \\ &= \log\left(\sum_{u_k=+1}^{(s',s)} \exp\left(\alpha_{k-1}^{LM}(s') + \gamma_k^{LM}(s',s) + \beta_k^{LM}(s)\right)\right) \\ &\quad - \log\left(\sum_{u_k=-1}^{(s',s)} \exp\left(\alpha_{k-1}^{LM}(s') + \gamma_k^{LM}(s',s) + \beta_k^{LM}(s)\right)\right) \end{aligned} \tag{1.14}$$

where $\alpha_{k-1}^{LM}(s')$, $\beta_k^{LM}(s)$ and $\gamma_k^{LM}(s',s)$ are the forward recursion, backward recursion, and transitional probability terms respectively in the logarithmic domain.

Max-Log-MAP

Despite having reduced the complexity of the MAP algorithm by moving to the logarithmic domain, the $\log\left(\sum_{i=1}^{k}(\exp(x_i))\right)$ function in the Log-MAP remains computationally intensive to implement. The Max-Log-MAP algorithm provides reduces the complexity of the Log-MAP algorithm drastically by performing the following approximation

$$\begin{aligned} \log\left(\sum_{i=1}^{k}(\exp\left(x_i\right))\right) &= \max\left(\exp(x_i)\right) + \log\left(\sum_{i=1}^{k}\left(\exp(x_i - \max(x_i))\right)\right) \\ &\approx \max\left(\exp(x_i)\right) \end{aligned} \tag{1.15}$$

By applying this approximation on the Log-MAP branch transition probabilities, Log-MAP forward and backward recursion equations, the following equations

are obtained

$$\alpha_k^{MLM}(s) = \max\left([\gamma_{u_k=+1}^{LM}(s',s) + \alpha_{k-1}^{LM}(s')], [\gamma_{u_k=-1}^{LM}(s',s) + \alpha_{k-1}^{LM}(s')]\right) \quad (1.16)$$

$$\beta_k^{MLM}(s') = \max\left([\gamma_{u_k=+1}^{LM}(s',s) + \beta_k^{LM}(s)], [\gamma_{u_k=-1}^{LM}(s',s) + \beta_k^{LM}(s)]\right) \quad (1.17)$$

and the resulting LLR for the Max-Log-MAP algorithm can be stated as

$$\begin{aligned} L(\hat{u}_k) = &\max_{s',s\,(u_k=+1)} \big([\gamma_k^{LM}(s',s) + \alpha_{k-1}^{MLM}(s) + \beta_k^{MLM}(s')], \\ &\qquad [\gamma_k^{LM}(s',s) + \alpha_k^{MLM}(s) + \beta_k^{MLM}(s')]\big) \\ &- \max_{s',s\,(u_k=-1)} \big([\gamma_k^{LM}(s',s) + \alpha_{k-1}^{MLM}(s) + \beta_k^{MLM}(s')], \\ &\qquad [\gamma_k^{LM}(s',s) + \alpha_k^{MLM}(s) + \beta_k^{MLM}(s')]\big) \end{aligned} \quad (1.18)$$

The approximation in (1.15) degrades the performance of the Max-Log-MAP algorithm as compared with the Log-MAP. A correction factor is typically added into the Max-Log-MAP algorithm to compensate for the degradation, and is used in practice to implement Log-MAP decoders in hardware [2].

1.2.3 SOVA Algorithm

The traditional approach to decoding convolutional codes is by using the Viterbi Algorithm (VA) that provides a Maximum Likelihood (ML) sequence [11]. The VA in the most general form is a maximum a-posteriori probability sequence estimator that finds the maximum likelihood path through the trellis diagram given a specific received sequence. Mathematically, the VA finds the trellis path that maximizes the a-posteriori probability $\mathrm{P}(\boldsymbol{S}|\boldsymbol{Y})$.

The Soft-Output Viterbi Algorithm (SOVA) decoder outputs soft values that makes it suitable for use in turbo decoders. The modifications to enable the traditional hard-decision VA to give soft-output values was proposed in [12]. The description of SOVA that follows adopts the notation used in [13].

Assuming that the state sequence $\boldsymbol{S}$ is a Markov sequence and since the received sequence $\boldsymbol{Y}$ is independent of the trellis path selection $\boldsymbol{S}$, the VA maximizes

$$\mathrm{p}(\boldsymbol{S_k}, \boldsymbol{Y_k}) = \mathrm{p}(\boldsymbol{S_{k-1}}, \boldsymbol{Y_{k-1}})\,\mathrm{P}(u_k)\,\mathrm{p}(y_k|s',s) \quad (1.19)$$

where $\boldsymbol{S_k} = \{s_1, s_2, \ldots, s_k\}$, $\boldsymbol{Y_k} = \{y_1, y_2, \ldots, y_k\}$, $s' = s_{k-1}$, $s = s_k$ and u_k is the source bit that corresponds to a state transition $s' \rightarrow s$ in path $\boldsymbol{S_k}$.

Since the path metric $M_k(S_k)$ that is associated with the path $\boldsymbol{S_k}$ in the trellis is defined as

$$M_k(S_k) = \log\left(\mathrm{p}(\boldsymbol{S_k}, \boldsymbol{Y_k})\right)$$

we obtain the following equation after substituting into (1.19)

$$M_k(S_k) = M_{k-1}\left(S_k\right) + \log \mathrm{P}(u_k) + \log(\mathrm{p}(y_k|s',s)) \quad (1.20)$$

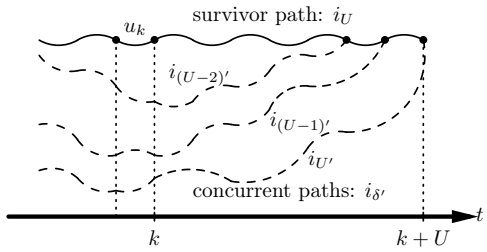


Figure 1.4: Trellis diagram showing survivor and discarded concurrent paths

A forward recursion similar to that in the Log-MAP algorithm (1.11) is used to compute the path metrics. The metric of the ith path at $t = k$ can be expressed recursively [9][13] as

$$M_k(S_k) = M_{k-1}(S_{k-1}) + \frac{1}{2}L(u_k) \cdot u_k + \frac{1}{2}\sum_{v=1}^{n} L_c \cdot y_{k,v} \cdot x_{k,v} \tag{1.21}$$

$$\begin{aligned} M_k(S_k) = M_{k-1}(S_{k-1}) &+ \frac{1}{2}L(u_k) \cdot u_k + \frac{1}{2}L_c \cdot y_{k,1} \cdot u_k \\ &+ \frac{1}{2}\sum_{v=2}^{n} L_c \cdot y_{k,v} \cdot x_{k,v} \end{aligned} \tag{1.22}$$

where (1.22) is the special case of (1.21) for systematic codes with $y_{k,1}$ denoting the received systematic bit. It is useful to note that both (1.21) and (1.22) only hold for convolutional codes generated by feedback encoders, and not for codes generated by feed-forward encoders.

The desired soft output bit is decided by the VA after a merging delay of U bit. At $t = k + U$, the VA has selected the survivor path i_U that terminates at the ML state as shown in Figure 1.4. The other (concurrent) path terminating at the ML state $i_{U'}$ is discarded. Since the path i_U decides the soft output of $\hat{u}_k$ at $t = k$, there are a total of $U + 1$ discarded non-surviving paths denoted by $i_{0'}, i_{1'}, \ldots, i_{U'}$, one for each state in path i_U between $t = k$ and $t = k + U$. The metric difference for a state on the path along i_U is defined as

$$\Delta_{k+l} = M_{k+l}(s_{k+l}) - M_{k+l}(s'_{k+l}) \tag{1.23}$$

The probability, P(*correct*) of a correct path decision of the survivor at $t = k+l$ given that $y_{j \leq k+l}$ can be simplified as [13]

$$\mathrm{P}(\textit{correct}) = \frac{\exp(\Delta_{k+l})}{1 + \exp(\Delta_{k+l})} \tag{1.24}$$

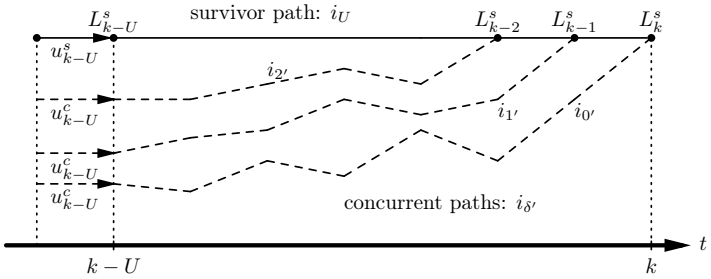


Figure 1.5: Trellis diagram for HR-SOVA

This leads to the LLR of the binary path decision to be

$$\log \frac{\mathrm{P}(\mathit{correct})}{1-\mathrm{P}(\mathit{correct})} = \Delta_{k+l} \tag{1.25}$$

The resulting soft output of the SOVA is the decision bit $\hat{u}_k$ multiplied by the reliability values of all the errors, which can be approximated using

$$L(\hat{u}_k) \approx \hat{u}_k \cdot \min_{l=0,\ldots,U} \Delta_{k+l} \tag{1.26}$$

where the minimum in (1.26) is taken over all the non-surviving paths that would have led to a different decision bit $\hat{u}_k$. Equation 1.26 shows that the same hard decisions $\hat{u}_k$ that are obtained in classical VA, together with the updates considering the non-surviving paths are used to obtain the LLR of each bit.

To make (1.26) easier to implement in practice, update rules are used to modify the metric differences during decoding. The update rule given in (1.26) is often referred to as the Hagenauer rule [12], or HR-SOVA, and can be written mathematically as follows:

$$L^s_{k-U} \leftarrow \min\left[L^s_k, L^s_{k-U}\right] \quad \text{when } u^s_{k-U} \neq u^c_{k-U} \tag{1.27}$$

The updating in (1.27) is applied for U time steps between $t = k$ and $t = k-U+1$ and thus it can be seen that each reliability value, L^s_{k-U} can be updated up to U times. As shown in Figure 1.5, the Hagenauer rule considers the survivor path, as well as up to U concurrent paths to the survivor path, using the cases when the concurrent path will give a different decision bit ($\hat{u}_{k-U}$) to update the reliability values, thus re-stating (1.26).

Another update rule in the literature that was described by Battail [7], often referred to as the Battail updating rule or BR-SOVA is given as:

$$L^s_{k-U} \leftarrow \min\left[L^s_k, L^s_{k-U}\right] \qquad \text{when } u^s_{k-U} \neq u^c_{k-U} \tag{1.28a}$$

$$L^s_{k-U} \leftarrow \min\left[L^s_k + L^c_{k-U}, L^s_{k-U}\right] \qquad \text{when } u^s_{k-U} = u^c_{k-U} \tag{1.28b}$$

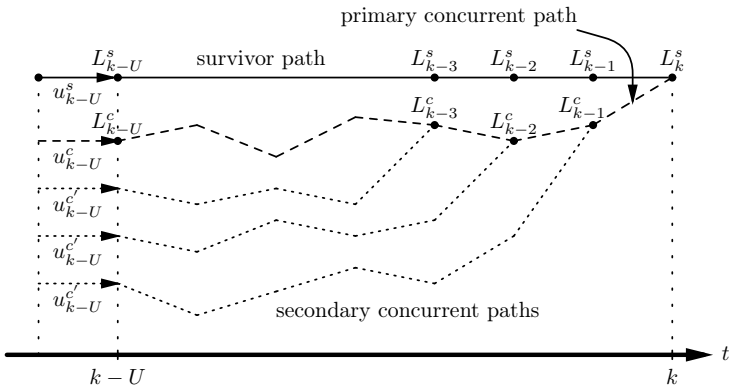


Figure 1.6: Trellis diagram for BR-SOVA showing one primary concurrent path

The Battail rule is not documented in sufficient detail in the open literature other than the two updating rules in (1.28), and so an interpretation of BR-SOVA is used in the analysis that follows. In addition to the U concurrent paths to the survivor path considered by the Hagenauer rule, the Battail rule considers the U secondary concurrent paths to the main (primary) concurrent path. Figure 1.6 shows one primary concurrent concurrent path and the resulting secondary concurrent paths. The secondary concurrent paths to the main concurrent path are traced back to $t = k - U$, and the decision bit of the main concurrent path (u^c_{k-U}) is compared with the decision bit of the secondary concurrent path ($u^{c'}_{k-U}$) and the HR-SOVA update rules in (1.27) are used to update L^c_{k-U}. Finally, the reliability bits of the survivor path (u^s_{k-U}) and main concurrent path (u^c_{k-U}) are compared, and the BR-SOVA updating rules in (1.28) are applied to update L^s_{k-U}.

The updates in the previous paragraph will be repeated for all U primary concurrent paths and thus up to U updates can be applied to every L^s_{k-U}. BR-SOVA thus provides for possibly the same number of reliability value updates as HR-SOVA.

Chapter 2

Improved SOVA Algorithm

This chapter describes the modifications that are made to the traditional Hagenauer Rule SOVA (HR-SOVA) and Battail Rule SOVA (BR-SOVA) algorithms to improve the performance of the SOVA algorithms. The modifications to the SOVA algorithms are described in Section 2.1. The results and analysis from computer simulations are presented in Section 2.2.

2.1 Modifications to SOVA

Based on the HR-SOVA as presented in [12] and [13], there have been various modification techniques presented to improve its performance. The algorithmic improvements to SOVA that were investigated can be divided into three main categories, namely by varying the merging and update depths, scaling and thresholding intermediate values, and modifying the reliability update values. These three categories are described in Sections 2.1.1, 2.1.2 and 2.1.3. With hardware architecture in mind, Sections 2.1.4 and 2.1.5 describe the various considerations and modifications made to the SOVA algorithm in order to improve hardware performance.

2.1.1 Merging and Updating Depths

In a practical decoder, it is not possible to perform VA over the entire block of channel data due to the excessive latency and storage requirements. Instead, sliding windows of merging depth L and update depth U are used to limit the traceback and decoding depths. One possible modification is to vary L and U parameters for the SOVA algorithm. Increasing L will increase the likelihood for a merged path while using a larger U increases the number of updates for the reliability value. However, it is clear that increasing these two values will increase the memory requirements and latency.

The simulation results for determining the optimal values for L and U are presented in Section 2.2.2. Hardware considerations such as performance, latency

and hardware requirements are also discussed in Section 2.2.2.

2.1.2 Reliability Thresholding

As described in [6], the output of the SOVA decoder tends to be over-estimated due to a correlation between the extrinsic output and the intrinsic input into the decoder. Moreover, since SOVA only considers two paths, it is possible that the output will be overly optimistic when the closest "true" competitor to the ML path is eliminated and thus the resulting metric difference between the surviving ML path and its concurrent path is larger than it should be. This occurrence will result in an over-estimation of the final reliability value $L(\hat{u}_k)$, which may result in errors in the output of the decoder. In order to overcome this issue, several remedies have been proposed in various papers. In [7], the authors proposed to apply a thresholding limit Δ_{TH} on the metric difference value $\Delta_{k,s}$ by applying the following rule

$$\Delta_{k,s} \leftarrow \Delta_{TH} \quad \text{if } \Delta_{k,s} > \Delta_{TH} \tag{2.1}$$

The value of Δ_{TH} needs to be selected with caution. When using Δ_{TH} that is too low, the resulting output may be over-compensated, resulting in an overly pessimistic reliability output which may be significant enough to adversely affect the decoding result in the next iteration. When Δ_{TH} is too large, the thresholding does not take effect, and does not provide the necessary compensation.

According to [7], the appropriate value for Δ_{TH} is dependent on the channel quality and the optimum value is usually determined by simulation. For the case when the SOVA decoder is implemented in hardware, quantization effects will also need to be taken into consideration. To simplify hardware implementation, the value of Δ_{TH} can be selected to be a power of 2. The simulation results for the reliability thresholding are presented and discussed in section 2.2.

2.1.3 Scaling of SOVA Outputs

An alternative approach to control the extrinsic reliability values is to apply attenuation factors on two parameters, namely the immediate output of the SOVA decoder, $L(\hat{u}_k)$, and the input to the next decoder, $L_e(\hat{u}_k)$ as described in [14] and [15]. To re-cap from section 1.2.3, the output of the SOVA decoder and the input to the next SOVA decoder are related by

$$\begin{aligned} L_e(\hat{u}_k) &= L(\hat{u}_k) - \big(L(u_k) + L_c \cdot y_k\big) \\ &= L(\hat{u}_k) - L_i(u_k) \end{aligned} \tag{2.2}$$

where $L_i(u_k)$ represents the intrinsic input information provided to the SOVA decoder.

The attenuation of the output reliability values is applied by scaling them with a pair of parameters (c, d) as shown in Figure 2.1. Mathematically, the scaling can

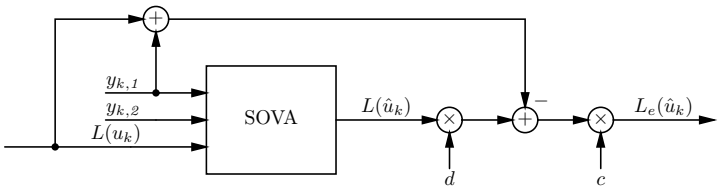


Figure 2.1: Modified SOVA with scaling factors c and d

Table 2.1: Efficient hardware scaling values

Scale Value	Binary Representation	Number of Bits
0.25	0.01b	3 bit
0.5	0.1b	2 bit
0.625	0.101b	4 bit
0.75	0.11b	3 bit
0.875	0.111b	4 bit

be expressed as follows

$$L_e(\hat{u}_k) = c\big(d \cdot L(\hat{u}_k) - L_i(u_k)\big) \tag{2.3}$$

As the factors c and d are attenuation factors, they obey the following condition.

$$0 < c, d \leq 1 \tag{2.4}$$

The detailed analysis of this Modified Soft-Output Viterbi Algorithm (MSOVA) approach is covered in [14] where MSOVA simulations with varying (c, d) parameters are performed. However, the simulation results documented so far are only based on HR-SOVA update rule but not BR-SOVA. Since the performance of BR-SOVA in turbo decoding is to be investigated, the performance of MSOVA with the BR-SOVA update rule is also considered in this report.

With hardware considerations in mind, it is desirable to consider the computation of extrinsic reliability values in the domain of fixed point operations. This indicates that (c, d) parameters are to be quantized. Examples of scaling values of (c, d) and the number of bits that will be needed to represent these values in fixed point are tabulated in Table 2.1.

The simulation results for MSOVA with BR-SOVA and HR-SOVA update rules are presented and discussed in section 2.2.2.

2.1.4 Simplification of BR-SOVA Update Rule

According to the BR-SOVA update rules given in (1.28), when $u^s_{k-U} = u^c_{k-U}$, L^c_j is to be updated with reliability values of its own secondary concurrent paths,

according to the HR-SOVA update rule. The process is equivalent to that being performed for L_j^s, which is $L_j^s \leftarrow \min\left\{L_j^s, \Delta_k^s\right\}$. This implies that for BR-SOVA, the HR-SOVA reliability update rule needs to be applied twice for every pair of survivor-concurrent paths, with the survivor path update dependent on the results of the concurrent path update.

The additional updates will result in a more complicated implementation of BR-SOVA as compared to HR-SOVA. In addition to the hardware complexity to perform the updates, there is a need to store L_j^c, which will require extra memory to store concurrent path information. As the two updates to the reliability values have to be performed serially, the overall latency for BR-SOVA is expected to be increased.

With increased complexity, BR-SOVA is expected to take a longer time in simulation as compared to HR-SOVA. For hardware implementation, BR-SOVA is expected to consume more hardware resources and to have a longer processing latency.

Considering practical issues such as latency and hardware cost, it is desirable to investigate methodologies to reduce the simulation time, and more importantly to reduce the chip area used in hardware implementations of BR-SOVA. To this end, the following simplification for BR-SOVA is considered.

$$L_j^s \approx \begin{cases} \min\left\{\Delta_k^s, L_j^s\right\} & \text{when } u_j^s \neq u_j^c \\ \min\left\{\Delta_k^s + \Delta_j^c, L_j^s\right\} & \text{when } u_j^s = u_j^c \end{cases} \tag{2.5}$$

Equation (2.5) shows that L_j^s is updated with Δ_j^c instead of L_j^c when $u_k^s = u_k^c$. The analogy for the above simplification is as follows. Since L_j^c is supposed to be updated with the minimum among all its concurrent Δ_j, it can be deduced that

$$\Delta_j^c \geq L_j^c$$

which implies that

$$\Delta_k^s + \Delta_j^c \geq \Delta_k^s + L_j^c$$

This equation implies that L_j^s is updated with a value larger than $\Delta_k^s + L_j^c$, which means that it is less likely for $\Delta_k^s + \Delta_j^c$ to be smaller than L_j^s. The effect of this simplification will only be that L_j^s will be updated less often than in the original BR-SOVA. The effect of reduced update occurrences for $u_j^s = u_j^c$ may be investigated with simulations. Simulation results for the Simplified Battail Rule SOVA (SB-SOVA) update rule are presented and discussed in section 2.2.3.

The advantage in the SB-SOVA update rule can be observed as follows. With the simplification, L_j^s update can be performed immediately when the condition is met, without having to update L_j^c first. This eliminates the extra complexity involved in updating L_j^s with Δ_j of other concurrent paths. This also eliminates the need to wait for L_j^c to be updated before L_j^s update is performed, thus reducing the latency when SB-SOVA is applied.

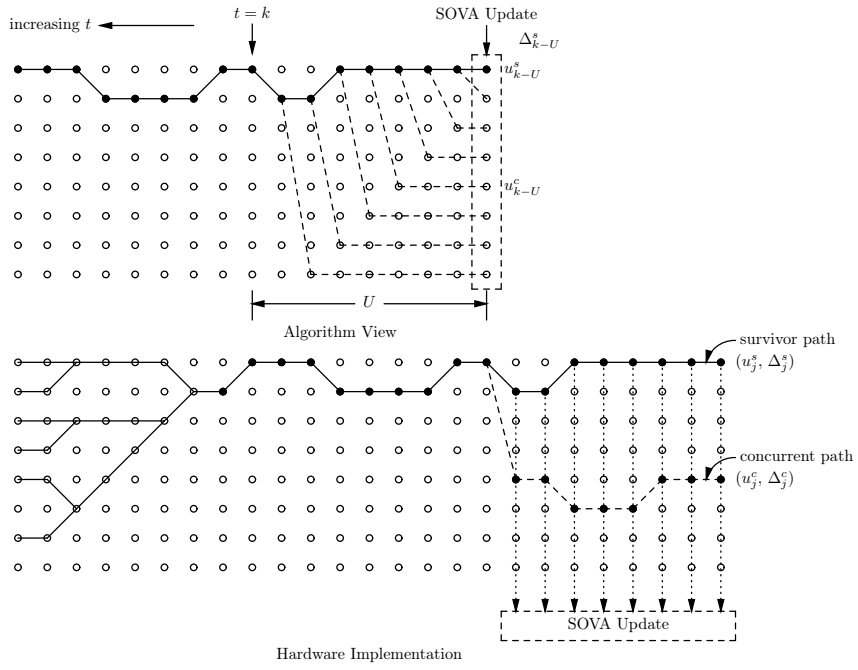


Figure 2.2: Two methods to perform SOVA updates

2.1.5 Reliability Update Methods for SOVA

Consider a U-stage survivor path sequence that is obtained at stage k, via Viterbi decoding traceback. The hardware decision bits and corresponding reliability values of the survivor path are denoted by

$$\{u^s_{k-j}, L^s_{k-j}\} \quad \text{for } j = 1, 2, \ldots, U$$

Given these information, there are two methods to perform updates on the reliability values L^s_{k-j}, namely the *algorithmic* and *hardware* methods. Figure 2.2 presents a graphical representation of these two methods of reliability updates for SOVA, which are described in the following sections.

Algorithmic

The algorithmic method presented in most technical papers involves performing updates on just the reliability value L^s_{k-U} of the last decision bit u^s_{k-U}. In this case, *all* the concurrent paths emerging from the survivor path are considered. During reliability updates, L^s_{k-U} is updated with Δ^s_{k-j} (HR-SOVA), L^c_{k-U} and L^s_{k-j} (BR-SOVA), or Δ^c_{k-U} and L^s_{k-j} (SB-SOVA).

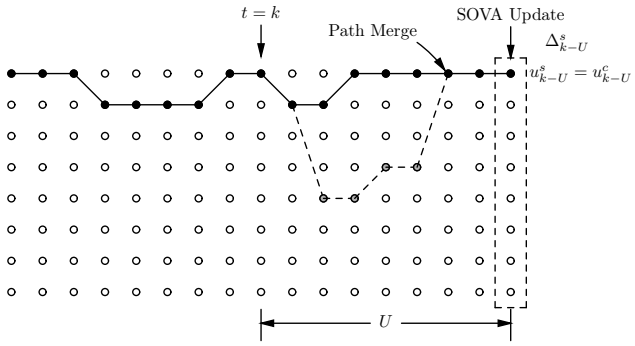


Figure 2.3: Merged survivor and concurrent paths

A U-stage survivor path is expected to have at most $U-1$ concurrent paths. For these concurrent paths, there may be cases where the concurrent path merges with the survivor path before stage $k-U$ (as shown in Figure 2.3). When the paths merge,

$$u^s_{k-U} = u^c_{k-U} \quad \text{and} \quad L^s_{k-j} = L^c_{k-j}.$$

which implies that no update is expected for HR-SOVA since the survivor and concurrent decision bits are the same.

As for BR-SOVA, the condition when the decision bits are the same is such that

$$\begin{aligned} L^s_{k-j} &= \min\left\{\Delta^s_k + L^c_{k-j}, L^s_{k-j}\right\} \\ &= \min\left\{\Delta^s_k + L^s_{k-j}, L^s_{k-j}\right\} \\ &= L^s_{k-j} \quad \text{always} \end{aligned}$$

Thus, there will once again be no reliability update in this case.

The detailed description of the algorithmic SOVA decoding is as follows. At every stage/time index k, the state metric M_i of every state i is obtained. The ML state s_k is selected among the N states based on maximum metric value.

Starting from state s_k, a U-stage traceback is performed to obtain the ML state sequence $\{s_{k-1}, \ldots, s_{k-U}\}$. The corresponding hard decision bits $\{u^s_{k-1}, \ldots, u^s_{k-U}\}$ and metric difference values $\{\Delta^s_{k-1}, \ldots, \Delta^s_{k-U}\}$ are also obtained.

For s_{k-1} to s_{k-U}, the corresponding survivor and concurrent paths are extracted to determine the decision bits u^s_{k-U} and u^c_{k-U}. The reliability value for decision bit u^s_{k-U} is then updated as follows.

For HR-SOVA,

$$L^s_{k-U} = \min_{u^c_{k-U} \neq u^s_{k-U}} \left\{\Delta^s_{k-j}\right\} \quad \text{for } j = 1, \ldots, U$$

For BR-SOVA,

$$L^s_{k-U} = \begin{cases} \min\left\{\Delta^s_{k-j}\right\}, & u^c_{k-U} \neq u^s_{k-U} \\ \min\left\{\Delta^s_{k-j} + L^c_{k-U}\right\}, & u^c_{k-U} = u^s_{k-U} \end{cases} \quad \text{for } j = 1, \ldots, U$$

which can be further simplified using the simplified BR-SOVA updating rules to obtain

$$L^s_{k-U} \approx \begin{cases} \min\left\{\Delta^s_{k-j}\right\}, & u^c_{k-U} \neq u^s_{k-U} \\ \min\left\{\Delta^s_{k-j} + \Delta^c_{k-U}\right\}, & u^c_{k-U} = u^s_{k-U} \end{cases} \quad \text{for } j = 1, \ldots, U$$

For the algorithmic SOVA method, the survivor path does not need to begin from a merged point. The survivor path depth U however needs to be sufficient for the path to merge at s_{k-U}.

Hardware

In the previous section, SOVA reliability updates are performed on a single L^s_{k-U} value with multiple concurrent paths. This section presents an alternative update method that is suitable for use in hardware implementations.

Given the same U-stage survivor path and its first concurrent path emerging from stage $k-1$, all U reliability values L^s_{k-j} within the same survivor path can be updated with L^s_k, as shown in the lower sub-figure of Figure 2.2.

The motivation for performing such an update method is to take advantage of the concurrent nature of hardware. With U sets of identical hardware that each performs reliability update for a single bit, the U instances of L^s_{k-j} can be updated concurrently within the same clock cycle.

Since reliability values of the survivor path from stage $k-1$ onwards are being updated, it is crucial to ensure that the survivor path has indeed merged. Hence, an important assumption to be made for this reliability update method is that the survivor path is starting at an ML state s_k where all the paths have merged, which means that a sufficiently large merge window, L is needed prior to performing SOVA reliability updates.

Given that the current trellis stage is computing the state metric at stage k, an L-stage Viterbi traceback is required to obtain the merged state at $k-L$. For all paths to merge at stage $k-L$, simulation results show that L has to be at least 5 to 10 times the constraint length, with the former suitable for rate $R = 1/2$ codes and the latter for higher coding rates. From stage $k-L$ onwards, a reliable U-stage survivor path is then determined via traceback. With that, the survivor path will have its ML state sequence starting from s_{k-L} be represented by s_{k-L-1} to s_{k-L-U}. The corresponding decision bits and metric differences will be

$$\left\{u^s_{k-L-1}, \ldots, u^s_{k-L-U}\right\}, \text{ and}$$
$$\left\{\Delta^s_{k-L-1}, \ldots, \Delta^s_{k-L-U}\right\}$$

The first concurrent path emerging from stage $k-L$ will have its decision bits and metric differences represented by

$$\left\{u^c_{k-L-1}, \ldots, u^c_{k-L-U}\right\}, \text{ and}$$
$$\left\{\Delta^c_{k-L-1}, \ldots, \Delta^c_{k-L-U}\right\}$$

For each L^s_{k-L-j} to be updated with sufficient number of reliability values, U has to be 3 to 5 times the constraint length. The effect of varying the merging and updating windows are presented with simulation results in Section 2.2.4.

2.2 Simulation Results

2.2.1 Simulation Environment

Based on the algorithmic studies made in Section 2.1, this section presents the simulation results obtained for these studies. Simulation is performed under an in-house simulation environment based on the open-source IT++ [16] package.

As the objective of the thesis is to evaluate the performance of a SOVA based turbo decoder under the LTE environment, the LTE internal interleaver is used for all simulations. In order to have meaningful performance comparisons of the turbo decoder with academic literature, an AWGN channel is used.

The performance of SOVA for simulation is determined by its BER and Block Error Rate (BLER) performance. The receiver in LTE accepts or discards data blocks based on the CRC checksum. If the CRC checksum of a block fails, the entire data block will be discarded. Under such circumstances, the BLER is a more accurate indication for performance than BER, as the BLER is a direct indication of the data throughput performance of the turbo decoder.

In turbo decoding, the MAP and Log-MAP algorithms give the optimal baseline performance for an a-posteriori probability decoder. The Max-Log-MAP algorithm is an approximation of the Log-MAP algorithm and its performance is sufficiently comparable to that of Log-MAP. In hardware implementations for turbo decoders, the Max-Log-MAP algorithm is commonly used due to the fact that Max-Log-MAP is reasonably simpler in hardware implementations, and yet able to achieve performance that is close to Log-MAP. The Max-Log-MAP algorithm therefore serves as a basis of comparison when evaluating the performance of the SOVA algorithms.

2.2.2 Optimum Reliability Thresholding and Scaling Factors for SOVA

This section presents the simulation results for SOVA with reliability thresholding Δ_{TH} and MSOVA applied. The values of Δ_{TH} and sets of MSOVA parameters (c, d) used for the simulation are listed as follows.

$$\Delta_{TH} = \{8, 16, 32, 64\}$$
$$(c, d) = \{0.25, 0.5, 0.68, 0.75, 0.8, 1.0\}$$

Table 2.2: Simulation conditions to determine optimum Δ_{TH}

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit
Simulation Parameters	
Length of Survivor/Concurrent paths: $U = 4\,416$	
Fixed:	$(c, d) = (0.75, 1.00)$
Variant:	$\Delta_{TH} = \{8, 16, 32, 64\}$
Number of half iterations:	$12, 16$

The parameters listed above are applied to both HR-SOVA and BR-SOVA. For Max-Log-MAP, a log-scale factor of 0.75 is applied to the extrinsic reliability output $L_e(\hat{u}_k)$, as the addition of a log-scale factor will improve the Max-Log-MAP decoder's performance [14].

In order to determine the effect of Δ_{TH}, simulations are performed with conditions listed in Table 2.2. The BER and BLER performance curves are shown in Figure 2.4. Note that the results of $\Delta_{TH} = 32, 64$ for HR-SOVA and $\Delta_{TH} = 32$ for BR-SOVA are not included in the plots because for both SOVAs, the results for $\Delta_{TH} = 16$ and $\Delta_{TH} = 32$ are similar and thus only results for $\Delta_{TH} = 16$ are shown. In the case of $\Delta_{TH} = 64$, the results obtained by using HR-SOVA indicate that Δ_{TH} is too large for the thresholding to be effective and thus the non-meaningful result has been omitted from the plots.

Another important observation from the simulation results is that the HR-SOVA and BR-SOVA turbo decoders both require at least 16 half iterations to obtain BER/BLER performance that is comparable to Max-Log-MAP with 12 half iterations. This is illustrated in a separate graph as shown in Figure 2.5. The number of iterations has an impact on the overall data throughput and is discussed in Section 3.7.

As observed from Figure 2.4, the performance for HR-SOVA with $\Delta_{TH} = 16$ is better than $\Delta_{TH} = 8$. With reliability thresholding, the performance of HR-SOVA is approximately 0.1 to 0.2 dB worse than Max-Log-MAP. The performance of HR-SOVA is found to be similar to that of Max-Log-MAP at lower Signal-to-Noise Ratio (SNR), but at higher SNR, the BLER performance is worse than Max-Log-MAP and increasing the thresholding value does not improve the performance.

The performance for BR-SOVA with reliability thresholding is found to be comparable to Max-Log-MAP for BER performance. Its BLER performance is found to be approximately 0.1 dB better than Max-Log-MAP. At SNR of between

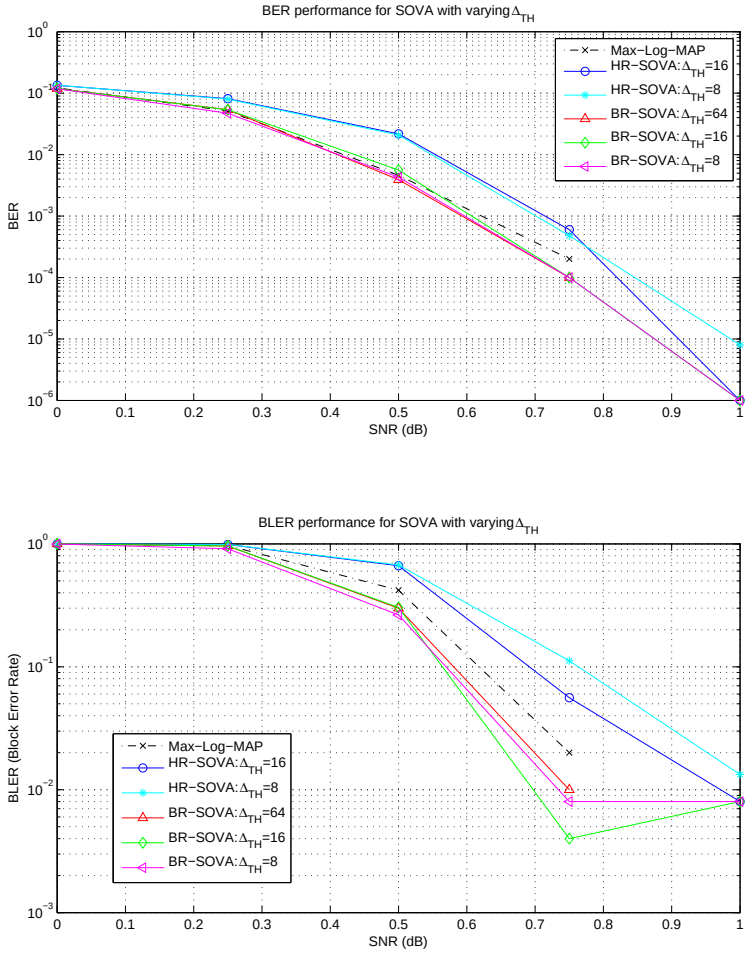


Figure 2.4: BER and BLER performance for SOVA with varying Δ_{TH}

0 dB and 0.5 dB, the BLER performance of BR-SOVA is best at $\Delta_{TH} = 8$. For SNR higher than 0.5 dB, BR-SOVA with $\Delta_{TH} \geq 16$ and $\Delta_{TH} = 64$ have BLER that is better than Max-Log-MAP.

The simulation results in Figure 2.4 are based on SOVA reliability updating performed over the entire block length K_i, using the algorithmic method as described

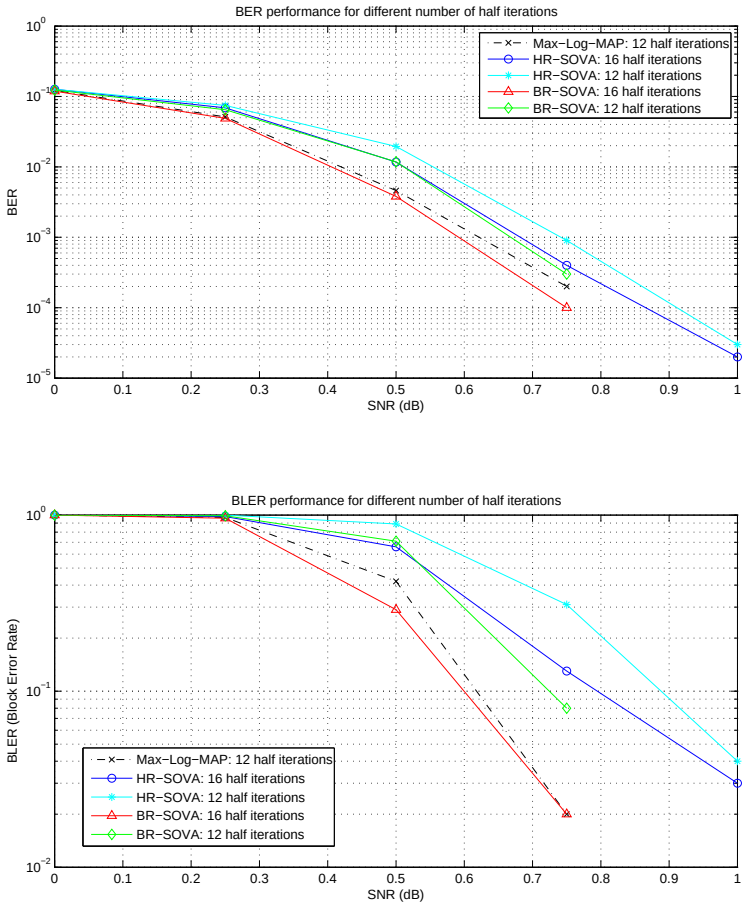


Figure 2.5: BER and BLER performance for SOVA with varying number of iterations

in section 2.1.5, that is $U = K_i$. Although the performance for BR-SOVA is better than Max-Log-MAP, it is however not feasible from the hardware architecture point of view. The algorithmic method translates to a higher hardware resource utilization, especially when the block length is large. To take a more practical approach, the reliability update path is instead limited by a window size U, where $U < K_i$.

Table 2.3: Simulation conditions to determine optimum MSOVA parameters

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit
Simulation Parameters for HR-SOVA	
Update Depth U:	40
Δ_{TH}:	16
(c, d):	$(0.75, 1.0)$ $(0.75, 0.75)$ $(0.75, 0.8)$ $(0.68, 0.8)$
Simulation Parameters for BR-SOVA	
Update Depth L:	40
Δ_{TH}:	8
(c, d):	$(0.75, 1.0)$ $(0.75, 0.75)$ $(0.75, 0.8)$ $(0.68, 0.8)$

The value of U then becomes an additional parameter to vary for the simulation. As a rule thumb, an update window U of 10 times the constraint length is sufficient for all paths to merge, and thus $U = 40$ is selected for the following simulations.

The simulation results presented so far are for conditions where the MSOVA parameters are kept constant, while determining the optimum Δ_{TH} for each SOVA method. To determine the optimum MSOVA parameters, simulations under conditions listed in Table 2.3 are performed and the results are shown in Figure 2.6.

As observed from the results, the optimum (c, d) parameters for HR-SOVA is found to be $(0.75, 0.8)$. However, it should be noted that it is not possible to precisely represent the value 0.8 using fixed point representation which will translate into a performance degradation when implemented in hardware. To mitigate this, more bits may be used to represent the scale factor. As an example, 0.8 can be approximated by $0.796875 = 0.110011$b which will require 7 bit to represent. A larger multiplier will be needed, and the result will have a longer wordlength. This will translate to either higher storage requirements for the extrinsic values, or will require truncation of the result of the multiplication. The truncation of the result will however reduce the advantage of choosing to have higher precision for the multiplication.

For BR-SOVA, the parameter set $(0.75, 1.00)$ is observed to give the best performance, with performance comparable to that of Max-Log-MAP.

As observed from the simulation results, the improvements on HR-SOVA is possible with Δ_{TH} and (c, d) parameters, but the price to pay is the increased wordlength required to implement the scale factor of 0.8. For BR-SOVA, per-

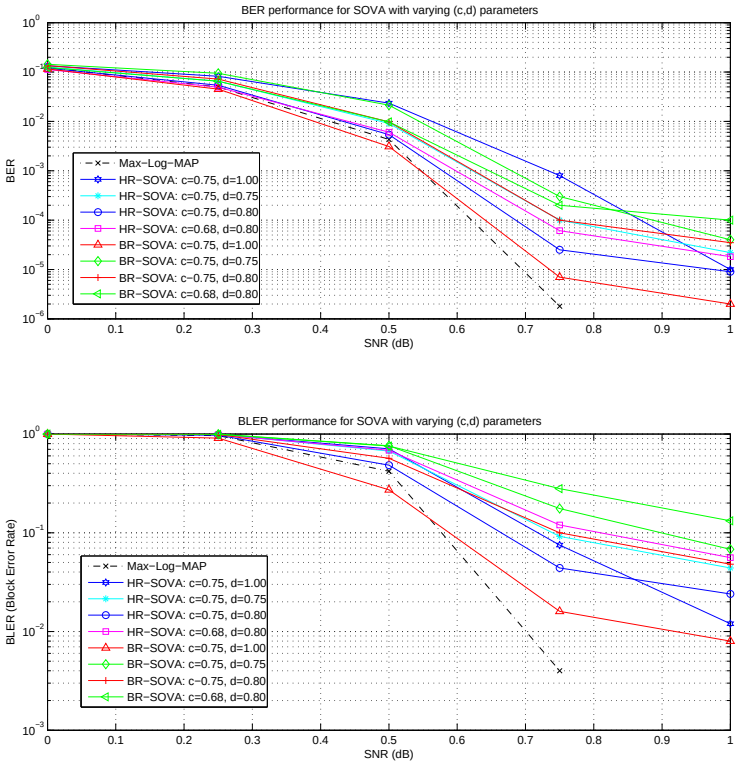


Figure 2.6: Simulation results for SOVA with varying (c, d) parameters

formance that is comparable or even better than Max-Log-MAP is achieved with attenuation of only the extrinsic reliability output (parameter c) and Δ_{TH}. There is no need for further attenuation of the intrinsic reliability output (parameter d) to obtain desirable BER/BLER performance for BR-SOVA. This is equivalent to the same log scale factor being applied to Max-Log-MAP. Thus, it can be concluded that for BR-SOVA, a properly selected reliability threshold value and an attenuation of the extrinsic reliability output are sufficient to obtain acceptable BER and BLER performance.

With the above presented results, the optimum MSOVA parameters and reliability threshold values for HR-SOVA and BR-SOVA are summarized as follows:

Table 2.4: Simulation conditions with SB-SOVA

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit
Simulation Parameters	
Update method:	Algorithmic with $U = 40$
	$(c, d) = (0.75, 1.00)$
	$\Delta_{TH} = \begin{cases} 8 & \text{SNR} \leq 0.75 \text{ dB} \\ 16 & \text{SNR} > 0.75 \text{ dB} \end{cases}$

HR-SOVA:

$$(c, d) = (0.75, 0.8)$$
$$\Delta_{TH} = 16 \quad \text{for all SNR}$$

BR-SOVA:

$$(c, d) = (0.75, 1.0)$$
$$\Delta_{TH} = \begin{cases} 8 & \text{SNR} \leq 0.5 \text{ dB} \\ 16 & \text{SNR} > 0.5 \text{ dB} \end{cases}$$

2.2.3 Effects of Simplified BR-SOVA Update Rule

As described in section 2.1.4, the main challenge of implementing BR-SOVA in hardware is due to the high complexity involved in updating the concurrent reliability values L_j^c. The SB-SOVA update rule described in Section 2.1.4 will eliminate the process of updating the concurrent reliability value L_j^c. This simplification will reduce the hardware complexity of SB-SOVA as compared to BR-SOVA.

To determine the extent of the degradation in the performance of SB-SOVA, BER and BLER performance simulations under the conditions in Table 2.4 were performed and the BER and BLER plots are shown in Figure 2.7.

The simulation results show that SB-SOVA has less than 0.1 dB degradation as compared to BR-SOVA. Even with this degradation, the performance of SB-SOVA is still comparable to that of Max-Log-MAP with 12 iterations, and with less than 0.15 dB in degradation as compared to Max-Log-MAP with 16 iterations.

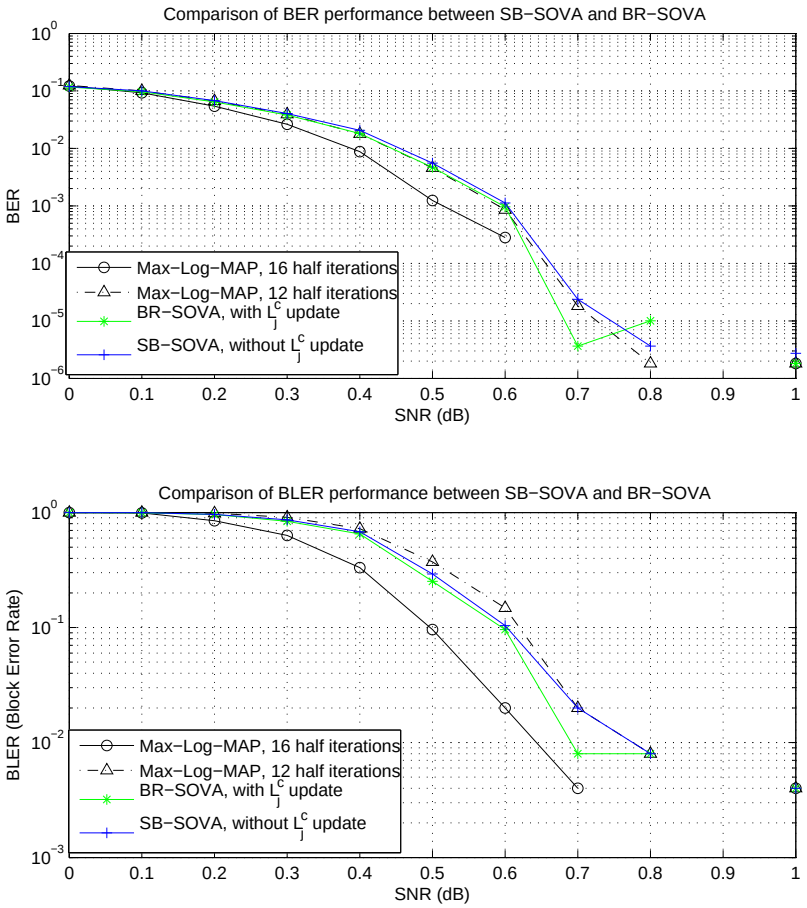


Figure 2.7: Simulation results for SB-SOVA versus BR-SOVA

This shows that SB-SOVA update rule is a reasonable simplification to make for BR-SOVA from the BER/BLER performance point of view.

2.2.4 Comparison of Reliability Update Methods for SOVA

As described in section 2.1.5, there are mainly two ways to perform SOVA reliability updates on the traceback trellis, namely the algorithmic and hardware methods.

Table 2.5: Simulation conditions with different reliability update methods

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit
Simulation Parameters	
$\Delta_{TH} = 8$	
$(c, d) = (0.75, 1.0)$	
Reliability update method:	
Algorithmic U:	40
Hardware (L, U):	$(24, 40)$, $(20, 20)$

The simulation results presented so far are only based on the algorithmic method. In this section, simulations are performed to show the equivalence of the algorithmic and hardware methods.

The performance of the two methods for SB-SOVA are presented in Figure 2.8 with the simulation conditions in Table 2.5. The results show that the BER and BLER performance of the two methods deviate from each other by at most 0.05 dB. This implies that the two reliability update methods are equivalent. Since the results for the algorithmic and hardware methods are similar, all future simulation results presented will utilize one of the two proposed methods.

2.2.5 Results for Hardware Reliability Update Method

As described in Section 2.1.5, the hardware update method is made up of an L-step merge stage and a U-step update/decode stage. At the merge stage, the traceback has to be sufficiently long for all states to merge. Based on academic studies, selecting L that is at least 5 times the constraint length is sufficient for all paths to merge into a common starting state. For LTE turbo codes, the constraint length is 4 and hence a minimum $L = 20$ is sufficient.

Simulations can be performed on SOVA to determine the minimum (L, U) parameter set that will give desirable performance. The conditions tabulated in Table 2.6 are used to run the simulation. The BER and BLER performance curves are shown in Figure 2.9. As observed from Figure 2.9, the depth of SOVA update window has a significant impact on the performance of both HR-SOVA and SB-SOVA. For similar performance, HR-SOVA will need a longer reliability update path U than SB-SOVA. With $(L, U) = (24, 40)$, SB-SOVA is able to achieve similar performance as Max-Log-MAP(12 iterations) and less than 0.2 dB degra-

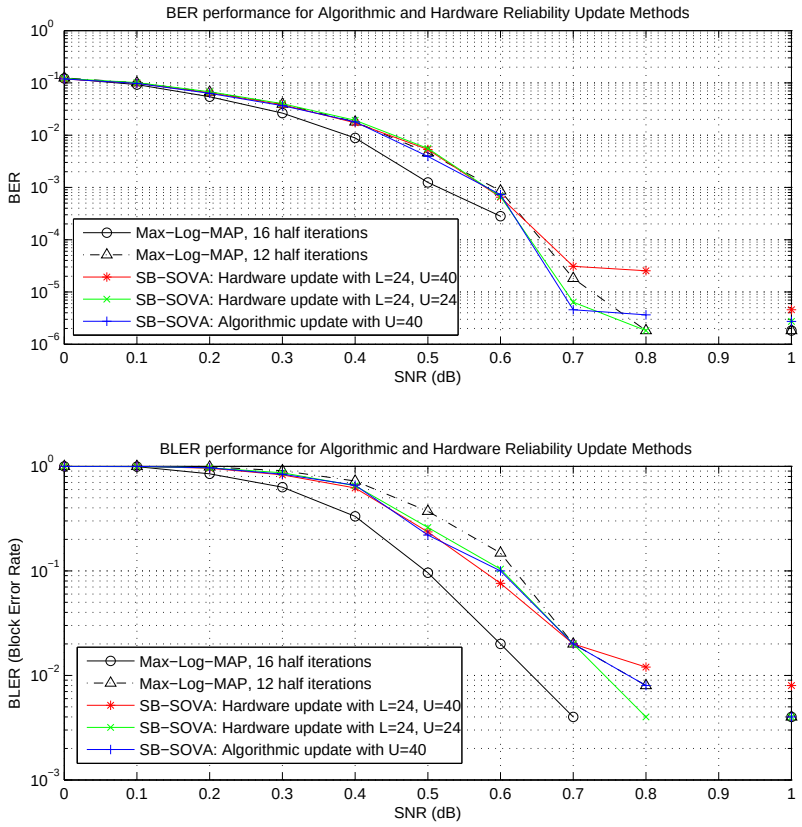


Figure 2.8: Simulation results for different reliability update methods

dation in performance as compared to Max-Log-MAP with 16 iterations. For the depths of $(L, U) = (24, 24)$, the performance of SB-SOVA is only 0.1 dB worse than Max-Log-MAP with the same number of iterations. For a shorter update depths of $(L, U) = (24, 20)$, the performance of SB-SOVA is still reasonably good, with less than 0.2 dB degradation from Max-Log-MAP. It is noted that the required Δ_{TH} increases with the SNR for SB-SOVA. At higher SNR, a larger Δ_{TH} is required to achieve satisfactory performance.

As for HR-SOVA, larger depths are required to improve its performance. For $(L, U) = (40, 40)$, HR-SOVA is still 0.2 dB worse off than SB-SOVA with smaller depths.

Table 2.6: Simulation conditions to obtain merge and update depths

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit
Simulation Parameters for HR-SOVA	
Update method:	Hardware
(L, U):	$(40, 40)$ $(24, 24)$
	$(c, d) = (0.75, 0.80)$, $\Delta_{TH} = 16$
Simulation Parameters for SB-SOVA	
Update method:	Hardware
(L, U):	$(24, 40)$ $(24, 24)$ $(24, 20)$ $(20, 20)$
	$(c, d) = (0.75, 1.00)$
	$\Delta_{TH} = \begin{cases} 8 & \text{SNR} \leq 0.75 \text{ dB} \\ 16 & \text{SNR} > 0.75 \text{ dB} \end{cases}$

The above simulation results serve as a reference for selecting the appropriate merge and update depths to be used in hardware architecture for an SB-SOVA turbo decoder.

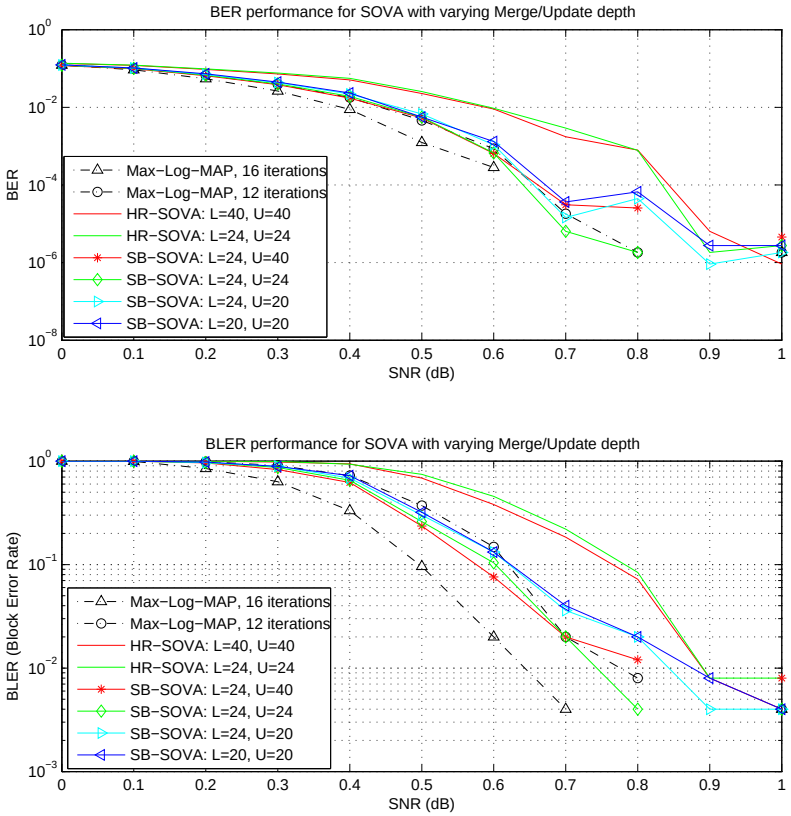


Figure 2.9: BER and BLER performance of SOVA for varying merge and update depths

Chapter 3

SISO Decoder Hardware Architectures

The simulation results presented in Section 2.2 determined the SOVA parameters that are needed for hardware implementation such as the merging and update depths. The simulation results also showed that the performance of the proposed SB-SOVA decoder was similar to that of a Max-Log-MAP decoder.

This chapter presents a discussion on the various hardware architectures that can be applied to implementations of SISO decoders. It begins with a literature review of the existing decoders based on Max-Log-MAP and HR-SOVA in Sections 3.1 and 3.2 respectively. Next, the proposed hardware architecture for SB-SOVA is presented in Section 3.3 together with optimizations that can be made to reduce the hardware complexity of the SB-SOVA decoder in Sections 3.4 and 3.5. In Section 3.6, considerations for a parallelized decoder is discussed.

Throughout the chapter, the key implementation considerations for the hardware design, namely the hardware complexity, memory utilization and latency, will be presented and discussed. Section 3.7 concludes the chapter by comparing the proposed SOVA based decoders with a baseline of the more traditional Max-Log-MAP based implementations so that the trade-offs between the hardware costs and performance can be evaluated.

3.1 Hardware Architecture of Max-Log-MAP

A typical hardware architecture for a Max-Log-MAP turbo decoder consists of recursion units to process both the forward recursion state metric $\alpha_k(s)$ and the backward recursion state metric $\beta_{k-1}(s')$, as described in section 1.2.2. Due to the backward recursive nature of $\beta_{k-1}(s')$, computation of the first $\beta_{k-1}(s')$ can only begin when the last bit of the block has been received. The timing behavior for a classical Max-Log-MAP architecture is as shown in Figure 3.1.

As shown in Figure 3.1, the latency for the first decoded output to be ready is $3K_i$, where K_i is the data block length. The latency marked in the diagram excludes the acquisition time for the channel data $\boldsymbol{y_k}$ and $L(u_k)$ since it is assumed

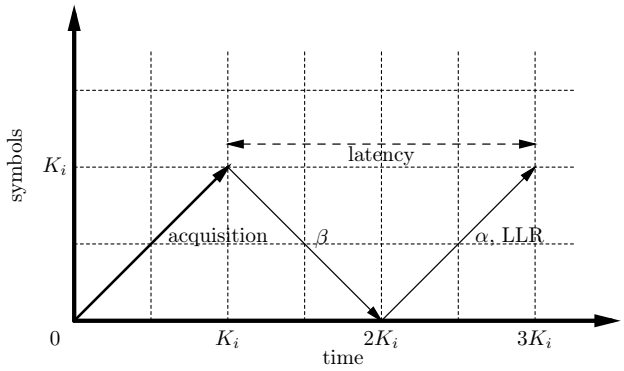


Figure 3.1: Timing behavior of a classical MAP-based turbo decoder

that all data are available before the start of the turbo decoding. The latency also excludes the time taken before the first branch metric γ is available, since this value is expected to be much smaller than K_i. When the block length K_i is large, such as 6 144 bit in LTE, the resulting latency is significant. To reduce the latency, the input data block is usually divided into multiple windows, each of size W. Various architectures with different configurations of forward and backward recursion units have been proposed in [2] and [17]. The trade-off lies between required latency and chip area.

A hardware architecture of a Max-Log-MAP SISO decoder is presented in Figure 3.2. The architecture consists of three branch metric computation (`BMC`) units, one forward/alpha recursion processing (`ARP`) unit, two backward/beta recursion processing `BRP` units and a `LLR` unit. These hardware units are all executing concurrently. The `BMC` units are responsible for computing the branch metrics γ, as illustrated in (1.10). The `ARP` unit is responsible for processing the forward state metric α according to (1.11) while the `BRP` units determine the backward state metric β according to (1.12). The purpose of having two `BRP` units is that while one unit is in acquisition mode, the other unit is performing the actual computation of the backward state metric as shown in Figure 3.3. The first `BRP` is used for acquisition to determine initial backward state metric β_k and the second `BRP` will process β_{k-1} down to β_{k-W}. Lastly, the `LLR` unit computes the final reliability value $L_e(\hat{u}_k)$ using both α and β. The block diagrams of the `ARP` and `LLR` units are as shown in Figure 3.4 and Figure 3.5 respectively.

The `ARP` determines forward state metrics α_{k-1} to α_{k-W} for each of the N states in a window of W. These $N \times W$ forward state metrics will be stored in the memory and will be used later to compute the final state metric together with the backward state metrics β_{k-W} to β_{k-1}. As α and β are calculated in opposite directions, the output of `ARP` has to be stored in a Last In First Out (LIFO) memory.

The hardware considerations for the Max-Log-MAP SISO decoder described

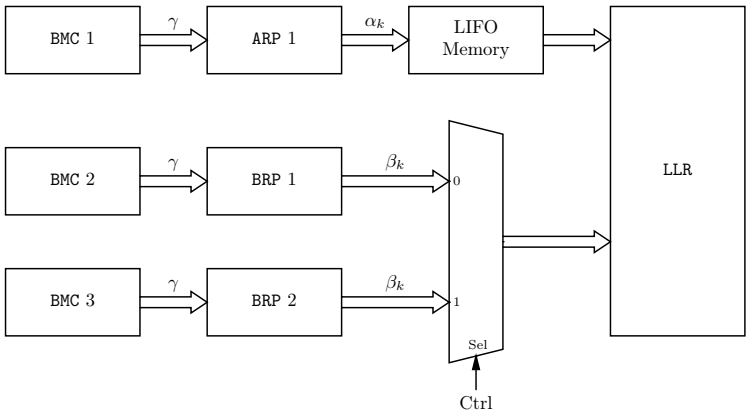


Figure 3.2: System architecture of a Max-Log-MAP decoder

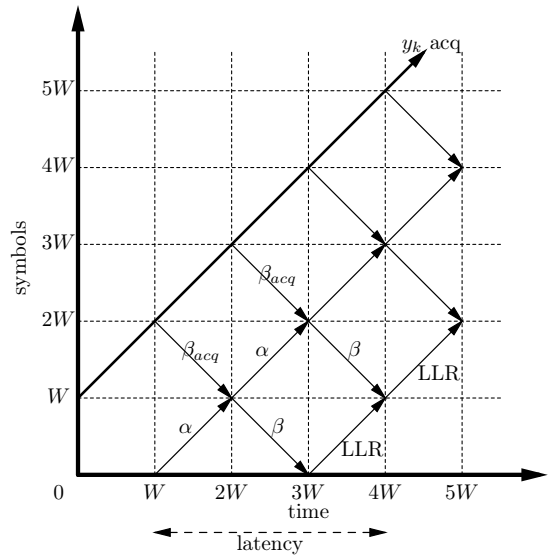


Figure 3.3: Write behavior diagram for Max-Log-MAP decoder

above are covered in the following sections.

3.1.1 Resource Utilization

An estimated hardware resource utilization for the Max-Log-MAP turbo decoder is computed as follows. The block diagram of a recursion unit is shown in Figure 3.4.

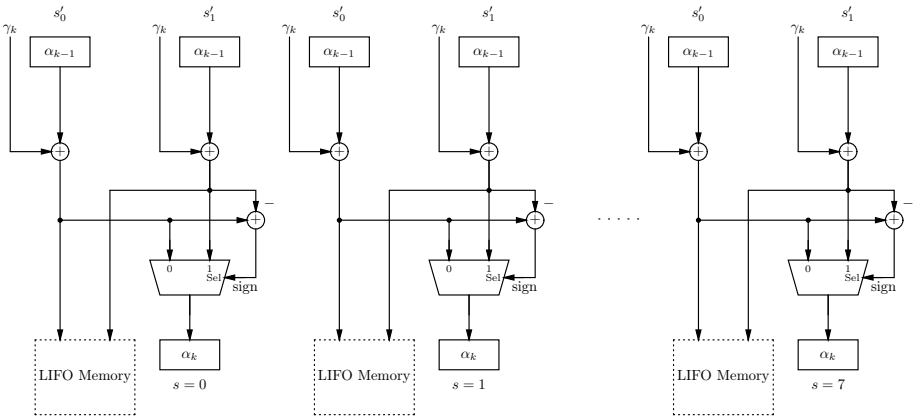


Figure 3.4: ARP unit for Max-Log-MAP

Each recursion unit is made up of $3N$ units of η_α-bit adders and N units of η_α-bit MUXes to compute the state metric. The block diagram of the LLR unit is as shown in Figure 3.5. As the objective of the LLR unit is to select the maximum reliability value, the unit is made up of multiplexers and adders to perform comparison and selection.

As observed from Figure 3.2, that there are N units of η_β-bit MUXes being used to select the β metric between BRP 1 and BRP 2 to be used by the LLR block.

The overall hardware resource utilization for Max-Log-MAP is summarized in Table 3.1.

3.1.2 Memory Requirements

The memory requirement for storing forward state metric α for each window is computed as follows. For turbo decoder with $M = 8$ and $W = 768$, the storage requirement for α will be

$$N \cdot W = 6\,144\ \eta_\alpha \text{ bit}$$

where η_α is the data width of α. If $\eta_\alpha = 8$, the storage requirement becomes $49\,152 = 48$ kbit for each window. The total storage for the entire data block will be $48 \times 8 = 384$ kbit. As the storage requirement is proportional to the total block length K_i, the total storage requirements for Max-Log-MAP is relatively high.

3.1.3 Latency

As shown in the Figure 3.3, the decoder takes $3W$ cycles to decode a block of W symbols. The latency can be reduced to $2W$ if values β and LLR can be obtained concurrently. Since α has already been computed between $t = W$ and $t = 2W$, the

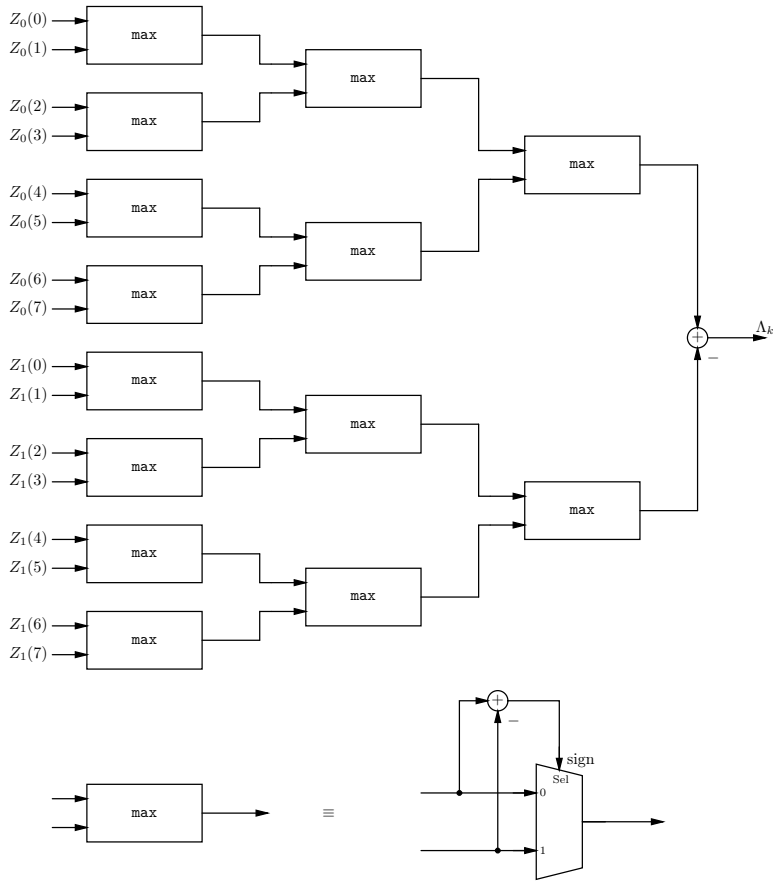


Figure 3.5: LLR unit for Max-Log-MAP

LLR can be computed immediately once β is available between $t = 2W$ and $t = 3W$, and written into memory. Since β is made available in the reverse order, the LLR will be computed in reverse order and thus can be stored in a LIFO memory. With this, the LLR values read out from the LIFO in the next iteration will be in the correct ascending order.

With parallel processing of a windowed channel input, the total latency is reduced from $2K_i$ to $2W$. $2W$ can still be a large number, if the block length of the channel input is large. For example, in LTE, the maximum block length K_i is 6 144. If the channel data block is divided into 8 windows, the size of each window will be $W = 768$, which gives a latency of 1 536 clock cycles.

Table 3.1: Resource utilization for Max-Log-MAP

Resources in $\eta_{\alpha,\beta}$ bit		
	Adders	MUXes (2-to-1)
`BMC` (3 units)	15	-
Recursion Units (`ARP`, `BRP 1`, `BRP 2`)	$9N$	$3N$
β selection between `BRP 1` & `BRP 2`	-	N
`LLR` unit	$6N-1$	$2(N-1)$
Total Resource Usage	$15N+14$	$6N-2$

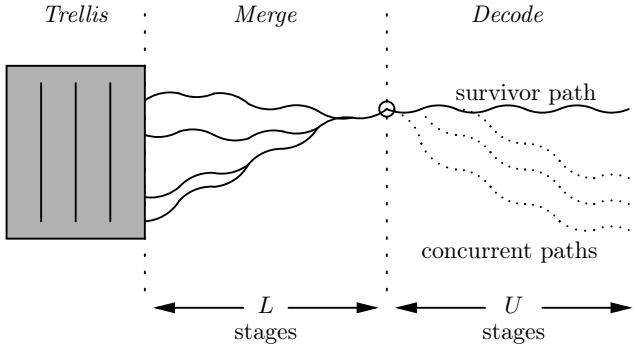


Figure 3.6: Hardware stages in SOVA

3.2 Hardware Architecture for HR-SOVA

Various hardware architectures for SOVA have been proposed [3][4][5], and these architectures are typically made up of 3 stages, namely the trellis, merge and decode stages, as illustrated in Figure 3.6.

3.2.1 Trellis Stage

The first stage is the *trellis* stage that computes the state metrics of each of the 2^{ν} states, according to (1.22), that can be separated into the recursion and the branch metric terms as as shown in (3.1).

$$M_k(S_k) = M_{k-1}(S_{k-1}) + \underbrace{\frac{1}{2}L(u_k)\cdot u_k + \frac{1}{2}L_c\cdot y_{k,1}\cdot u_k + \frac{1}{2}\sum_{v=2}^{n} L_c\cdot y_{k,v}\cdot x_{k,v}}_{\text{branch metric}} \quad (3.1)$$

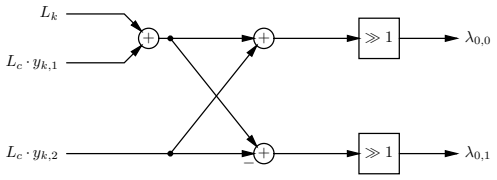


Figure 3.7: `BMC` module in the trellis unit

Assuming that the expected parity bit $x_{k,2} = +1$, or equivalently that the parity bit has a value of `'0'`, and let the branch metrics for the cases when the expected systematic bits of $u_k = 0$ and $u_k = 1$ be denoted by $\lambda_{0,0}$ and $\lambda_{1,0}$ respectively. These branch metrics are then calculated as follows

$$\begin{aligned}\lambda_{0,0} &= \frac{1}{2}L_k(+1) + \frac{1}{2}L_c \cdot y_{k,1}(+1) + \frac{1}{2}L_c \cdot y_{k,2}(+1) \\ &= \left(\frac{1}{2}L_k + \frac{1}{2}L_c \cdot y_{k,1}\right) + \frac{1}{2}L_c \cdot y_{k,2} \qquad (3.2)\\ \lambda_{1,0} &= \frac{1}{2}L_k(-1) + \frac{1}{2}L_c \cdot y_{k,1}(-1) + \frac{1}{2}L_c \cdot y_{k,2}(+1) \\ &= -\left(\frac{1}{2}L_k + \frac{1}{2}L_c \cdot y_{k,1}\right) + \frac{1}{2}L_c \cdot y_{k,2} \qquad (3.3)\end{aligned}$$

where $y_{k,1}$ and $y_{k,2}$ are the received channel systematic and parity values respectively, and L_k is input extrinsic information. Likewise, by assuming once again that the expected parity bit $x_{k,2} = -1$, the branch metrics for the expected systematic bits of $u_k = 0$ and $u_k = 1$ denoted by $\lambda_{0,1}$ and $\lambda_{1,1}$ respectively can be calculated.

$$\begin{aligned}\lambda_{0,1} &= \frac{1}{2}L_k(+1) + \frac{1}{2}L_c \cdot y_{k,1}(+1) + \frac{1}{2}L_c \cdot y_{k,2}(-1) \\ &= \left(\frac{1}{2}L_k + \frac{1}{2}L_c \cdot y_{k,1}\right) - \frac{1}{2}L_c \cdot y_{k,2} = -\lambda_{1,0} \qquad (3.4)\\ \lambda_{1,1} &= \frac{1}{2}L_k(-1) + \frac{1}{2}L_c \cdot y_{k,1}(-1) + \frac{1}{2}L_c \cdot y_{k,2}(-1) \\ &= -\left(\frac{1}{2}L_k + \frac{1}{2}L_c \cdot y_{k,1}\right) - \frac{1}{2}L_c \cdot y_{k,2} = -\lambda_{0,0} \qquad (3.5)\end{aligned}$$

It can be observed that $\lambda_{0,1}$ and $\lambda_{1,1}$ are simply the negative values of $\lambda_{1,0}$ and $\lambda_{0,0}$ respectively. This implies that there is only a need to generate two of the four branch metrics, since the other two can be easily obtained in the `ACS` when needed. A branch metric calculation unit `BMC` as shown in Figure 3.7 is used to generate $\lambda_{0,0}$ and $\lambda_{0,1}$.

Other than the `BMC` unit, there are 2^ν `ACS` in the trellis unit that will perform recursion by adding the branch metric to the previous survivor path metric, compare

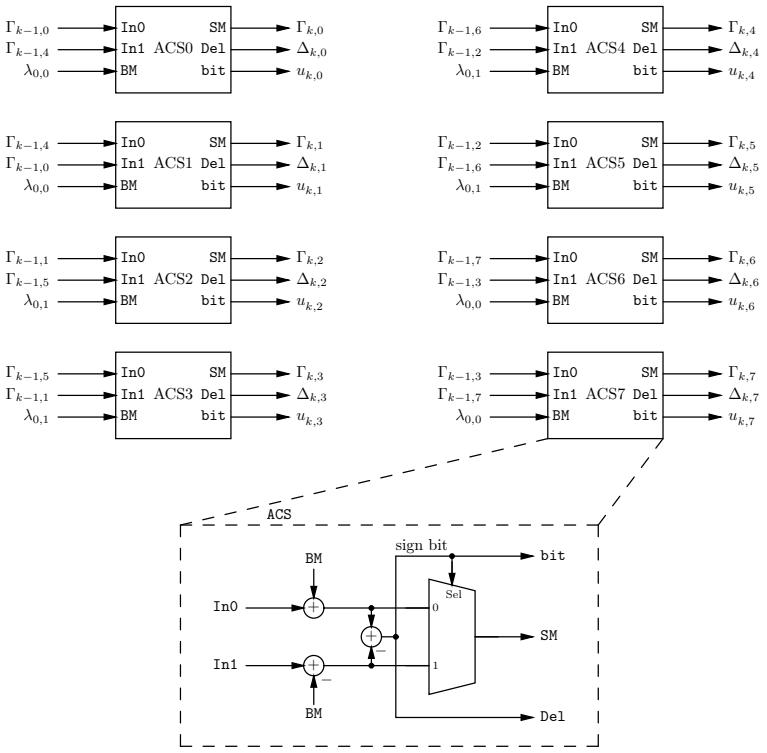


Figure 3.8: Trellis unit for LTE consisting of 8 ACS units

the two resultant metrics and finally select a survivor path metric to be saved for use at the next stage. Each ACS will perform calculations for one state, and the connections between the ACS are dependent on the generator's polynomials. In the case of the LTE turbo code, there are a total of 8 ACS modules connected together as shown in Figure 3.8.

A close up view of the ACS can be seen in Figure 3.8. By taking advantage of the fact that the branch metric and its inverse are negative values of one another, the ACS uses a subtractor at one of its input branches to obtain the required inverse branch metric. Thus by using one addition and one subtraction, the path metrics for both cases when the input decision bit is '1' and '0' can be obtained. The two path metrics are then compared to select the survivor (larger) metric which will be output from ACS and stored for the next stage of the trellis. The hard decision bit corresponding to the selection of the survivor path, together with the difference between the two metric values are also output to be used in later stages for merging and SOVA updates.

3.2.2 Merge Stage

The *merge* stage of depth L performs Viterbi decoding on the hard decision bits determined at the trellis stage. The depth L has to be sufficiently large for all 2^{ν} paths to merge after L stages, and is usually expressed in terms the constraint length of the encoder K. The depth L is dependent on the code rate of the channel and simulations show that a rather large merge depth of $L = 10K$ is required for all paths to merge, since the high coding rates that are defined in the LTE standard need to be supported. In hardware, the merge stage can either be implemented via the Register Exchange (RE) or traceback method. The block diagram of a RE unit suitable for use in LTE is shown in Figure 3.9.

The RE method utilizes registers to store all the $N \cdot L$ decision bits within the trellis. Each row in the RE unit contains the decision bits of the entire path of length L corresponding to the state of the first register of the row. The hard decision bits from the trellis unit are used as the select signals for the MUXes to control the state exchanges. The connections between the columns of registers are identical for all columns and dependent on the generator polynomial. Assuming that the depth of the RE is sufficient for merging, the output of all N rows of the RE unit would give the same decision bit (i.e. decision bit of the survivor path), which would be selected as the estimated received bit $\hat{u}_k$.

For the case of traceback, the decision bits are stored in a memory instead of registers, and a decision bit d stages away from a given state is to be determined by traversing d steps backwards in a trellis. The main advantage of traceback is that it can be implemented efficiently in dense memory, but the drawback is that there is increased latency as compared to RE. Both methods are commonly used in hardware designs of Viterbi decoders, and the chosen method is usually dependent on the trade-off between latency and hardware utilization. In this thesis, only the RE method is considered, due to the short latency required for the LTE decoder.

3.2.3 Decode Stage

The *decode* stage of depth U performs Viterbi decoding and reliability updates on the metric difference values obtained at the output of the trellis stage. Reliability updates are performed on the U reliability values on the survivor path $L^s_{k-L-1}, L^s_{k-L-2}, \ldots, L^s_{k-L-U}$ with each reliability value being updated for up to U times. The output of the decode stage, multiplied by the hard decision bit, i.e. $\hat{u}_{k-L-U} \cdot L^s_{k-L-U}$ is the intrinsic output of the SOVA SISO decoder.

Simulations show that the required update depth U for the decode stage is smaller than the merge depth L and for code rate of 1/3, a depth of approximately $5K$ is sufficient for the decode stage to provide satisfactory results.

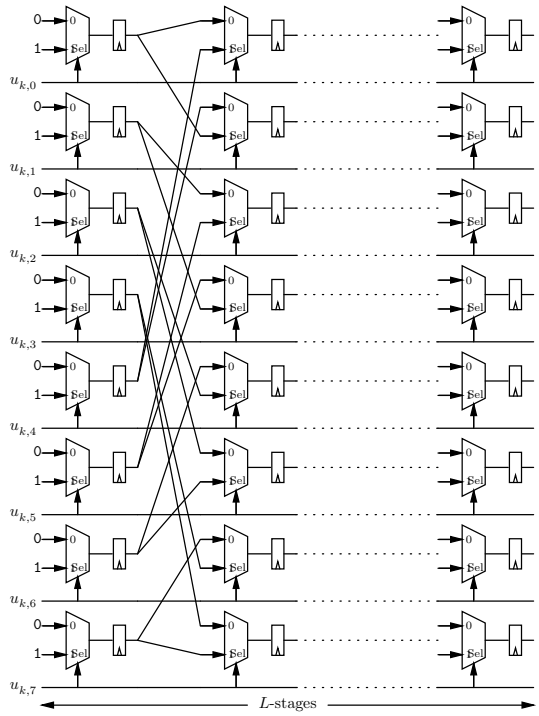


Figure 3.9: Block diagram of register exchange unit

3.2.4 Example of Hardware Architecture

Based on the architectures proposed in the literature [3][4][5], the block diagram of the hardware architecture for a HR-SOVA decoder is as shown in Figure 3.10. The trellis unit (`TRU`) performs the branch metric computation followed by the Add-Compare-Select (ACS) operations as described in Section 3.2.1. The Survivor Memory Unit (SMU) performs as the merge stage to determine the ML path at L stages away. The Path Comparison Unit (PCU) has a RE unit with similar structure as that in the `SMU` and it performs the decode stage with depth U. Viterbi decoding is performed using hardware decision bits that are stored in the First In First Out (FIFO) memory (`FIFO` U) as shown in the figure. There are two FIFOs required in the architecture; the first of which (`FIFO` U) is used to store the hard decision bits u_k decoded by the trellis unit and the second (`FIFO` Δ) is used to store the metric difference Δ_k computed by the trellis unit. For each stage, u_k and Δ_k for all 2^ν states will be stored. The `SMU` and `PCU` are made up of columns of RE units. Each row of RE registers stores the hard decision sequence of the respective state.

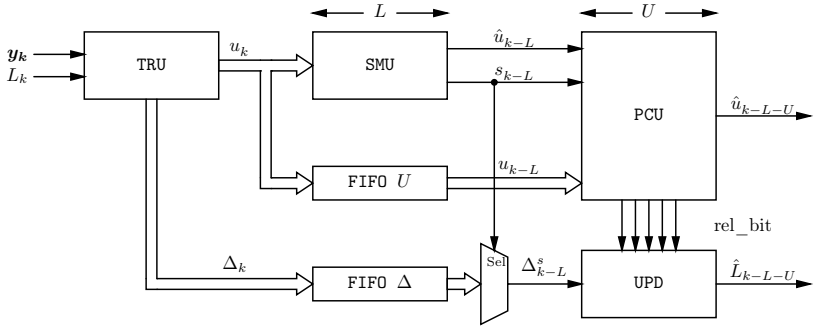


Figure 3.10: System architecture for HR-SOVA decoder

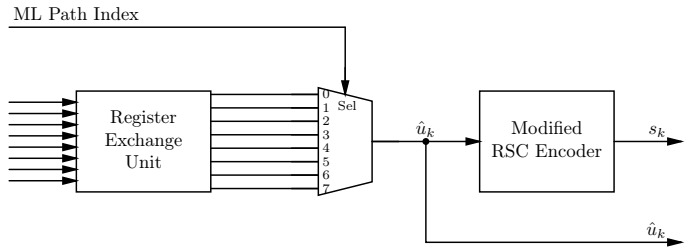


Figure 3.11: Block diagram of SMU module

The reliability updates are performed by the UPD module. The UPD module consists of U units of UPE elements that update and store the reliability values L^s_{k-L-j} at each stage of the decoding, based on the survivor and concurrent hard decision bit sequences. As reliability updates require survivor and concurrent path decision bits (u^c_{k-L-j} and u^s_{k-L-j}) for comparison before deciding if an update is needed, the PCU has additional logic to provide these *relevance bits*.

In order to obtain the survivor and concurrent paths, the SMU will first determine the ML state s_k by selecting the largest state metric Γ_i from the trellis unit as shown in Figure 3.11. Very often in practical designs, finding the largest state metric is not practical, and since it can be assumed that after L stages, the trellis has merged, any one of the N outputs of the register exchange unit can be used instead of the state chosen by the ML path index. The corresponding decision bit $\hat{u}_{k-L}$ is output from the RE unit, and the associated state at L stages away (end of SMU) s_{k-L} can be determined by using an encoder. The encoder used to determine s_{k-L} is illustrated in Figure 3.12.

With s_{k-L} and $\hat{u}_{k-L}$ determined, these two inputs are then used to select the desired rows from the RE in the PCU that correspond to the survivor and concur-

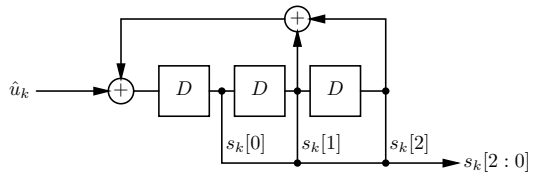


Figure 3.12: Encoder to determine state s_{k-L}

rent path sequences of decision bits. The selection is performed by letting $\hat{v}_{k-L}$ represent the complementary decision bit of $\hat{u}_{k-L}$. That is,

$$\begin{aligned} \hat{u}_{k-L} = 0 \quad &\Rightarrow \quad \hat{v}_{k-L} = 1 \\ \hat{u}_{k-L} = 1 \quad &\Rightarrow \quad \hat{v}_{k-L} = 0 \end{aligned}$$

The previous transition state of s_{k-L} will be

$$\begin{aligned} s^s_{k-L-1}(\hat{u}_{k-L}) &\xleftarrow{\hat{u}_{k-D}} s_{k-L} \\ s^c_{k-L-1}(\hat{v}_{k-L}) &\xleftarrow{\hat{v}_{k-L}} s_{k-L} \end{aligned}$$

and thus survivor and concurrent states s^s_{k-L-1} and a row select signal $\texttt{SC} = s^c_{k-L-1}$ can be obtained.

A block diagram of the PCU is shown in Figure 3.13. By means of 8-to-1 MUXes, the SC signal selects the row in the RE network that corresponds to the set of concurrent path bits $(u^c_{k-L-1}, u^c_{k-L-2}, \ldots, u^c_{k-L-U})$. The relevance bits are then determined by computing the XOR result of the decision bits of the survivor and the concurrent sequences. Thus, a relevance bit of '0' means that the survivor and concurrent bit at the stage are the same, i.e. $u^s_{k-L-j} = u^c_{k-L-j}$, and conversely a relevance bit of '1' implies that the bits are different $(u^s_{k-L-j} \neq u^c_{k-L-j})$.

In HR-SOVA, the update rule is only applied when the survivor and concurrent bits are different. Therefore the relevance bits generated in the PCU are used in the UPD to indicate if reliability updates are required for each of the U stages. The UPD module consists of U units of UPE, with each UPE element responsible for checking a relevance bit from the PCU and to decide if an update is required. If an update is required, the reliability value stored in the previous UPE stage is then compared against Δ^s_{k-L} that is selected from the FIFO Δ using s_{k-L}. The block diagram of the UPD is as shown in Figure 3.14.

3.2.5 Latency and Throughput

The data throughput of the presented HR-SOVA architecture depends directly on the data latency needed by the decoder to process a block of data. Given the merge

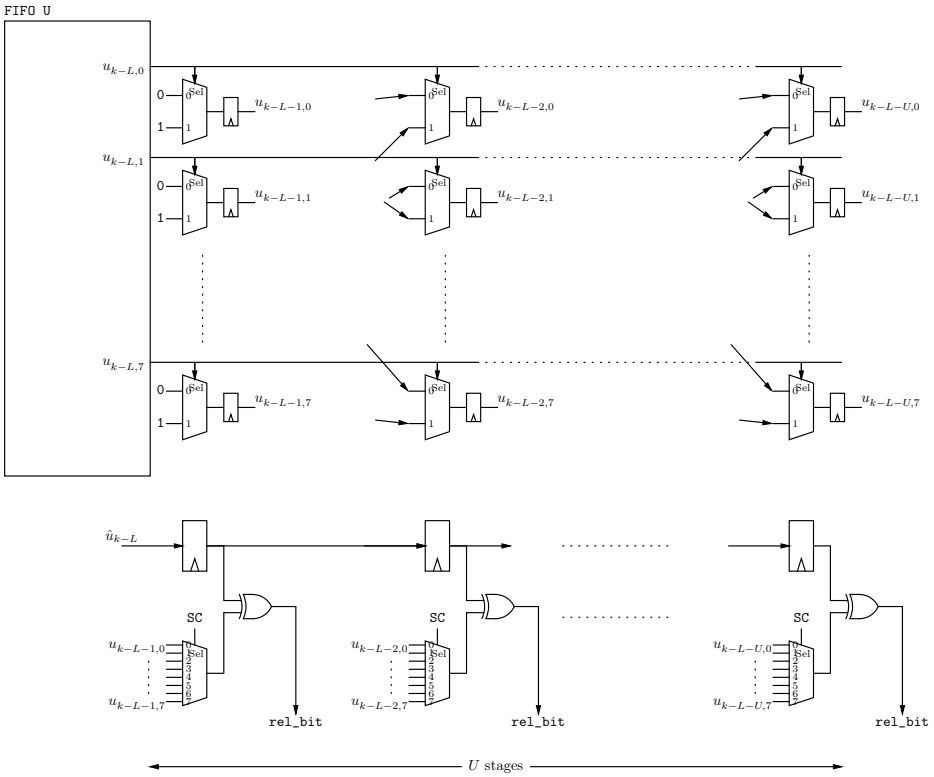


Figure 3.13: PCU for HR-SOVA

and decode depths L and U respectively, the latency due to the SMU and PCU for each block in a single iteration is expected to be approximately $L + U$.

For maximum throughput, the chip needs to be operated at the highest possible clock frequency. To achieve maximum the clock speeds, the critical path of the circuit needs to be minimized. For HR-SOVA, the critical path of decoder is expected to be at the trellis stage, due to ACS units. The block diagram of the ACS unit in Figure 3.8 shows that the critical path for the unit is approximately $T_{crit} = 2T_{add} + T_{mux}$, which is the minimal achievable critical path. To keep the critical path to the minimum, pipelines are inserted in the trellis stage, between the branch metric computation unit and the ACS unit, and at the output of ACS. Assuming a latency of total of 3 clock cycles for the additional pipelining in the decoder, the latency for the first output data is thus $3 + L + U$ clock cycles.

After the first data is output, the HR-SOVA based decoder is capable of producing one decoded data at every clock cycle. Therefore, the total time taken to process a K_i bit input block is $3 + L + U + K_i$ clock cycles.

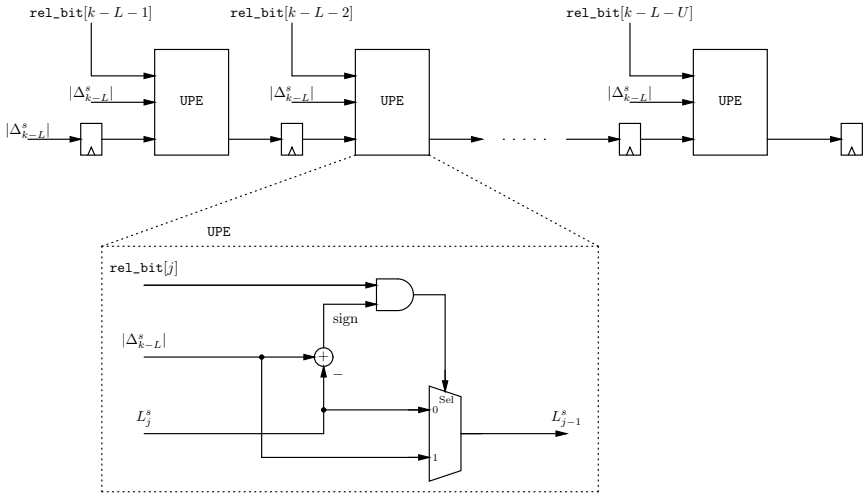


Figure 3.14: Block diagram of UPD module

3.3 Simplified BR-SOVA Architecture

The HR-SOVA hardware architecture may be modified to realize a hardware architecture for SB-SOVA that utilizes the same hardware stages, namely trellis, merge and decode.

Since the difference between HR-SOVA and SB-SOVA is in the reliability updates, the TRU and SMU from HR-SOVA can be used without modifications for SB-SOVA. In SB-SOVA, the Δ^c_{k-L-j} of the concurrent path needs to be taken into consideration when updating L^s_{k-L-j} and thus the RE units in the Battail rule PCU (BPCU) will have to perform register exchange for both the hard decision bit u_{k-L-j} and the metric difference Δ_{k-L-j}. As Δ_{k-L-j} has a longer wordlength than the hard decision bit u_{k-L-j}, the amount of silicon required to implement the BPCU module is expected to be much larger than an equivalent PCU module due to the larger MUXes and registers required to perform Register Exchange.

For SB-SOVA, the SOVA update rule is to be applied differently for

$$(u^s_{k-L-j} \neq u^c_{k-L-j}) \quad \text{and} \quad (u^s_{k-L-j} = u^c_{k-L-j})$$

and the additional SOVA update rule translates into additional hardware within the delta selection unit (DSU) module that calculates one of the two possible update values based on the relevance bits similar to those used in HR-SOVA. The DSU then passes the appropriate reliability update values to the Battail rule update (BUPD) module to perform the reliability value update. The top-level block diagram of the proposed SB-SOVA SISO decoder hardware architecture is shown

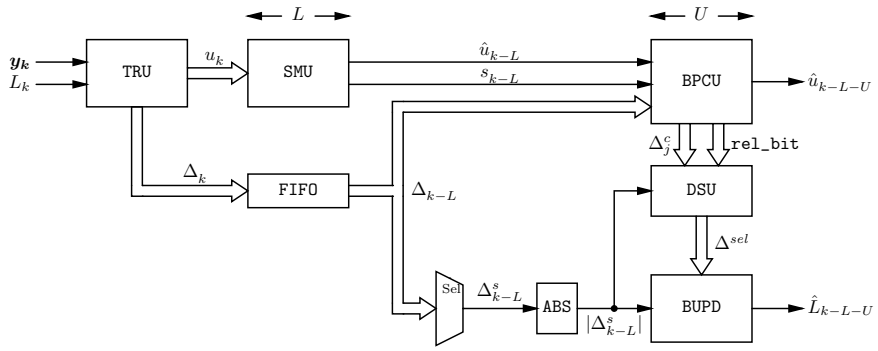


Figure 3.15: System architecture of SB-SOVA decoder

in Figure 3.15, and the various building blocks of SB-SOVA are described in detail in the paragraphs that follow.

The architecture of the trellis unit (`TRU`) used in the SB-SOVA architecture is similar to that for HR-SOVA. The `TRU` module determines the decision bit $u_{k,s}$ for each of the 8 states at every stage, and computes their corresponding metric differences $\Delta_{k,s}$. The literature for SOVA defines $\Delta_{k,s}$ as a positive number since it is the difference between the survivor and non-survivor path metric at a state. In this architecture, the value of $\Delta_{k,s}$ is instead calculated as follows.

$$\Delta_{k,s} = \Gamma(u_{k,s} = 0) - \Gamma(u_{k,s} = 1) \tag{3.6}$$

where Γ represents the metric computed at state s, for the selected decision bit $u_{k,s}$. Equation 3.6 may be re-stated as follows.

$$\begin{aligned} &\text{When} \quad \Gamma(u_{k,s} = 0) > \Gamma(u_{k,s} = 1), \\ &\text{Select} \quad u_{k,s} = 0 \\ &\quad \Rightarrow \quad \Delta_{k,s} > 0 \end{aligned}$$

$$\begin{aligned} &\text{When} \quad \Gamma(u_{k,s} = 1) > \Gamma(u_{k,s} = 0), \\ &\text{Select} \quad u_{k,s} = 1 \\ &\quad \Rightarrow \quad \Delta_{k,s} < 0 \end{aligned}$$

By representing $\Delta_{k,s}$ as a two's-complement signed number, the sign bit of $\Delta_{k,s}$ will represent the hard decision bit $u_{k,s}$. The main advantage of representing $\Delta_{k,s}$ in this form is that the set of hard decision bits, $u_{k,s}$ does not need to be stored separately since the information is already carried within $\Delta_{k,s}$. Performing RE on the metric differences will also perform RE on the hard decision bits at the same

time. A secondary advantage of this numeric representation is that it simplifies the `ACS` module, since there is no longer any need to perform the absolute operation to obtain a positive numbered $\Delta_{k,s}$.

In the original BR-SOVA algorithm, a concurrent reliability value L_j^c is to be updated before proceeding to update the survivor reliability value L_j^s. From the hardware point of view, this will mean that additional `PCU` and `UPD` hardware units will be needed to track the decision bits of the secondary concurrent paths in order to perform updates to L_j^c. The additional hardware will result in a significant increase in the hardware requirements for BR-SOVA as compared to HR-SOVA, which limits the practicality of the design. Therefore, only SB-SOVA will be considered in this thesis.

The SB-SOVA update rule from (2.5) is reproduced here for convenience

$$L_j^s \approx \begin{cases} \min\left\{\Delta_k^s, L_j^s\right\} & \text{when } u_k^s \neq u_k^c \\ \min\left\{\Delta_k^s + \Delta_j^c, L_j^s\right\} & \text{when } u_k^s = u_k^c \end{cases}$$

It can be seen that the SB-SOVA update rule requires the parameter Δ_j^c to be stored and this is handled by the `BPCU` module as shown in Figure 3.16. The `BPCU` performs RE on Δ_k instead of u_k as in the case of HR-SOVA, and the row containing the concurrent path will hold the values Δ_j^c. Since Δ_k is stored as a signed number with the sign bit representing the hard decision bit, there is no need for additional RE logic to handle the decision bits.

The metric differences of the concurrent path Δ_{k-L-j}^c needs to be selected and passed as an output to the `DSU` unit. As stated previously, the row containing the concurrent path will hold the required Δ_{k-L-j}^c values, and thus some row-select logic to calculate the correct row that contains the Δ_{k-L-j}^c values is needed. The concurrent row in the RE can be determined based on $\hat{u}_{k-L}$ and s_{k-L} inputs from the `SMU`. The concurrent state $\texttt{SC} = s_{k-L-1}^c$ can be determined as follows.

$$\begin{aligned} &\text{for } s_{k-L} = \{0, 2, 5, 6\}, \\ &\qquad \texttt{SC} = (\texttt{NOT}(\hat{u}_{k-L}) \ll 2) \parallel (s_{k-L} \gg 1) \\ &\text{for } s_{k-L} = \{1, 3, 4, 7\}, \\ &\qquad \texttt{SC} = (\hat{u}_{k-L} \ll 2) \parallel (s_{k-L} \gg 1) \end{aligned}$$

For the concurrent path, a η_Δ-bit 8-to-1 MUX is used to select the desired Δ_{k-L-j}^c from the 8 elements in their respective column of the `BPCU` RE network. The concurrent hard decision bits, u_{k-L-j}^c are obtained by using $\texttt{sign}(\Delta_{k-L-j}^c)$, and in a manner similar to HR-SOVA, the selected survivor and concurrent hard decision bit sequences are used to compute the relevance bit sequence `rel_bit` by `XOR` operations. These relevance bits `rel_bit`, and the set of U metric differences along the concurrent path $(\Delta_{k-L-1}^c, \Delta_{k-L-2}^c \ldots, \Delta_{k-L-U}^c)$ are passed to the `DSU` unit to compute the values that are to be used to compare against the set of U reliability values $(L_{k-L-1}^s, L_{k-L-2}^s \ldots, L_{k-L-U}^s)$.

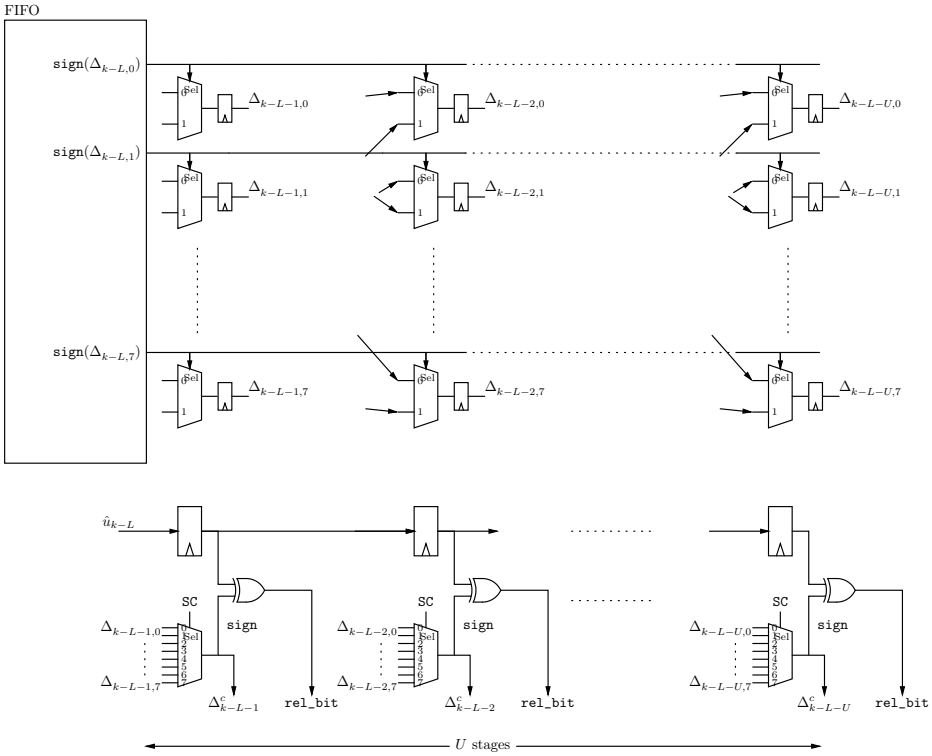


Figure 3.16: Block diagram of BPCU module

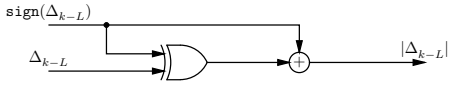


Figure 3.17: Block diagram of ABS module

A block (ABS) that calculates the absolute value a number in two's-complement representation [18] is shown in Figure 3.17. The ABS unit converts the Δ^s_{k-L} into $|\Delta^s_{k-L}|$ for use by the DSU and BUPD modules that require inputs of positive metric differences.

The DSU is responsible for selecting one of the two metric difference values that will be used to update the L^s_j corresponding to the two cases in (2.5). A block diagram of the DSU is shown in Figure 3.18. The selected metric difference value is passed on to BUPD as Δ^{sel}_j. The relevance bit is used here to obtain one of the two cases. Since $\Delta^{sel}_j = |\Delta^s_{k-L}|$ when the relevance bit is '1', and $\Delta^{sel}_j = |\Delta^s_{k-L}| + |\Delta^c_j|$

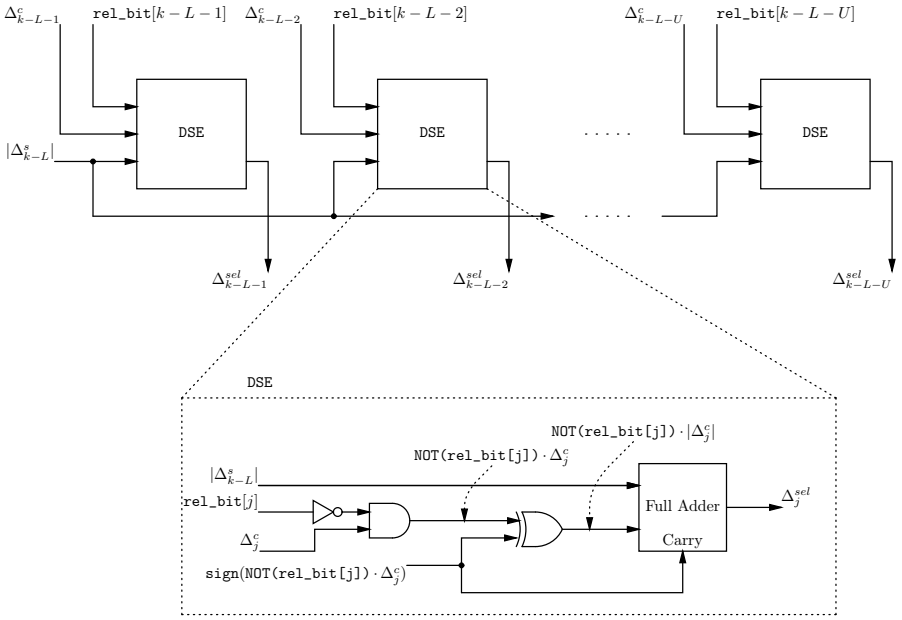


Figure 3.18: Block diagram of DSU module

when the relevance bit is '0', Δ_j^{sel} can be determined as follows.

$$\Delta_j^{sel} = |\Delta_{k-L}^s| + \texttt{NOT(rel\_bit[j])} \cdot |\Delta_j^c|$$

A conditional add/subtract unit [18] is used to perform the following operation when the reliability bit is '0',

$$\Delta_j^{sel} = \begin{cases} |\Delta_{k-L}^s| + \Delta_j^c, & \Delta_j^c \geq 0 \\ |\Delta_{k-L}^s| - \Delta_j^c, & \Delta_j^c < 0 \end{cases}$$

which eliminates the need for an absolute function on Δ_j^c before performing the addition. The circuit to calculate $|\Delta_{k-L}^s| + |\Delta_j^c|$ can be realized using a full adder and XOR gates as shown in Figure 3.18.

The set of U values of Δ_j^{sel} from the DSU unit is then passed to BUPD to perform the reliability updates. The BUPD module is made up of U units of BUPE elements and has a block diagram as shown in Figure 3.19. Each BUPE module performs a SOVA reliability update by doing a compare and select to obtain the minimum of the Δ_j^{sel} and L_j^s. The main difference between the UPD used in HR-SOVA and the BUPD used in SB-SOVA is that the SB-SOVA reliability updates are independent of the relevance bits.

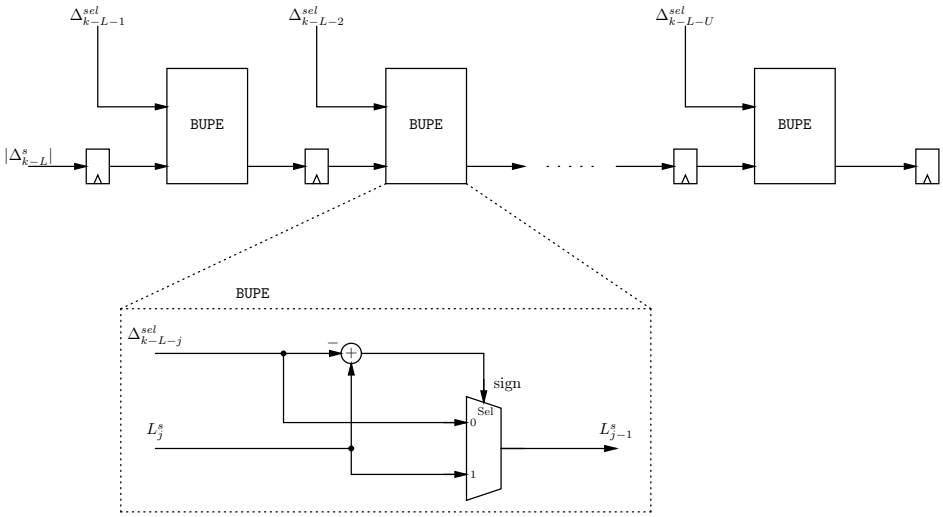


Figure 3.19: Block diagram of BUPD module

3.3.1 Resource Utilization

The hardware resource utilization for the SB-SOVA architecture mainly involves the various adders and multiplexers used for the SMU and BPCU units. For these units, each of the RE unit can easily be translated into a 2-to-1 multiplexer. For HR-SOVA, both SMU and PCU involve only 1-bit MUXes for u_k. For the BPCU module, the RE unit is performing register exchange for Δ which means that each RE will consist of a η_Δ-bit MUX that can for estimation purposes be considered to be η_Δ times larger than a 1-bit MUX.

The additional DSU unit for SB-SOVA will increase the number of adders and XOR/AND logic gates used. The resource utilization for SB-SOVA and HR-SOVA is presented in Table 3.2. As observed from the table, SB-SOVA is expected to have significantly higher hardware utilization than HR-SOVA. To make a simple estimation, consider the following case

$$N = 8, \quad L = 24, \quad U = 24, \quad \eta_\Delta = 10$$

Assume that an 8-to-1 MUX is equivalent to five 2-to-1 MUXes, and a stage of n-bit adder is equivalent to a stage of n-bit MUX, the hardware resource utilization can be estimated by expressing it in terms of number of units of 1-bit adder/MUX. The resource utilization for SB-SOVA and HR-SOVA are shown in Table 3.3. Since the XOR and AND gates utilize less resources than MUXes and adders, they are omitted to simplify the comparison.

Table 3.2: Comparison of resource utilization for SB-SOVA

	HR-SOVA	SB-SOVA
`TRU`		
Adders	$3N$ (η_Δ bit)	$3N$ (η_Δ bit)
MUXes (2-to-1)	N (η_Δ bit)	N (η_Δ bit)
`SMU`		
Adders	-	-
MUXes (2-to-1)	NL (1 bit)	NL (1 bit)
`PCU`/`BPCU`		
Adders	-	-
MUXes (2-to-1)	NU (1 bit)	NU (η_Δ bit)
MUXes (8-to-1)	U (1 bit)	U (η_Δ bit)
`UPD`/`BUPD`		
Adders	U (η_Δ bit)	$1+U$ (η_Δ bit)
MUXes (2-to-1)	U (η_Δ bit)	U (η_Δ bit)
`XOR` gates	-	η_Δ
`DSU`		
Adders	-	U (η_Δ bit)
`XOR`/`AND` gates	-	$2U \cdot \eta_\Delta$
Total Resource Usage		
Adders	$3N+U$ (η_Δ bit)	$3N+2U+1$ (η_Δ bit)
MUXes (2-to-1)	$N+U$ (η_Δ bit) $NL+NU$ (1 bit)	$N+U+NU$ (η_Δ bit) NL (1 bit)
MUXes (8-to-1)	U (1 bit)	U (η_Δ bit)
`XOR`/`AND` gates	-	$(2U+1) \cdot \eta_\Delta$

3.3.2 Memory and Register Utilization

One FIFO is required in parallel to the merge unit (`SMU`) to store the N sets of Δ_k for each stage, for the entire depth of the SMU, i.e. L stages. Therefore, the total FIFO requirement in the merge stage will be $L \cdot N \cdot \eta_\Delta$ bit, where η_Δ denotes the wordlength of Δ.

The total memory requirements (including registers in merge/updating units and FIFO) for SB-SOVA and HR-SOVA are presented in Table 3.4. Simulation

Table 3.3: Numerical example for resource utilization for HR-SOVA and SB-SOVA

	HR-SOVA	SB-SOVA
Adders	48 (10 bit)	73 (10 bit)
MUXes (2-to-1)	32 (10 bit) 384 (1 bit)	224 (10 bit) 192 (1 bit)
MUXes (8-to-1) (×5)	24 (1 bit)	24 (10 bit)
Total count (Adders/MUX)	1 304	4 362

Table 3.4: Comparison of memory requirements for SB-SOVA

	HR-SOVA	SB-SOVA
`FIFO` Δ	$LN \cdot \eta_\Delta$	$LN \cdot \eta_\Delta$
`SMU`	LN	LN
`PCU`/`BPCU`	$U + UN$	$U + UN \cdot \eta_\Delta$
`UPD`/`BUPD` & `DSU`	$U \cdot \eta_\Delta$	$2U \cdot \eta_\Delta$
Total	$U + (L+U)N +$ $(LN + U)\eta_\Delta$	$U + LN +$ $(LN + UN + 2U)\eta_\Delta$
Example $N = 8, L = 24, U = 24, \eta_\Delta = 10$	2 568 bit $\approx$ 2.5 kbit	4 536 bit $\approx$ 4.5 kbit

results in Section 2.2.5 show that $L = 24$ and $U = 24$ are sufficient to give comparable results to Max-Log-MAP. Assuming that $N = 8$ and $\eta_\Delta = 10$ bit, the total memory requirement in this case will be 3.8 kbit and 5.8 kbit for HR-SOVA and SB-SOVA respectively. It is useful to note that the memory requirement for SB-SOVA is approximately twice that of HR-SOVA. However, the storage requirement for SB-SOVA is still much lower than that of Max-Log-MAP, which was calculated previously to be 48 kbit.

3.3.3 Latency and Throughput

The critical path for the SB-SOVA architecture has moved from the `ACS` module in HR-SOVA to the signal flow from `BPCU` to the `BUPD`. The critical path is estimated to be $4T_{mux} + 2T_{add}$, which is a significant increase from $2T_{add} + T_{mux}$ in HR-SOVA. Left unchecked, this will have an impact on the overall data throughput. To reduce the critical path, additional pipelining is required. By adding two sets of pipeline

of registers, one between BPCU and DSU, and the other between DSU and BUPD, the combinatorial delay in the reliability update modules is reduced to $T_{mux} + T_{add}$. In this case, the T_{crit} is restored to the same value value as that in HR-SOVA, with the critical path once again in the ACS unit.

The latency for HR-SOVA and SB-SOVA is expected to be the largely unchanged. Both decoders will takes L cycles to determine the merged ML state and a further U cycles to obtain the first reliability output value $L_e(\hat{u}_k)$. Considering a 3-stage pipeline at the trellis unit and 2-stage pipeline between BPCU and BUPD to reduce the critical path, the total latency for SB-SOVA will be $L + U + 5$ as compared with $L+U+3$ for HR-SOVA. With $L = 24$ and $U = 24$, the total latency for the SB-SOVA decoder will be

$$\text{latency} = L + U + 5 = 53 \,\text{clock cycles}$$

Assuming that the block length is 6 144 bit and is divided into 8 parallel windows, this gives a window size of $W = 768$. As described in section 3.6, each window will require a warm-up time of $\alpha = 32$ cycles. The total number of clock cycles for each simplified BR-SOVA decoder to process a data block will be

$$W + L + U + 5 + \alpha = 853 \,\text{clock cycles}$$

As a comparison, a set of Max-Log-MAP SISO decoders working with 8 parallel windows will take $2W = 1\,536$ cycles to process a data with block length of $K_i = 6\,144$ and $W = 768$. This shows that the latency of SB-SOVA is lower than that of Max-Log-MAP. It can also be concluded that a SB-SOVA based turbo decoder is expected to achieve a higher throughput than one using Max-Log-MAP based decoders.

3.4 Hybrid-SOVA Architecture

The hardware resource utilization for SB-SOVA as presented in Table 3.2 showed that the resource utilization of SB-SOVA is a few times higher than that for HR-SOVA. This is highly undesirable as the die area for a SB-SOVA based accelerator will be much larger than one that incorporates HR-SOVA. In order to reduce the amount of hardware resources required by the SB-SOVA architecture, the following alternative is proposed.

Consider an architecture with identical TRU and SMU units as in SB-SOVA. Instead of performing SB-SOVA for all U stages, the SOVA reliability updates are divided into two stages, with U_1 stages of updates using SB-SOVA rules followed by U_2 stages of updates using HR-SOVA rules. The modifications to the reliability update modules are described as follows.

For the BPCU, instead of containing U columns of RE units for Δ_k, the number of columns is reduced to U_1, where $U_1 < U$. The remaining $(U_2 = U - U_1)$ columns of RE in the PCU will only be register exchange for the decision bits u_k. Hence, the

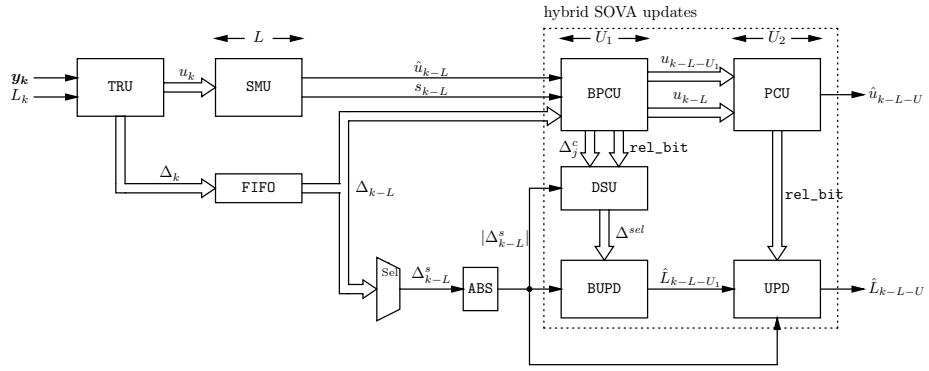


Figure 3.20: System architecture of hybrid-SOVA

PCU for hybrid-SOVA may be viewed as operating in `SB-SOVA mode` in the first U_1 stages, followed by `HR-SOVA mode` in the last U_2 stages.

The `DSU` is also modified to provide only U_1 values of Δ_j^c to the `BUPD`. The update unit is also modified to consist of U_1 deep `BUPD` and U_2 deep `UPE`.

The architecture of the proposed hybrid-SOVA SISO decoder is presented in Figure 3.20. The hybrid-SOVA updates with modifications described in the paragraphs above are indicated in the dotted box. Between the `BPCU` and `PCU` are the hard decision bits at the exit of the `BPCU` RE units that will continue the RE process in the `PCU`, and decision bits from `FIFO` (i.e. $\texttt{sign}(\Delta_{k-L})$) that will be used to perform row exchange.

The reliability values that are updated U_1 times using SB-SOVA updating rules in `BUPD` will be passed to the `UPD` module for further U_2 updates using the HR-SOVA updating rules. Other than the differences highlighted above, all other signals and modules within the hybrid-SOVA block serves the same functionality as they did in the SB-SOVA or HR-SOVA architectures.

To determine the performance of hybrid-SOVA, simulations are performed with $U = 24$ for different U_1 values. The environment for the simulation is given in Table 3.5 and the BER/BLER performance for hybrid-SOVA are plotted in Figure 3.21.

As observed from the plots, most U_1 values are able to obtain comparable BLER performance with degradation of less than 0.1 dB as compared to SB-SOVA. The performance of hybrid-SOVA with $U_1 = 12$ is sufficiently close to that of SB-SOVA. For the same number of iterations, the performance of hybrid-SOVA is approximately 0.15 dB worse off than Max-Log-MAP.

Table 3.5: Simulation conditions for hybrid-SOVA

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit

Simulation Parameters	
Algorithm:	hybrid-SOVA
Update method:	Hardware
	$(L, U) = (24, 24)$
	$(c, d) = (0.75, 1.00)$
	$\Delta_{TH} = \begin{cases} 8 & \text{SNR} \leq 0.7 \text{ dB} \\ 16 & \text{SNR} > 0.7 \text{ dB} \end{cases}$

3.4.1 Resource Utilization

Since the motivation of hybrid-SOVA is to reduce the hardware complexity of SB-SOVA, the hardware resource utilization for hybrid-SOVA is tabulated. Taking the case of $U_1 = U_2 = 12$, the main savings is expected to be at the `BPCU` unit where the number of η_Δ-bit MUXes is reduced by half. The resource utilization for hybrid-SOVA and SB-SOVA is presented in Table 3.6.

As observed from Table 3.6, hybrid-SOVA requires fewer MUXes than SB-SOVA. In order to make a better estimate, consider the following numerical example based on similar assumptions to those in Section 3.3.

$$N = 8, \quad L = 24, \quad U = 24, \quad U_1 = 12, \quad \eta_\Delta = 10$$

The estimated resource utilization is as shown in Table 3.7 and the savings made by hybrid-SOVA is estimated to be approximately 35 %. This illustrates the advantage of using hybrid-SOVA in place of SB-SOVA to achieve performance that is comparable to using a Max-Log-MAP decoder.

3.4.2 Memory and Register Utilization

The memory and register utilization of the hybrid-SOVA architecture is computed and compared against SB-SOVA. The result of this comparison is tabulated in Table 3.8.

The numerical example illustrated in Table 3.8 shows that the memory usage for hybrid-SOVA is 4.8 kbit, which is 984 bit less than that of SB-SOVA. The savings in memory usage for hybrid-SOVA as compared to SB-SOVA is approximately 16 %.

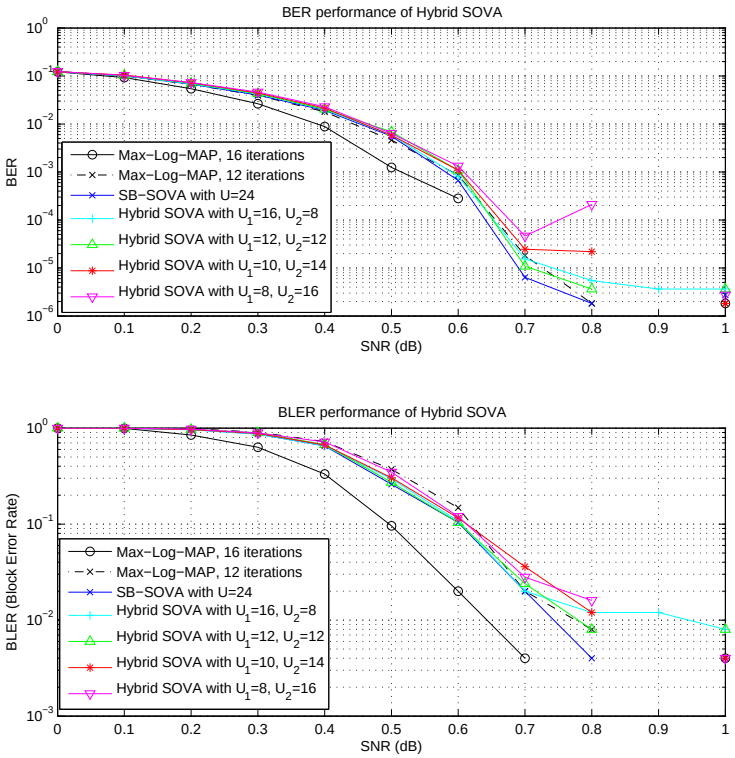


Figure 3.21: BER performance of hybrid-SOVA

3.4.3 Latency and Throughput

Since hybrid-SOVA can be seen as a combination of SB-SOVA and HR-SOVA, the analysis of the critical path in Section 3.3.3 can be used to analyze the hybrid-SOVA architecture. The critical path of the hybrid-SOVA architecture is $T_{crit} = 2T_{add} + T_{mux}$ and lies within the `ACS` module.

As the number of stages in the architecture of hybrid-SOVA remains the same as that of SB-SOVA, its latency and data throughput are expected to be the same as that of SB-SOVA. By considering a practical implementation of hybrid-SOVA in 8 parallel windows, the overall latency will be

$$W + L + U + 5 + \alpha = 853 \text{ clock cycles}$$

where $W = 768$, $L = 24$, $U = 24$ and $\alpha = 32$.

Table 3.6: Comparison of resource utilization for SB-SOVA and hybrid-SOVA

	SB-SOVA	hybrid-SOVA
`TRU`		
Adders	$3N$ (η_Δ bit)	$3N$ (η_Δ bit)
MUXes (2-to-1)	N (η_Δ bit)	N (η_Δ bit)
`SMU`		
Adders	-	-
MUXes (2-to-1)	$N \cdot L$ (1 bit)	$N \cdot L$ (1 bit)
`BPCU`/`BPCU` & `PCU`		
Adders	-	-
MUXes (2-to-1)	$N \cdot U$ (η_Δ bit)	$N \cdot U_1$ (η_Δ bit) $N \cdot (U - U_1)$ (1 bit)
MUXes (8-to-1)	U (η_Δ bit)	U_1 (η_Δ bit) $(U - U_1)$ (1 bit)
`BUPD`/`BUPD` & `UPD`		
Adders	$1 + U$ (η_Δ bit)	$1 + U$ (η_Δ bit)
MUXes (2-to-1)	U (η_Δ bit)	U (η_Δ bit)
`XOR`/`AND` gates	η_Δ	η_Δ
DSU		
Adders	U (η_Δ bit)	U_1 (η_Δ bit)
`XOR`/`AND` gates	$2U \cdot \eta_\Delta$	$2U_1 \cdot \eta_\Delta$
Total Resource Usage		
Adders	$3N + 2U + 1$ (η_Δ bit)	$3N + U + U_1 + 1$ (η_Δ bit)
MUXes (2-to-1)	$N + U + NU$ (η_Δ bit) NL (1 bit)	$N + U + NU_1$ (η_Δ bit) $NL + N(U - U_1)$ (1 bit)
MUXes (8-to-1)	U (η_Δ bit)	U_1 (η_Δ bit) $U - U_1$ (1 bit)
`XOR`/`AND` gates	$(2U + 1) \cdot \eta_\Delta$	$(2U_1 + 1) \cdot \eta_\Delta$

3.5 Improved BPCU for SB-SOVA

As observed in the hardware architectures for SB-SOVA and hybrid-SOVA, the use of register exchange on Δ_k in SB-SOVA and hybrid-SOVA `BPCU` results in large number of η_Δ-bit 2-to-1 MUXes being used. Given that the `BPCU` is U-stages deep

Table 3.7: Numerical example for resource utilization of hybrid-SOVA

	SB-SOVA	hybrid-SOVA
Adders	73 (10 bit)	61 (10bit)
MUXes (2-to-1)	224 (10 bit) 192 (1 bit)	128 (10 bit) 288 (1 bit)
MUXes (8-to-1) (×5)	24 (10 bit)	12 (10 bit) 12 (1 bit)
Total count (Adders/MUXes)	4 362	2 838 (≈ 35% savings)

Table 3.8: Memory requirements for hybrid-SOVA and SB-SOVA

	SB-SOVA	hybrid-SOVA
`FIFO` Δ	$LN \cdot \eta_\Delta$	$LN \cdot \eta_\Delta$
`SMU`	LN	LN
`BPCU`/`BPCU` & `PCU`	$U + UN \cdot \eta_\Delta$	$U + U_1 N \cdot \eta_\Delta + (U - U_1)N$
`BUPD`/`BUPD` & `UPD`	$2U \cdot \eta_\Delta$	$U \cdot \eta_\Delta + U_1 \cdot \eta_\Delta$
Total	$U + LN +$ $(LN + UN + 2U)\eta_\Delta$	$U + (L + U - U_1)N +$ $(LN + U_1 N + U + U_1)\eta_\Delta$
Example: $N = 8, L = 24$ $U = 24, U_1 = 12, \eta_\Delta = 10$	4 536 bit ≈ 4.5 kbit	3 552 bit ≈ 3.5 kbit

(or U_1 for hybrid-SOVA) with 8 states per stage and $\eta_\Delta = 10$, the number of 2-to-1 MUXes used will be

$$8 \times U \times 10 = 80U$$

which implies that for SB-SOVA with $U = 24$, there will be 1 920 MUXes in the PCU. Such a large number of MUXes will translate into higher hardware resource utilization for SB-SOVA/hybrid-SOVA implementations and indicates that the `BPCU` is a good target for optimization. By reducing the number of MUXes in the `BPCU`, the hardware resource requirements for implementing SB-SOVA and hybrid-SOVA can be made more manageable.

Instead of performing register exchange on the Δ_k, the SOVA update unit can be modified to perform register exchange on the state s_k. Upon row selection, the state sequence for the concurrent path s_k^c is selected from the s_k registers. The concurrent state s_k^c selected for each stage is then used to select Δ_k^c from the column of registers in each stage.

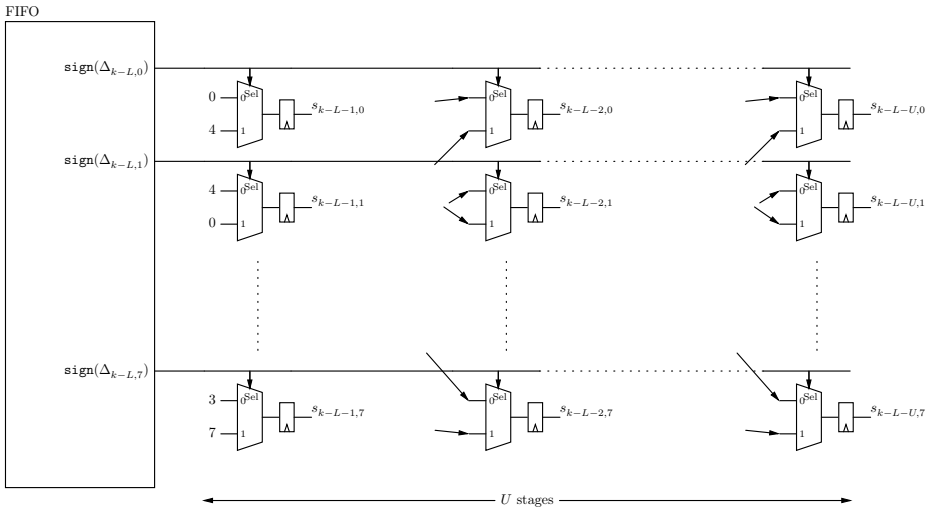


Figure 3.22: Register exchange for s_k

Figure 3.22 shows the register exchange of s_k in the BPCU module. Since there are 8 states and each s_k has a wordlength of 3 bit, each register and MUX in the figure has a wordlength of 3 bit. The concurrent state sequence s_k^c is selected using U units of 8-to-1 3-bit MUXes. The selected concurrent state sequence is then used to select the concurrent metric differences Δ_k^c from the column of 8 registers that are stored as shown in Figure 3.23a.

The survivor decision output bits from the stored sequence of survivor decision bits and the concurrent state sequence in Figure 3.22 are then used to obtain the relevance bits as shown in Figure 3.23b, similar to the way it is done in the normal BPCU.

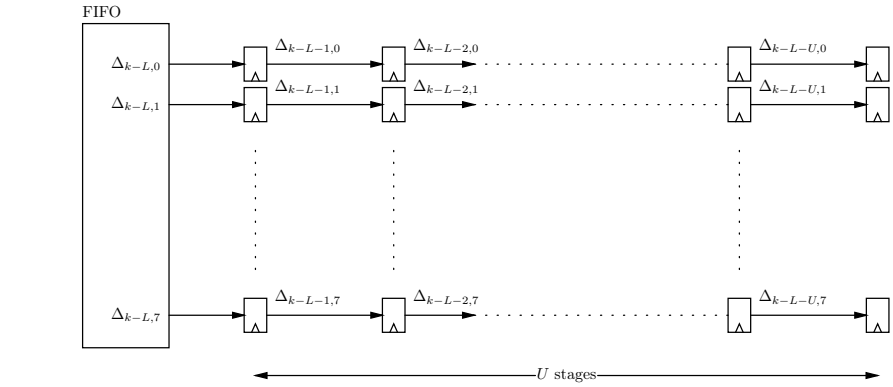
3.5.1 Resource Utilization

The objective of the improved BPCU for SB-SOVA and hybrid-SOVA is to reduce the wordlengths of MUXes used in Register Exchange (RE). A clear advantage of using this configuration for register exchange is that with reduced wordlengths in the MUXes, there is clearly less wiring between the components within the BPCU.

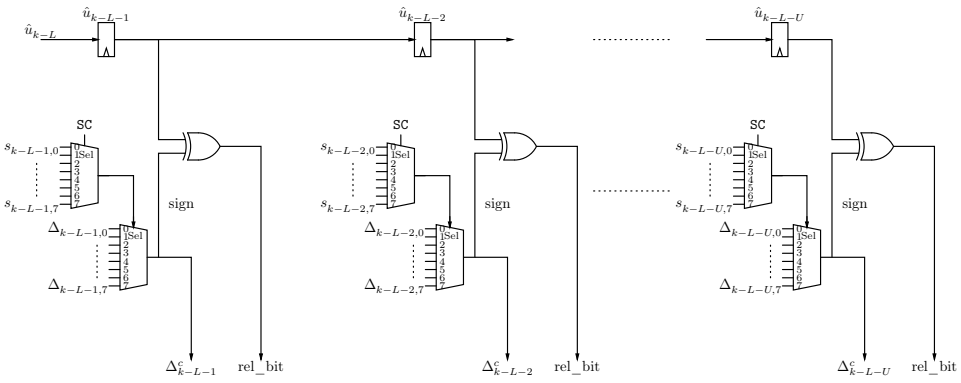
The savings in the area consumed by the MUXes can be estimated as follows. Since the register exchange is performed on s_k, the width of each MUX will be $\nu = 3$ bit. Assuming $\eta_\Delta = 10$ bit, the savings in the number of 1-bit MUXes will be

$$(\eta_\Delta - \nu) \cdot N \cdot U = 56U$$

Comparing Figure 3.23b with Figure 3.16 shows that the improved PCU requires



(a) Register storage for Δ_k



(b) Obtaining the relevance bits and Δ^c in PCU

Figure 3.23: Hardware implementation of improved `BPCU`

an additional 8-to-1 3-bit MUX at each stage to select the concurrent path state. This is equivalent to having additional $3U$ units of 8-to-1 single bit MUX. If an 8-to-1 MUX is assumed to have 5 times the complexity of a 2-to-1 MUX, the total reduction in MUXes becomes

$$\begin{aligned} 56U - 5(3U) &= 41U \\ &= 984 \quad \text{when } U = 24. \end{aligned}$$

The comparison of resource utilization for the improved `BPCU` in SB-SOVA and hybrid-SOVA is as shown in Table 3.9.

As observed from Table 3.9, the modified `BPCU` has resulted in hardware savings of approximately 22 % and 17 % for SB-SOVA and hybrid-SOVA respectively.

Table 3.9: Resource utilization with improved `BPCU` in SB-SOVA and hybrid-SOVA

SB-SOVA		
`PCU`	(Δ_k exchange)	(s_k exchange)
MUXes (2-to-1)	$N \cdot U$ (η_Δ bit)	$N \cdot U$ (ν bit)
MUXes (8-to-1) ($\times 5$)	U (η_Δ bit) U (ν bit)	U (η_Δ bit) U (ν bit)
Total count: $N = 8$, $U = 24, \eta_\Delta = 10$	4 362	3 378 (22% savings)
hybrid-SOVA		
`BPCU`	(Δ_k exchange)	(s_k exchange)
MUXes (2-to-1)	$N \cdot U_1$ (η_Δ bit) $N \cdot (U - U_1)$ (1 bit)	$N \cdot U_1$ (ν bit) $N \cdot (U - U_1)$ (1 bit)
MUXes (8-to-1) ($\times 5$)	U_1 (η_Δ bit) $(U - U_1)$ (1 bit)	U_1 (η_Δ bit) U_1 (ν bit) $(U - U_1)$ (1 bit)
Total count: $U_1 = 12$ (as above)	2 838	2 346 (17% savings)

3.5.2 Memory requirements

As observed in Figure 3.23a, the content of Δ_k from the FIFO are still being stored in η_Δ-bit registers. The requirement for these registers will thus remain unchanged. Besides these registers for Δ, extra $N \cdot U$ $(\nu + 1)$-bit registers are required for the exchange of $\boldsymbol{s_k}$ and $\boldsymbol{u_k}$ respectively. For $U = 24$, the number of extra registers will be 768, which is approximately the same amount of reduction as in the case of the MUXes. Thus it can be seen that the savings in MUXes has been translated into extra registers. It is thus a trade-off between the usage of MUXes and registers. The new memory requirement for the improved `BPCU` is as shown in Table 3.10.

3.6 Considerations for Parallel Windows

When the channel input to a SOVA SISO decoder is processed serially, the amount of time needed to process the entire block is dependent on the block length of the data. Denoting the block length by K_i, the total time taken for each iteration of SOVA decoder is at least $3 + L + U + K_i$. The block length K_i for LTE ranges from 40 to 6 144. When K_i is large, the computation time increases and the overall throughput is reduced.

Table 3.10: Memory requirements with improved BPCU for SB-SOVA and hybrid-SOVA

SB-SOVA		
	(Δ_k exchange)	(s_k exchange)
FIFO Δ	$LN \cdot \eta_\Delta$	
SMU	LN	
BPCU	$U + UN \cdot \eta_\Delta$	$U + UN \cdot (\eta_\Delta + \nu)$
BUPD & DSU	$2U \cdot \eta_\Delta$	
Total	$U + LN +$ $(LN + UN + 2U)\eta_\Delta$	$U + LN + UN \cdot \nu$ $+(LN + UN + 2U)\eta_\Delta$
Example: $N = 8$, $L = 24$ $U = 24$, $\eta_\Delta = 10$	4 536 bit ≈ 4.5 kbit	5 112 bit ≈ 5 kbit (13 % more memory)

hybrid-SOVA		
	(Δ_k exchange)	(s_k exchange)
FIFO Δ	$LN \cdot \eta_\Delta$	
SMU	LN	
BPCU & PCU	$U + U_1 N \cdot \eta_\Delta +$ $(U - U_1)N$	$U + U_1 N \cdot \eta_\Delta +$ $(U - U_1)N + U_1 N \cdot \nu$
BUPD, DSU & UPD	$(U + U_1) \cdot \eta_\Delta$	
Total	$U + (L + U - U_1)N +$ $(LN + U_1 \cdot N + U + U_1)\eta_\Delta$	$U + (L + U - U_1)N +$ $(LN + U_1 N + U + U_1)\eta_\Delta$ $+(U_1 N)\nu$
Example: $N = 8$, $L = 24$ $U = 24$, $U_1 = 12$, $\eta_\Delta = 10$	3 552 bit ≈ 3.5 kbit	3 840 bit ≈ 3.75 kbit (8 % more memory)

To ensure that the total processing time is manageable, the channel input data can be split into M windows and to use M instances of SISO decoders operating concurrently on the data. The trade-off to obtain higher throughout is the increase in hardware resources, as the same SISO decoder hardware will need to be duplicated M times. An example of a windowed SOVA implementation is described in [19].

The following sections discuss the considerations that need to be taken into account when implementing parallel windows for a SOVA based turbo decoder. Section 3.6.1 describes a possible banked memory organization for M decoders and Section 3.6.2 discusses methods in which the initial state of a each window can be estimated. Finally, Section 3.6.3 describes the overall memory access requirements for a SOVA decoder and shows that the banked memory organization is suitable

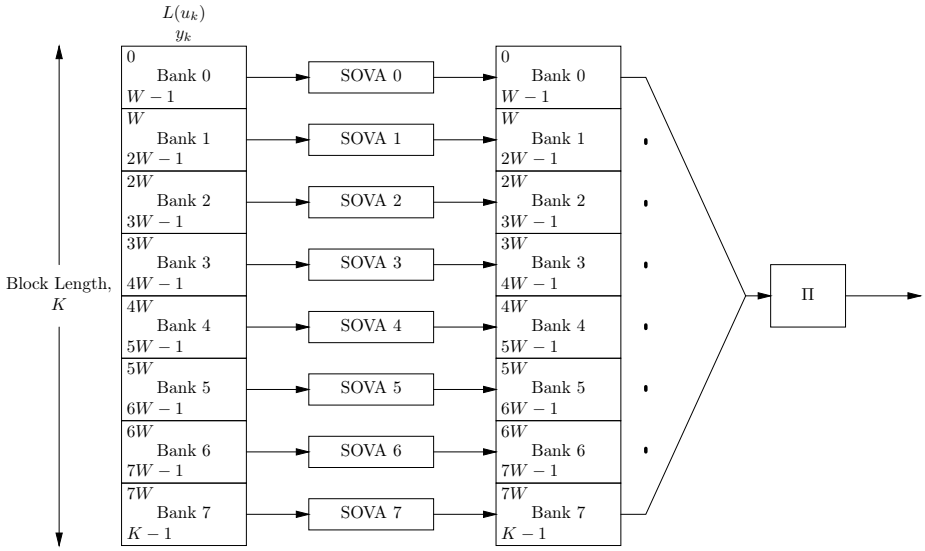


Figure 3.24: Dividing an input block into 8 windows

for use in a multiple windowed SOVA based turbo decoder.

3.6.1 Memory Organization for Multiple Windows

One important consideration when designing a high speed parallelized turbo decoder is to ensure that the memory access of the input by all parallel decoders remains contention free, i.e. the decoder can always the access the data that it needs without delay. To meet this requirement, the inputs to each of the decoders, that is the channel input $\boldsymbol{y_k}$ and intrinsic data $L(u_k)$, have to be divided into *memory banks* where at any one time, each memory bank is only accessed by one decoder. Consider a channel data block that is divided into 8 banks, each of size W. The input to each SOVA RSC decoder is then of size W. By dividing the input data block into 8 banks, contention free memory access can be achieved, as shown in Figure 3.24.

3.6.2 Estimation of Window Initial State

One of the properties of the turbo code used in LTE is that the starting and terminating states are known a-priori to start and end with state 0. When turbo decoding is performed in parallel windows, the starting state for second window onwards is unknown. For instance, the starting state of window i is expected to be the end state of window $i-1$. However, since both windows are running concurrently, the end state of window $i-1$ is not known until decoding of the entire

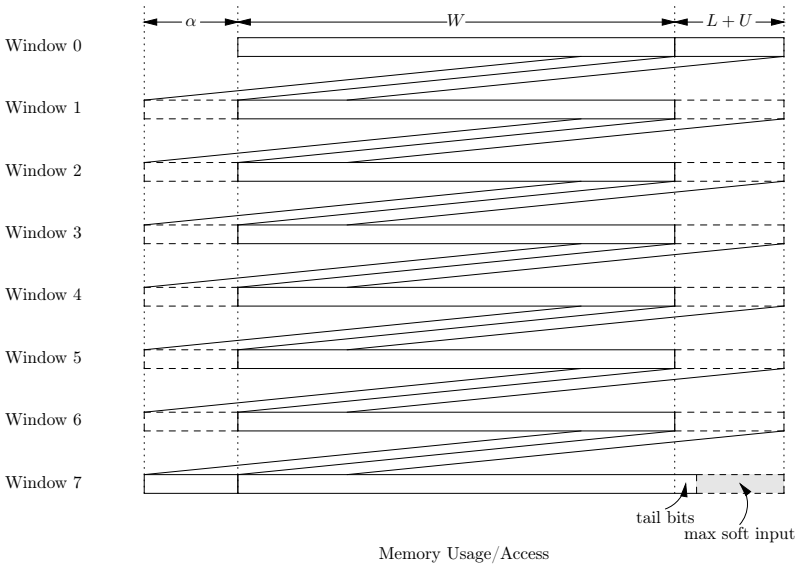


Figure 3.25: Data access requirements for each SOVA window

window is completed. Therefore, for successful parallelization using windows, the starting state of each window needs to be estimated before decoding of the data in each window can begin.

There are two methods that can be used to estimate the initial start state, namely using the VA to perform state metric accumulation and by using state information from the previous iteration. For the metric accumulation method, the VA is used to accumulate the path metric for α received symbols prior to the start of the window. After accumulating α state metrics, the state with the best path metric at the start of the window is an estimate of the starting state for the window. By using the entire set of 2^{ν} path metrics as the initial metrics to begin decoding, it is equivalent to skewing the metrics within the window to start from the estimated start state. Therefore one can look at the initial α stages as a *warm up* phase for every window before the start of actual decoding. All windows with the exception of the first one will perform a warm-up path metric accumulation for α received symbols prior to the first symbol in the window. Figure 3.25 shows the received symbols processed by each of the windows to perform warm-up.

To determine the effect of α-stage warm-up method described earlier, simulations were performed to determine the optimum α required for comparable results as Max-Log-MAP, with the conditions tabulated in Table 3.11.

The BER and BLER performance for SB-SOVA with different α values is presented in Figure 3.26. As observed from the plots, parallel SB-SOVA with $\alpha = 40$ and $\alpha = 32$ have comparable BLER performance as that of Max-Log-MAP with

Table 3.11: Simulation conditions for α-stage warm-up

Environment:	AWGN channel
Block length:	4 416
Number of blocks:	250
Total data size:	$\approx 10^6$ bit
Simulation Parameters	
Algorithm:	SB-SOVA
Update method:	Hardware
Parallel Windows:	8
	$(L, U) = (24, 24)$
	$(c, d) = (0.75, 1.00)$
	$\Delta_{TH} = 8$
	$\alpha = \{20, 24, 32, 40\}$

12 iterations. For the same number of iterations, parallel SB-SOVA with $\alpha = 40$ and $\alpha = 32$ are at most 0.2 dB worse than Max-Log-MAP. Considering the fact that the comparison is made with an ideal non-windowed implementation of Max-Log-MAP, the performance of the windowed SB-SOVA is desirable. As a higher α implies a longer latency, $\alpha = 32$ can be viewed as the optimum value to be used for parallel windowed SOVA decoder. Based on these results, it can be concluded that the proposed α-stage metric accumulation method is a feasible method for initial state estimation in parallel windowed SOVA architectures.

The second method to estimate the start state of the window is to use data from the previous iteration, and is similar to the next iteration initialize (NII) method used in parallel window MAP decoders [20]. When applied to SOVA SISO decoders, the state metric vector for the last stage of the previous window in the previous iteration will be used as the initial state metric vector for the current window. The main advantage of this is method is that warm up is needed only for the first iteration which leads to reduced latency for the turbo decoder.

The effect of NII method for SOVA is observed with simulations. The first n_1 half iterations are run using α-stage metric accumulation method, while the remaining n_2 decoding iterations are run using the state metric vector of the previous window. The value of n_1 is varied between 2 to 14 and results are as shown in Figure 3.27. As observed from the result plots, the proposed NII method has slightly degraded performance as compared to the metric accumulation method. The optimum settings for the NII method is with $n_1 = 4$ half iterations of α-stage warm-up, but it suffers 0.1 dB degradation performance as compared to the α-stage

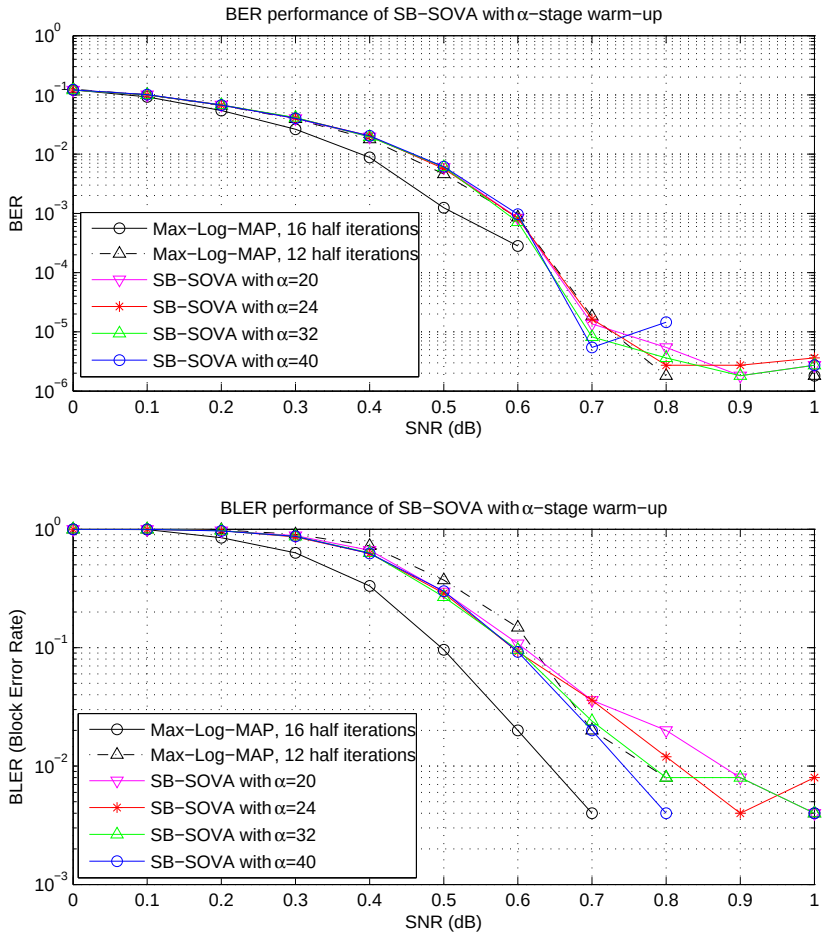


Figure 3.26: BER and BLER performance for SB-SOVA with varying α-warm-up

metric accumulation method.

The two methods described above are suitable for parallel SISO decoding. The preferred method to use will depend on the trade-off between data throughput and performance in the system requirements. For the SOVA based LTE decoder, the latency for each window is already short compared to decoders using Max-Log-MAP, so using $\alpha = 32$ does not affect the throughput by much. Moreover, choosing to use the NII method will require additional logic to control and perform

the exchange of state vectors across iterations, which makes it a less desirable solution. Therefore only the α-stage warmup method will be considered in the thesis.

3.6.3 Inter-Bank Memory Access

In Section 3.6.1, it was assumed that each SISO decoder will only utilize its own dedicated memory bank. This is however not the case for SOVA, as each decoder will require data from other window's memory banks. Consider the case for window i, where channel input data from bank i (where indices iW to $(i+1)W-1$ are stored) is decoded and extrinsic output $L_e(iW)$ to $L_e((i+1)W-1)$ needs to be output. As described in Section 3.6.2, a SOVA decoder needs α symbols of warm-up to estimate the window's starting state, which means that the last α data stored in memory bank $i-1$ is required for the warm up stage.

To obtain the last output $L_e((i+1)W-1)$, an additional $L+U$ received symbols that are stored in memory bank $i+1$ are needed for SOVA reliability updates. This means that the first $L+U$ data that are stored in memory bank $i+1$ will be needed by the decoder in window i for performing reliability updates. The data access requirements for window i is given in Figure 3.25 where it can be seen that window i will require access to data stored in memory banks $i-1$, i and $i+1$.

However, it can be noted that since all the M decoders are operating concurrently and in step, all the windows will perform bank switching at the same time and the scheduling of the memory bank access by the various decoders is shown in Figure 3.28. At any time, only one decoder will be accessing a given bank of memory, and thus it can be concluded that contention-free memory access is achievable through the memory organization described in Section 3.6.1.

3.7 Comparison of SOVA and Max-Log-MAP

In order to have a good feel for how the SOVA based turbo decoders stack up to the traditional Max-Log-MAP counterparts, this section will compare the two classes of turbo decoders based on the following criteria

- Hardware resource utilization (Section 3.7.1)
- Memory requirements (Section 3.7.2)
- Throughput (Section 3.7.3)

Finally, a recommended hardware architecture that provides the best trade-off is suggested.

3.7.1 Resource Utilization

The hardware resource utilization in terms of estimated MUX and adder counts is as shown in Table 3.12. The number of MUXes and adders required for the

proposed SOVA based decoders is larger than that of the Max-Log-MAP decoders. There is once again a trade-off between memory requirements and hardware resource utilization when choosing between Max-Log-MAP based decoders and SOVA based decoders. Of the proposed SOVA architectures, the hybrid-SOVA with the improved `BPCU` has the lowest resource utilization, requiring approximately 30 % more hardware resources than Max-Log-MAP.

3.7.2 Memory Requirement

The estimated memory/register utilization for each of the hardware architectures is tabulated in Table 3.13. As expected, the SOVA based turbo decoder has lower memory requirements than one that uses the Max-Log-MAP. Max-Log-MAP based decoders need memory to store the outputs of the `ARP` and `BRP` units before finally performing LLR calculations, and the amount of memory needed in multiples of the window size W of the decoder. For SOVA, the merge and update depths are in multiples of 5 to 10 times the constraint length, which is much smaller than W, and thus SOVA based decoders will require much less memory than the Max-Log-MAP ones.

3.7.3 Data Throughput

The data throughput for a turbo decoder is related to the latency of each of the processes in the turbo decoder, as well as the number of iterations to be run for obtaining the final decoded data.

In order to compare the throughput performance of the various SOVA-based turbo decoders to that of a Max-Log-MAP based turbo decoder, the following assumptions are made.

Assume that the window size $W = 768$ and the latency for the LTE internal interleaver is 10 cycles. Based on simulation results, for 16 half iterations, the ideal Max-Log-MAP turbo decoding has around 0.1 to 0.2 dB better BLER performance than SOVA. The BLER performance for SOVA with 16 half iterations is equivalent to that for Max-Log-MAP with 12 half iterations. The estimated data throughput for the turbo decoder architectures are presented in Table 3.14. For comparison purposes, the throughput of Max-Log-MAP for both 12 and 16 half iterations are included in the table.

As observed from Table 3.14, the proposed SOVA based turbo decoders have better throughput than the Max-Log-MAP based decoder. The SOVA based turbo decoder is able to achieve a 79 % improvement in the data throughput as compared to a Max-Log-MAP turbo decoder, at the expense of 0.2 dB degradation in performance. For comparable performance with Max-Log-MAP, a SOVA based turbo decoder that uses 16 half iterations is still 34 % better in data throughput.

3.7.4 Optimal SOVA Decoder

Of the various SOVA based decoder architectures proposed, the recommended architecture for a LTE decoder is one based on the hybrid-SOVA with the improved `BPCU` module. With a memory requirement of only 10 % that of a comparable Max-Log-MAP based decoder, it is able to provide at least 30 % improvement in the overall data throughput.

The only drawback to the SOVA based decoder is that it requires 30 % more hardware resources compared with Max-Log-MAP. Despite this, it is worth keeping in mind that the complexity of the controllers within the two classes of decoders is not considered in this thesis. The SOVA algorithm is a data-driven algorithm that requires minimal control, whereas the Max-Log-MAP algorithm requires more complicated control, and the additional hardware resources utilized by the controller logic may reduce the hardware resource advantage of Max-Log-MAP decoders.

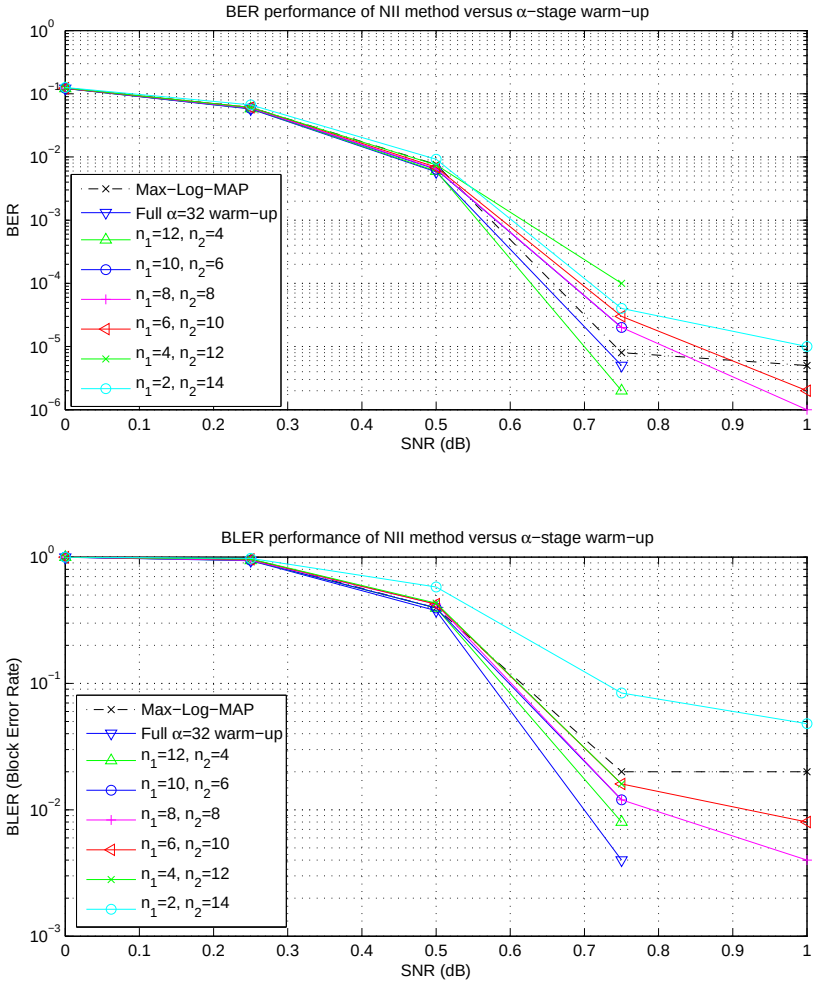


Figure 3.27: BER and BLER performance for SB-SOVA with varying n_1 iterations of α-warmup

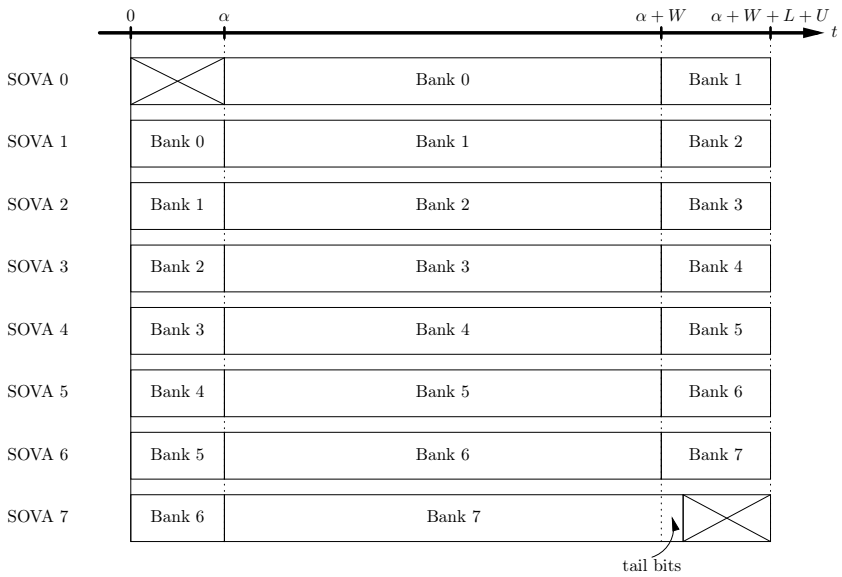


Figure 3.28: Access schedule for memory banks

Table 3.12: Comparison of resource utilization of various SISO decoders

SISO Decoder	Resource utilization		
	General equation for each window	$N = 8$, $W = 768$ $L = 24$, $U = 24$, $U_1 = 12$ $\eta_{\alpha,\beta} = \eta_\Delta = 10$	
		Per Window	Total
Required Number of MUXes			
Max-Log-MAP	$6N - 2$ ($\eta_{\alpha,\beta}$ bit)	460	3 680
HR-SOVA	$N + U$ (η_Δ bit) $NL + NU + 5U$ (1 bit)	824	6 592
SB-SOVA (Δ_k exchange)	$N + NU + 6U$ (η_Δ bit) NL (1 bit)	3 632	29 056
SB-SOVA (s_k exchange)	$N + 6U$ (η_Δ bit) $NU + 5U$ (ν bit) NL (1 bit)	2 648	21 184
hybrid-SOVA (Δ_k exchange)	$N + NU_1 + 5U_1 + U$ (η_Δ bit) $NL + (N + 5)(U - U_1)$ (1 bit)	2 228	17 824
hybrid-SOVA (s_k exchange)	$N + 5U_1 + U$ (η_Δ bit) $NL + (N + 5)(U - U_1)$ (1 bit) $NU_1 + 5U_1$ (ν bit)	1 736	13 888
Required Number of Adders			
Max-Log-MAP	$15N + 14$ ($\eta_{\alpha,\beta}$ bit)	1 340	10 720
HR-SOVA	$3N + U$ (η_Δ bit)	480	3 840
SB-SOVA	$3N + 2U + 1$ (η_Δ bit)	730	5 840
hybrid-SOVA	$3N + U + U_1 + 1$ (η_Δ bit)	610	4 880
Total count (Adders/MUX)			
Max-Log-MAP		1 800	14 400
HR-SOVA		1 304	10 432
SB-SOVA (Δ_k exchange)		4 362	34 896
SB-SOVA (s_k exchange)		3 378	27 024
hybrid-SOVA (Δ_k exchange)		2 838	22 704
hybrid-SOVA (s_k exchange)		2 346	18 768

Table 3.13: Comparison of memory requirement for various SISO decoders

SISO Decoder	Memory Requirement		
	General equation for each window	$N = 8$, $W = 768$ $L = 24$, $U = 24$, $U_1 = 12$ $\eta_L = \eta_\Delta = 10$	
		Per Window	Total
Max-Log-MAP	$N(W+3)$ (η_L bit)	60.234 kbit	482 kbit
HR-SOVA	$NL+U$ (η_Δ bit) $U+N(2L+U)$ (1 bit)	2.508 kbit	20 kbit
SB-SOVA (Δ_k exchange)	$N(L+U)+2U$ (η_Δ bit) $U+NL$ (1 bit)	4.43 kbit	36 kbit
SB-SOVA (s_k exchange)	$N(L+U)+2U$ (η_Δ bit) NU (ν bit) $U+NL$ (1 bit)	4.992 kbit	40 kbit
hybrid-SOVA (Δ_k exchange)	$N(L+U_1)+U+U_1$ (η_Δ bit) $U+N(L+U-U_1)$ (1 bit)	3.469 kbit	28 kbit
hybrid-SOVA (s_k exchange)	$N(L+U_1)+U+U_1$ (η_Δ kbit) $U+N(L+U-U_1)$ (1 bit) NU_1 (ν bit)	3.75 kbit	30 kbit

Table 3.14: Comparison of throughput between SOVA and Max-Log-MAP decoders

	Max-Log-MAP		SB-SOVA, hybrid-SOVA
Block length	6 144		
Number of windows	8		
Window size	768		
Time taken for each process per half iteration (cycles)			
SISO decode	1 536		853
Interleave/deinterleave	10		
Total per half iteration	1 546		863
Number of half iterations	12	16	16
Total time (cycles)	18 552	24 736	13 808
Min clock rate for data rate of 100 Mbit/s	302 MHz	403 MHz	225 MHz
Max data rate at 200 MHz	66.3 Mbit/s	50 Mbit/s	89 Mbit/s
Improvement (%)			
with respect to 16 half iterations			79 %
with respect to 12 half iterations			34 %

Chapter 4

Conclusions

The performance of various SOVA algorithms in the context of LTE turbo decoding was investigated, and it was found that although BR-SOVA gives the best performance, it is not feasible for hardware implementations due to the need for secondary concurrent path tracking and the latency involved in updating the concurrent path reliability values. The proposed SB-SOVA algorithm works around the limitations of BR-SOVA with minimal performance degradation, although the hardware resource utilization is still high as compared with Max-Log-MAP. By utilizing both SB-SOVA and HR-SOVA rules for reliability updates, the new hybrid-SOVA closes the resource utilization gap between the SOVA algorithms and Max-Log-MAP.

Although both resource utilization in adders and MUXes, and memory utilization of SOVA and Max-Log-MAP decoders have been investigated, the actual hardware complexity in terms of silicon area is much harder to quantify. The Max-Log-MAP decoders utilize dense memory, which occupy less silicon area per bit as compared with registers. The control logic, which is expected to be more complicated in Max-Log-MAP as compared with the data-driven SOVA algorithm is also not investigated here. These factors will affect not only the hardware complexity and die area of the resulting hardware accelerator, but also the latency and throughput.

One other performance measure that is not investigated in this thesis is a comparison of the power consumption of the two decoders. This is increasingly important as the increased throughput of the decoder will lead to a corresponding increase in power consumption, which is often the critical factor in mobile devices.

For a better comparison between SOVA and Max-Log-MAP decoders, hardware implementations will be needed. The synthesized netlists will give the actual gate counts, which will enable the designer and computer aided tools to estimate the die area and power consumption of the design.

Bibliography

[1] 3rd Generation Partnership Project. *Technical Specification Group Radio Access Network, Evolved Universal Terrestrial Radio Access (E-UTRA), Multiplexing and Channel Coding*, 8.5.0 edition, December 2008.

[2] E. Boutillon, W. J. Gross, and P. G. Gulak. VLSI architectures for the MAP algorithm. *IEEE Transactions on Communications* 51(2):175–185, 2003.

[3] O. Joeressen and H. Meyr. A 40 Mb/s soft-output viterbi decoder. *IEEE Journal of Solid-State Circuits* 30(7):812–817, July 1995.

[4] E. Yeo, S. Augsberger, W. R. Davis, and B. Nikolić. 500 Mb/s soft-output viterbi decoder. *ESSCIRC* pp. 523–526, 2002.

[5] O. Joeressen, M. Vaupel, and H. Meyr. High-speed VLSI architectures for soft-output viterbi decoding. *Proceedings of the International Conference on Applications Specific Array Processors* pp. 373–384, August 1992.

[6] L. Papke, P. Robertson, and E. Villebrun. Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme. *IEEE International Conference on Communications* 1(23–27):102–106, June 1996.

[7] L. Lin and R. S. Cheng. Improvements in SOVA-based decoding for turbo codes. *Proceedings International Conference on Communications* 3:137–139, June 1997.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes. *IEEE International Conference on Communications* 2:1064–1070, May 1993.

[9] M. Soleymani, Y. Gao, and U. Vilaipornsawai. *Turbo Coding for Satellite and Wireless Communications*. Kluwer Academic Publishers, 2002.

[10] L. R. Bahl, J. Cocke, F. Jelinek, and J. Rajiv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory* 20(2):284–287, March 1974.

[11] G. D. Forney, Jr. The Viterbi Algorithm. *Proceedings of the IEEE* 61(3):268–278, March 1973.

[12] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. *Proceedings Globecom* pp. 1680–1686, November 1989.

[13] J. Hagenauer. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on Information Theory* 42(2):429–445, March 1996.

[14] C. X. Huang and A. Ghrayeb. A simple remedy for the exaggerated extrinsic information produced by the SOVA algorithm. *IEEE Transactions on Wireless Communications* 5(5):996–1002, May 2006.

[15] S. Papaharalabos, P. Sweeney, B. Evans, and P. Mathiopoulos. Improved performance SOVA turbo decoder. *IEE Proceedings Communications* 153(5):586–590, October 2006.

[16] IT++, A C++ library of mathematical, signal processing and communication routines. `http://itpp.sourceforge.net`.

[17] C.-M. Wu, M.-D. Shieh, C.-H. Wu, Y.-T. Hwang, and J.-H. Chen. VLSI architectural design tradeoffs for sliding-window log-MAP decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13(4):439–447, April 2005.

[18] B. Parhami. *Computer Arithmetic Algorithms and Hardware Designs*. Oxford University Press, 2000.

[19] Z. Wang and K. K. Parhi. High performance, high throughput turbo/SOVA decoder design. *IEEE Transactions on Communications* 51(4):570–579, April 2003.

[20] J. Dielissen and J. Huisken. State vector reduction for initialization of sliding windows MAP. *Second International Symposium on Turbo Codes and Related Topics*, September 2000.