Master's Thesis

Turbo decoder with early stopping criteria

Henrik Ljunger

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Turbo decoder with early stopping criteria

Henrik Ljunger
ael10hlj@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Joachim Rodrigues

October 9, 2016

Printed in Sweden
E-huset, Lund, 2016

Abstract

The turbo code used in the 3GPP Long Term Evolution(LTE) standard have been chosen specifically to simplify parallel turbo decoding and thus achieving higher throughputs. The higher data rates however leads to an increased computational complexity and thus a higher power and energy consumption of the decoder. This report presents a turbo decoder for the LTE standard with a stopping criteria aimed to reduce the power and energy consumption of the turbo decoder. The decoder can be configured to use 1,2 ,4 ,8 or 16 MAP decoders in parallel achieving a throughput of 110 Mb/s for 7 iterations when running at a clock frequency of 200 MHz. The decoder were synthesised with 65 nm low power libraries with an area of 1.6 mm^2. The post-synthesis simulations shows that the stopping criteria can lead to a significant lower energy consumption with no performance loss.

Acknowledgements

First I would like to thank Michal Stala and Muris Sarajlic for all their support guidance and encouragement during the whole duration of this thesis. I would also like to thank Liang Liu for his support and help with practical things. Furthermore I would like to thank Magnus Midholt and the rest of the team at Mistbase for their encouragement and for making my time at the office fun and enjoyable. Lastly I would also like to thank my family and friends for their support throughout this work.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

The mobile communication market is constantly growing and is expected to continue doing so with more connected devices each day. With a crowded frequency spectrum, the available bandwidth is limited and has to be shared between the different service providers. To accommodate the growing number of users yet providing a high quality of service and data transfer rates, an efficient use of the available bandwidth are essential. For this reason, error correction algorithms have become a vital part of modern communication systems. These algorithms are often key when approaching the theoretical Shannon limit which determines the maximal throughput of the system. One efficient and popular error correction algorithm are the turbo code, consisting of an encoder and decoder, which are used in many modern standards. One such standard is the long term evolution(LTE) standard, which is part of the 4th generation(4G) communication networks, where turbo codes are used in the shared data channel. The focus of this master thesis will be on the implementation of a turbo decoder for the LTE standard.

1.1 Background

The popularity of turbo codes is much due to their impressive performance which can reach within a few tenths of dB from the Shannon limit [2]. The error correcting capabilities allows for an efficient communication since, among others, the amount of data that has to be retransmitted due to the occurrence of errors can be reduced. This however, increases the requirements on the receiver leading to a more complex implementation. The turbo decoder can therefore be a major part of the system's total energy consumption. With the majority of the user equipment being handheld devices such as mobile phones or tablets with a limited energy budget, keeping the energy consumption low are of great importance. Therefore various modifications exist that aims to reduce the energy consumption of the decoder. One of them is to stop the updating of those codeword bits whose reliability is above a certain threshold. This leads to potential savings in the number of calculations as well as the total number of memory accesses, thus leading to a higher energy efficiency of the decoder. In this thesis, the aim is to investigate the potential savings of an actual hardware implementation of the above described stopping criteria.

1.2 Objective

The goal of this master thesis is to implement a turbo decoder in hardware for the LTE standard with the previously described stopping criteria. The decoder shall be compliant with the LTE standard and meet the requirements regarding timing and latency and the design should be able to run on an FPGA. The development of the thesis can be divided into three major milestones. First, the implementation of the max-log-map module. Second, the integration of the two sub-modules and the implementation of the interleaver. Third, the implementation of the early stopping criteria.

1.3 Methodology

At first, a literature study was performed in which the underlying theory and algorithm were investigated. There exist different algorithms that can be used to implement a turbo decoder, this thesis however specified the use of the max-log-MAP algorithm which therefore was the main focus. The max-log-MAP algorithm and it's components were studied to determine their impact on the performance, throughput and hardware complexity etc. In this step, the parts that were most crucial for the performance and throughput was identified as well as which was the most costly in terms of hardware. With this information different architecture with varying degrees of parallelism and time multiplexing could be explored from which an efficient final architecture could be chosen.

A major part of this work has been the use of high-level synthesis(HLS). In HLS the functionality of the hardware is described on a cycle accurate basis using bit accurate types in a high-level programming language which in this thesis has been C++. These models are then turned into RTL models by the HLS tool. The main advantage is that the testing and simulations can be done using test benches written in the programming language which speeds up the process considerably.

The implementation of the milestones thus began with the creation of behavioral models in Matlab in which their functionality also was simulated. In the next step, a behavioral model in C++ was created based on the Matlab version to which it's functionality was verified. These models were then modified to use bit accurate types, and after verifying the functionality they were transformed to RTL code using the Catapult HLS tool. The RTL was then implemented and tested on an FPGA.

1.4 Structure

The remainder of this report is structured as follows. In chapter 2 an overview of a system using turbo codes will be given and the encoder and decoder will be introduced as well as the proposed stopping criteria. The concept of error control coding and hard/soft decisions will be briefly introduced together with interleaving and the MAP algorithm on which the decoder in this work is based on. In chapter 3 the hardware implementation of the turbo decoder and it's components will be presented. In chapter 4 the results of the simulations will be given together with area and power consumption figures. The obtained results will then be analyzed and discussed in chapter 5.

Chapter 2

System Overview

Turbo codes were introduced by Berrou in 1993[2] and are a popular error correction algorithm widely used in modern communication. The popularity of turbo codes is mainly as previously mentioned due to their capability of reaching performances within a few tenths of dB from the Shannon limits that set the theoretical throughput limit[13].

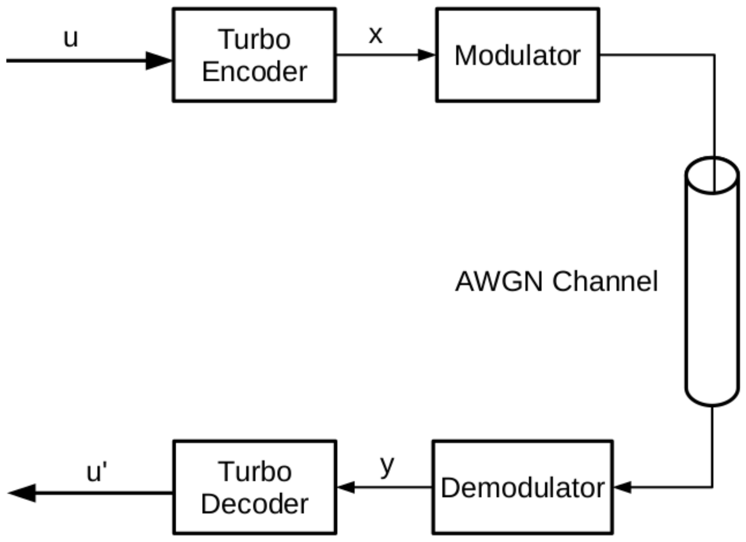


Figure 2.1: A simplified overview of a communication system

An overview of a system using turbo codes can be modeled as shown in figure 2.1 where the part specific for turbo codes are the turbo encoder at the transmitter and the turbo decoder at the receiver. At the transmitter, the encoder receives the information message(u) from which it produces the coded message(x). The coded message are then modulated to an analog signal and transmitted over a channel. In this example, the channel is modeled as an additive white Gaussian noise (AWGN) channel which is the simplest possible case of a channel. In reality however, the channel will be time- and frequency-selective i.e change over time and frequency. As a consequence of passing through the channel, the signal is subjected to noise and interference thus increasing the probability of errors occurring in the received data. At the receiver the signal is demodulated before the received message(y) are passed to the decoder. The decoder then produces an estimation($\hat{u}$) of the information message trying to detect and correct any introduced errors during the transmission.

2.1 Error Control Coding

When transmitting over a physical channel the signal is subjected to noise, interference and fading along it's path from the transmitter to the receiver and it is inevitable that errors are introduced at times. In a cellular system the data are transmitted by radio waves where a number of devices share a limited amount of bandwidth. In these cases, some of the major sources of errors, in combination with thermal noise, consists of the interference from other nearby devices as well as the interference due to multipathing in which multiple versions of the signal are received, which can arise due to the signal being reflected by obstacles in the path. There are mainly two methods that a system can employ to handle the occurrence of errors, these are either forward error correction(FEC) or some version of automatic repeat request(ARQ).

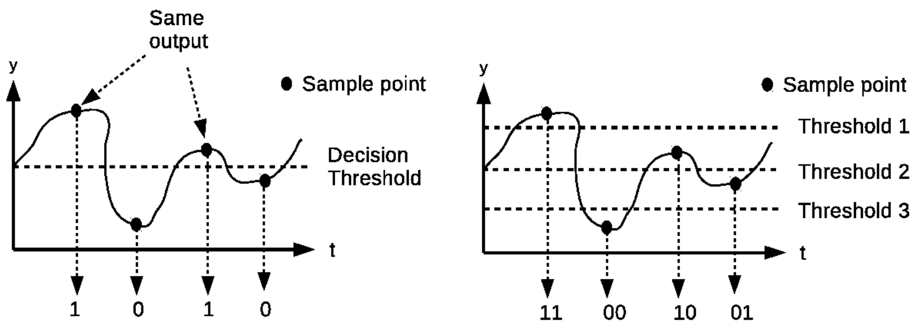
In ARQ systems the receiver sends an acknowledgment (ACK) to the transmitter if the message was correctly received and a negative acknowledgement (NAK) if it contained any errors. Messages that contains errors have to be retransmitted until they are correctly received and ARQ is therefore sometimes also called backward error correction(BEC). An FEC system is on the other hand also required to be able to correct up to a certain amount of errors in the received message. This is achieved by using channel coding in which the message are encoded by appending redundancy bits to the information message before transmitting it. The produced redundancy bits depends on the information sequence and the relationship depends on the type of code that is used, for example, linear block codes or convolutional codes. The redundancy bits are then used by the decoder to detect and correct the introduced errors. ARQ only requires the use of error detection which leads to simpler hardware implementations, however, the retransmission of data reduces the spectral efficiency which can be un-tolerable if the occurrence of errors is common. FEC requires a more complex implementation of the receiver and an increased data overhead due to the added redundancy but in turn, reduces the amount of data that has to be retransmitted.

In cellular systems the wireless transmission of data over varying distances with a lot of interference from different sources often leads to an unreliable channel and FEC are therefore often used in cellular communication systems. Usually, the FEC are combined with ARQ to form an HARQ(Hybrid-ARQ) system that combines the advantages of both methods.

2.2 Hard and Soft Decision

Transmitting data over a physical channel are usually done using modulation in which the digital data are transformed to a continuous waveform which is then transmitted. This is done by mapping the bits of the binary information to symbols which are represented by a waveform of a specific duration. How this mapping is performed depends on the used modulation scheme which for LTE consists of QPSK, 16QAM or 64QAM for the data channel.

At the receiver, the demodulator converts the received waveform back to discrete values which are done by sampling the received signal creating samples of real values. If hard decision is used these samples are quantized into binary data by comparing the sample to a threshold to determine the bits values. In soft decision the output consists of so-called soft bits which are real values whose sign corresponds to the hard decision of the bit and the magnitude the confidence level of this decision which depends on the distance to the threshold. Using soft decision, values close to the threshold, which is unreliable, are given less of a negative impact on the decoding performance compared to hard decision and the use of soft decision has been shown to increase the performance over hard decision[14]. In figure 2.2 an example of the decision methods are shown.



(a) Hard Decision **(b)** Soft Decision

Figure 2.2: Illustrative example of Hard and Soft Decisions

2.3 Encoder

The turbo encoder used in LTE is a parallel systematic concatenated convolutional (PSCC) code that consists of two recursive systematic convolutional (RSC) codes separated by an interleaver and are shown in figure 2.3. The two RSC encoders are identical and have eight states and each produces a two-bit codeword consisting of a systematic and a parity bit. The turbo encoder thus produces one two-bit codeword for the non-interleaved data sequence and one for the interleaved sequence, however, the interleaved systematic stream are not transmitted. The output thus consists of three output bits for every input bit resulting in a

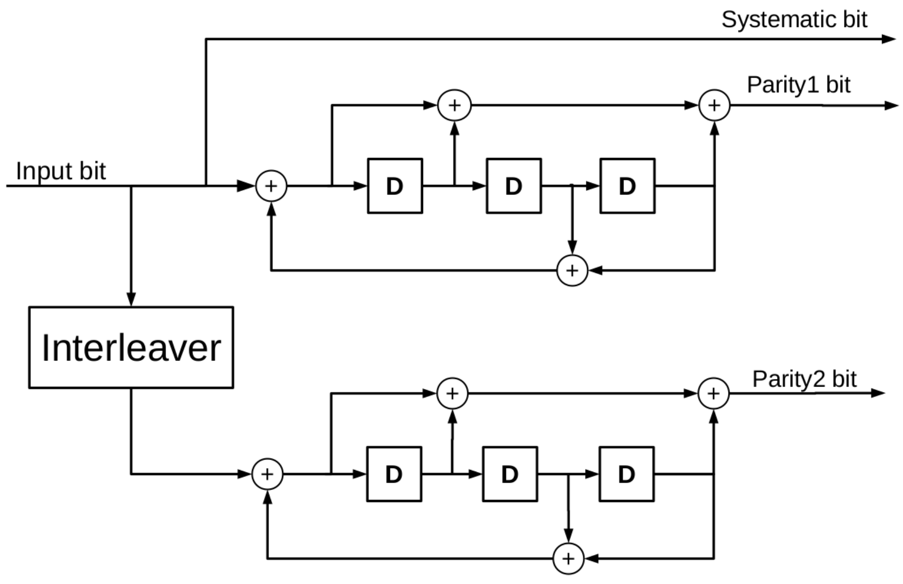


Figure 2.3: The turbo encoder for the LTE standard[1]

code rate of $\frac{1}{3}$. Higher or lower code rates can be achieved by puncturing respectively repetition of the encoded data. The RSC encoders are systematic meaning that the input bits appears unchanged in the output and the systematic bit is thus identical to the information bit.

2.4 Maximum A-Posteriori Algorithm

The maximum a-posteriori(MAP) algorithm also known as the BCJR named after it's inventors Bahl, Cocke, Jelinek and Raviv was presented in 1974[3]. The increased computational complexity compared to the Viterbi algorithm lead to that it was seldom used in practice. The algorithm has lately risen in popularity together with turbo codes.

The BCJR calculates an a-posteriori log-likelihood ratio(LLR) for each transmitted information bit based on the whole received input sequence. For the system in consideration, the input to the BCJR consists of the codeword produced by one of the RSC encoders in figure 2.3 as well as the apriori information. The encoded message bit (u_k) can take the values of either +1 or -1, with -1 representing a '0', with an apriori probability $P(u_k)$ from which the apriori LLR can be calculated as in 2.1. $L(u_k)$ represents the confidence level of the decision in which bit that was encoded at time k prior to the start of the decoding and are zero if both cases are equally likely. The a-posteriori are then the confidence in the decision after the whole sequence has been received as defined in 2.2.

$$L(u_k) = \frac{P(u_k = +1)}{P(u_k = -1)} \tag{2.1}$$

$$L(u_k|y) = \frac{P(u_k = +1|y)}{P(u_k = -1|y)} \tag{2.2}$$

To help visualize the decoding process a trellis state diagram is often used. In figure 2.4 an example of a trellis state diagram of a four state RSC encoder are shown. The trellis consists of states that are connected through branches, each branch is associated with the information bit being either '1' or '0' where a solid line corresponds to a '0' and a dotted line to a '1'. Each branch has a branch metric (γ) associated with it and each state has a forward (α) and a backward (β) state metric associated with it. These metrics will be defined later on in this section.

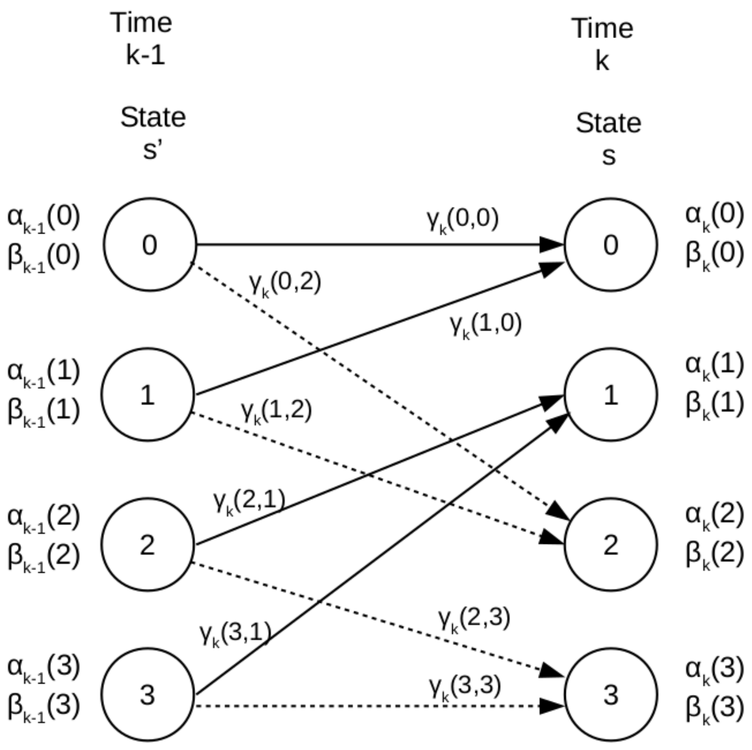

Time k-1
State s'
Time k
State s
$\alpha_{k-1}(0)$ $\beta_{k-1}(0)$
$\alpha_{k-1}(1)$ $\beta_{k-1}(1)$
$\alpha_{k-1}(2)$ $\beta_{k-1}(2)$
$\alpha_{k-1}(3)$ $\beta_{k-1}(3)$
0
1
2
3
$\gamma_k(0,0)$
$\gamma_k(0,2)$
$\gamma_k(1,0)$
$\gamma_k(1,2)$
$\gamma_k(2,1)$
$\gamma_k(2,3)$
$\gamma_k(3,1)$
$\gamma_k(3,3)$
$\alpha_k(0)$ $\beta_k(0)$
$\alpha_k(1)$ $\beta_k(1)$
$\alpha_k(2)$ $\beta_k(2)$
$\alpha_k(3)$ $\beta_k(3)$

Figure 2.4: An example of a trellis state diagram from a four state RSC encoder

Assuming that a N bit message is received and that the state at the time step k are $S_k = s$ and that the previous state was $S_{k-1} = s'$. Up to this point, k-1 input symbols have already been received and N - k are to be received. The received symbol sequence can be divided into three parts as

$$y = y_1 y_2 ... y_{k-1} y_k y_{k+1} ... y_N = y_{<k} y_k y_{>k}$$

where one part represents the past, one the present and one the future. The a-posteriori LLR of the kth input bit are then calculated from 2.3 where $P(s', s, y)$ are the joint probability of receiving the N-bit sequence y and that the state at time k are s with the previous state being s′. Here R_1 and R_0 marks the summation over the state transitions corresponding to u_k = 1 respectively u_k = -1. At the time step k α, γ and β are probabilities associated with the past, present and future of y and $L(u_k|y)$ can be expressed using these probabilities as in 2.4.

$$L(u_k|y) = ln \frac{\sum_{R_1} P(s', s, y)}{\sum_{R_0} P(s', s, y)} \tag{2.3}$$

$$L(u_k|y) = ln \frac{\sum_{R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)} \tag{2.4}$$

The branch metric $\gamma_k(s', s)$ are thus the conditional probability of the received symbol being y_k when the current state is s and the previous state were s′. For an AWGN channel this becomes

$$\gamma_k(s', s) = C_k e^{u_k \frac{L(u_k)}{2}} * e^{(\frac{L_c}{2} \sum_{l=1}^{n} x_{kl} * y_{kl})} \tag{2.5}$$

where L_c is the channel reliability measure defined as

$$L_c = 4aE_cN_0 = \frac{4aR_cE_b}{N_0}$$

where R$_c$ is the code rate, $\frac{N_0}{2}$ is the noise bilateral power spectral density, a is the fading amplitude, E$_c$ and E$_b$ are the transmitted energy per coded bit respectively message bit. The C$_k$ are a constant that will be canceled out in the LLR calculations. The forward and backward state metrics are calculated recursively from 2.9 and 2.10.

$$\alpha_k(s') = \sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s) \quad \alpha_0(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \tag{2.6}$$

$$\beta_{k-1}(s') = \sum_{s'} \beta_k(s) \gamma_k(s', s) \quad \beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \tag{2.7}$$

The γ are required by both α and β and must be calculated first. The α's are calculated during the recursion of the trellis in the forward direction. The β metrics are calculated during the backward recursion and can only be obtained after the whole sequence has been received.

2.4.1 MAP Simplifications

The MAP algorithm requires a lot of multiplications which makes it unattractive for a hardware implementation. Instead, some simplification is usually employed. Two popular simplifications are the max-log-MAP and the log-MAP. Both operates in the logarithmic domain which turns the multiplications to additions thus reducing the computational complexity. The branch metrics and state metrics are now defined as

$$\Gamma_k(s',s) = lnC_k + \frac{u_k L(u_k)}{2} + \frac{L_c}{2} \sum_{l=1}^{n} x_{kl} y_{kl} \tag{2.8}$$

$$A_k(s) = max^*(A_{k-1}(s') + \Gamma_k(s',s)) \quad A_0(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases} \tag{2.9}$$

$$B_{k-1}(s') = max^*(B_k(s) + \Gamma_k(s',s)) \quad B_N(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases} \tag{2.10}$$

in which the max operation differs for the two and are defined as

$$max^*(a,b) = \begin{cases} max(a,b) + ln(1 + e^{-|a-b|}) & log - MAP \\ max(a,b) & max - log - MAP \end{cases} \tag{2.11}$$

The a-posteriori LLR then becomes

$$L(u_k|y) = max^*_{R_1}(A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)) - max^*_{R_0}(A_{k-1}(s') + \Gamma_k(s',s) + B_k(s)) \tag{2.12}$$

The correction term of the log-MAP improves it's performance compared to the max-log-MAP but makes the implementation more complex. With the use of extrinsic scaling the performance gap can be reduced and in this work the max-log-MAP approach will be taken.

2.5 Turbo Decoder

The turbo decoding algorithm is an iterative SISO (Soft-Input Soft-Output) algorithm that in each iteration updates the a-posteriori LLR based on the channel information of the systematic and parity bits together with the apriori LLR. The general structure of a turbo decoder is shown in figure 2.5 and consists of two SISO decoders, an interleaver and a de-interleaver. Each pass of a decoder is called a half-iteration whereas a pass of both decoders is called a full iteration. Each decoder produces an a-posteriori LLR as well as an extrinsic LLR(L_e) which for a systematic encoder are obtained as

$$L_e = L(u_k|y) - L(u_k) - L_c y_{sk} \tag{2.13}$$

and are an updated and presumably better estimation of the apriori LLR. This extrinsic information is passed as the apriori input to the subsequent decoder in the next half-iteration. By repeating this procedure the confidence in the decision of the output are increased after each half iteration and after a defined number of iterations or if a stopping criterion is fulfilled the decoding is halted and the final hard decision is made.

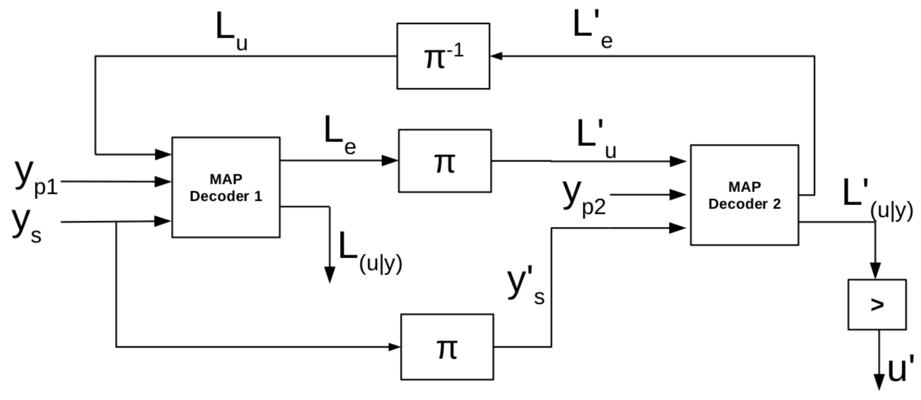


Figure 2.5: Architecture of the typical turbo decoder

2.6 Interleaving

Interleaving mainly serves two different tasks in turbo codes. One is to increase the distance of the code by breaking up so-called self-terminating input sequences[5]. The other is to improve the exchange of extrinsic information between the two decoders in figure 2.5 by decreasing the correlation between the extrinsic inputs[6].

2.6.1 QPP Interleaver

The use of interleaving is a major contributor to the performance of turbo codes. The choice of interleaver is important when it comes to parallel decoding in which multiple MAP decoders are employed. This potentially introduces memory conflicts during the second half of the iteration if two or more decoders tries to access different addresses of the same memory as shown in figure 2.6. The LTE standard specifies the use of a QPP interleaver that is based on algebraic constructions via permutation polynomials over integer rings which have been shown to be contention free for every factor of the interleaver length. An example of a contention free situation are shown in figure 2.7 where each decoder needs data from different memories. For a codeword of length K the interleaving address at index i is calculated as

$$\pi(i) = (f_1 * i + f_2 * i^2) mod K \tag{2.14}$$

where f_1 and f_2 are parameters that depends on K and are specified in the standard[1].

2.6.2 Contention Free Property

An interleaver is said to be contention free for a window of length L if the following property holds,

$$\left\lfloor \frac{f(i+mL)}{L} \right\rfloor \neq \left\lfloor \frac{f(i+nL}{L} \right\rfloor \tag{2.15}$$

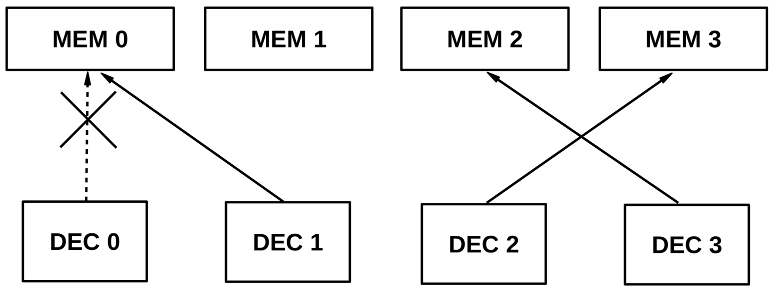


Figure 2.6: Example of a memory conflict. Decoder 0 and 1 both need to read data from memory 0

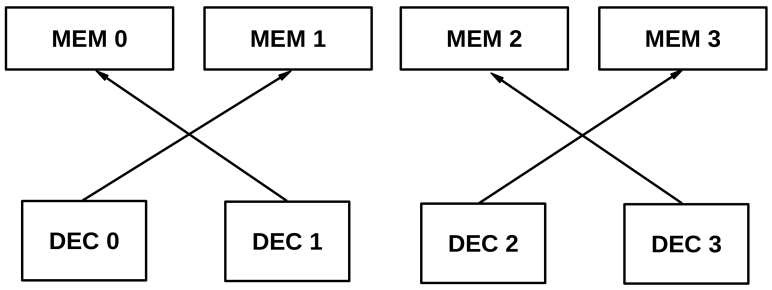


Figure 2.7: Example of a contention free situation. All decoders reads data from different memories

where $0 \leq i < L, m \neq n$ and $0 \leq m, n < P$ where P is the factor of parallelization. The LTE standard specifies 188 different codeword sizes which all are even divisible by 2,4 and 8 which means that the contention free property holds for these parallelism degrees. Furthermore all the codeword sizes larger than 512, 1024 and 2048 are also even divisible by 16, 32 respectively 64. The contention free property basically means that the generated addresses are separated by a factor of L.

2.6.3 Recursive Calculation

Calculating the address from 2.14 requires the use of multiplications and the modulo operator that would lead to a complex hardware implementation. However, the QPP interleaver has a couple of algebraic properties that allows calculation of the addresses to be simplified[12].

These properties allows the addresses to be calculated recursively from the equations in 2.16 - 2.21 where i_0 is the starting address, d is the step size and π_0, g_0 and z are pre-calculated values. This method also includes the modulo operator but π and g will always be smaller than K[7] thus the modulo operator can be implemented using adders.

$$\pi_0 = (f_2 i_0^2 + f_1 i_0) modK \tag{2.16}$$

$$\pi(i+d) = (f_2(i+d)^2 + f_1(i+d)) modK = (\pi(i) + g(i)) modK \tag{2.17}$$

$$g(i) = (2df_2 i + d^2 f_2 + df_1) modK \tag{2.18}$$

$$g(i+d) = (g(i) + 2d^2 f_2) modK = (g(i) + z) modK \tag{2.19}$$

$$g_0 = (2df_2 i_0 + d^2 f_2 + df_1) modK \tag{2.20}$$

$$z = (2d^2 f_2) modK \tag{2.21}$$

By slightly modifying 2.17 and 2.19 the addresses can be calculated in the backward direction as in 2.22 and 2.23.

$$\pi(i-d) = (\pi(i) - g(i-d)) modK \tag{2.22}$$

$$g(i-d) = (g(i) - z) modK \tag{2.23}$$

2.7 Stopping Criteria

The stopping criteria investigated are based on the iterative manner of the turbo decoding algorithm in which the confidence of the output decision is increased after each half iteration. After each pass of a decoder, the produced extrinsic LLR are compared to a threshold. If above the threshold the updating of the LLR is stopped for the remainder of the decoding. This threshold thus represents the point in which the confidence in the output decision is said to be strong enough.

If L_e have reached the threshold, throughout the rest of the decoding process no further calculations of the a-posteriori or extrinsic LLR are needed, thus for the remaining iteration the MAP decoder only needs to calculate the state metrics. No further updating of the value in the memory are required as well leading to a reduced number of required memory accesses. The savings in memory accesses and calculations will potentially lead to a more energy efficient implementation.

2.8 High Level Synthesis

In this work high-level synthesis(HLS) have been used in the design of the hardware implementation. In HLS the functionality of the hardware is described using a high-level programming language such as C or C++ instead of using VHDL or Verilog as in the traditional RTL design flow. The design flow of HLS is shown in figure 2.8.

To start with, a behavioral model in C++ is created, which can be compiled and executed like any other C++ program, to test the functionality of the algorithm and architecture. This model is then transformed into a synthesizable model using bit-accurate data types. The functionality of the newly formed model can then be simulated using the same test bench as for the behavioral model.

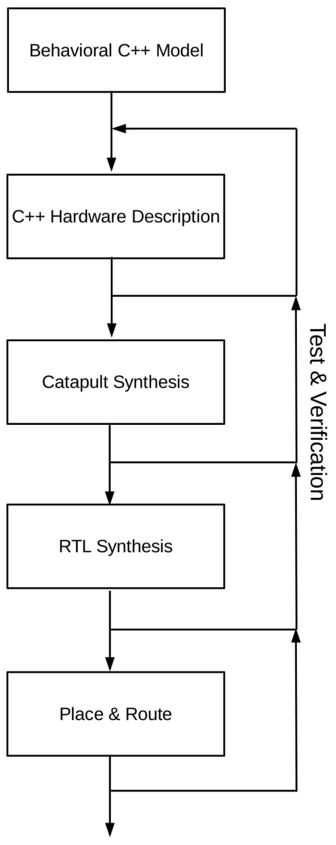


Figure 2.8: Model of a HLS design flow

When the correct functionality as been established the synthesizable model are run through the HLS tool which transforms it into an RTL model in either VHDL or Verilog. An RTL simulation can then be performed to verify the functionality before performing the RTL synthesis for either FPGA or ASIC.

A major advantage of using HLS are the shorter simulation times since running the C++ test benches are around 1000 times faster than RTL simulations. Another advantage is that the tool can perform timing transformations such as pipelining the design automatically thus freeing up time for the designer.

The quality of the resulting RTL depends of course on the high-level description inputted to the HLS tool. Writing a functional description as if it was targeted at software may thus result in a poor end result. It is therefore important that the development of the high-level description are carried out keeping in mind that the final target is hardware as one would when designing RTL using VHDL.

Chapter 3

Hardware implementation

A key challenge in the design and implementation of a turbo decoder is balancing the decoding performance and throughput versus the area and power consumption. In this thesis, the max-log-MAP algorithm has been implemented as it is more hardware friendly than the log-MAP while still offering good decoding performance. The target communication standard imposes strict requirements on the throughput that has to be met. To obtain high throughput values the decoder generally needs to employ a high degree of parallelism. Thus a high throughput leads to an increased area. As the LTE standard consists of different categories with different throughput requirements and the hardware implementation in this work are therefore aimed at having a scalable parallelism degree to support the different requirements. To decrease the power consumption an early stopping criterion can be implemented. In this thesis, a new stopping criterion with the aim of a negligible performance and area penalty are implemented in hardware.

3.1 Max-log-MAP Decoder

The max-log-MAP decoder implements the equations (2.8) - (2.12) that were presented in the previous chapter. These are implemented in three sub-blocks, the branch metric unit (BMU) that calculates the set of $\gamma_k(s', s)$ at each trellis step, the state metric unit (SMU) that calculates the forward and backward state metrics during the forward respectively backward recursion and the log-likelihood ratio unit (LLRU) which calculates the a-posteriori LLR from (2.12). The input to the decoder consists of the channel estimation of the systematic and parity bit provided as soft decisions as well as the apriori LLR. From this, the decoder then outputs the a-posteriori LLR and the extrinsic LLR. The architecture of the implemented max-log-MAP decoder is shown in figure 3.1.

The apriori LLRs are stored in a memory during the forward recursion to prevent a memory read/write conflict during the backward recursion in which the a-posterior LLR are written to the LLR memory. Instead of storing the forward state metric during the forward recursion, the A + Γ term in (2.12) that, as seen in (2.9), are calculated in the SMU are stored in the $A\Gamma$ memory thus saving an addition step in the LLRU. The Beta stakes memory stores the backward state metrics between iterations and are needed due to the utilization of the sliding window(SW) approach.

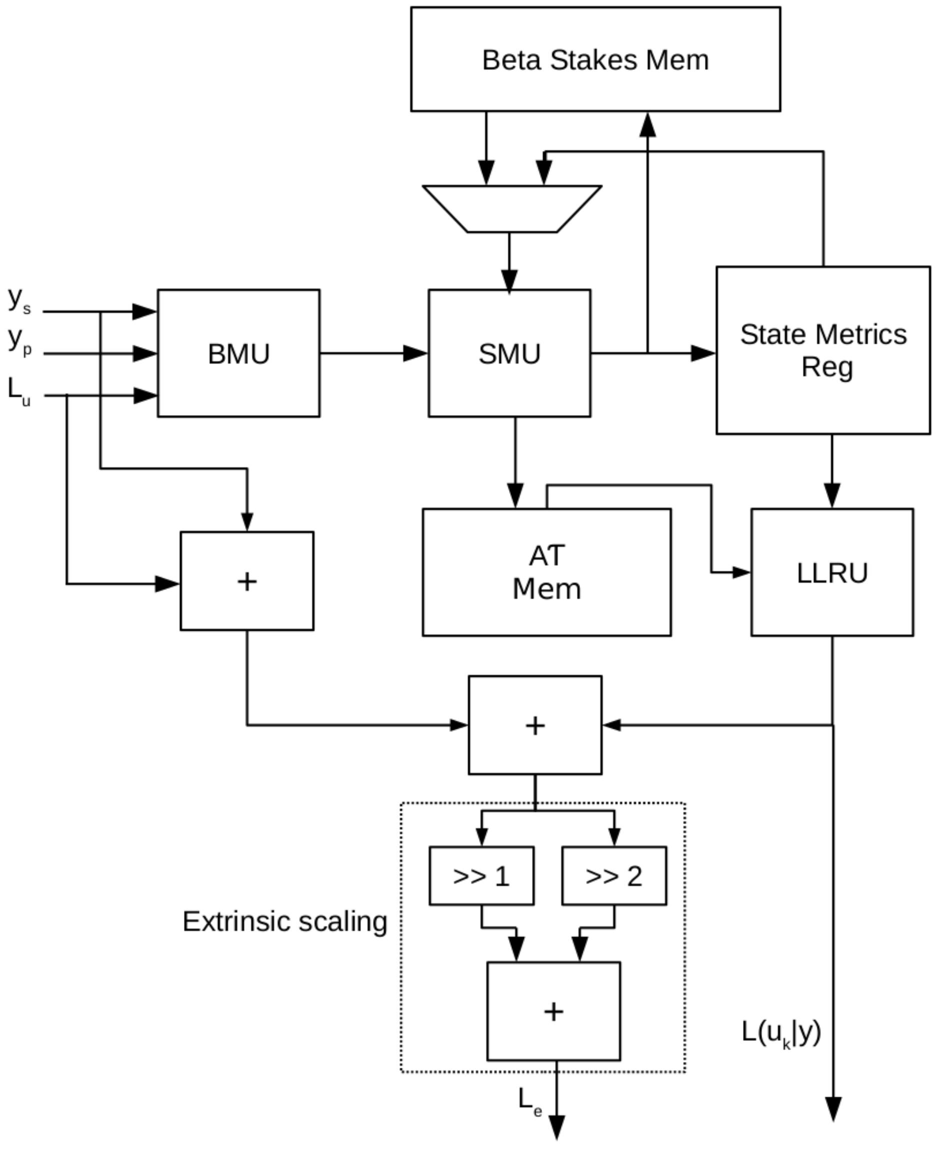


Figure 3.1: Architecture of the implemented max-log-MAP decoder with extrinsic scaling factor of 0.75

3.1.1 Sliding window

In the sliding window approach, the codeword is split into windows of length L. The decoding are then performed by a forward and backward recursion of each window. In contrast an implementation of the max-log-MAP without any modifications i.e the no sliding window approach (NSW), performs a full forward recursion before the start of the backward recursion. The NSW suffers from high memory requirements for large codeword sizes since at minimum eight state metrics needs to be stored for each trellis step. The maximum codeword size of LTE is 6144 which would lead to a substantial amount of on-chip memory. In the sliding window (SW) approach on the other hand only L sets of state metrics need to be stored and with L ranging around 32-64, this leads to significant memory savings thus the SW approach is practically always taken for hardware implementations.

A problem with the use of an SW is that the backward metrics are unknown at the start of the backward recursion of a window. The initial values at the end of the window therefore have to be estimated. A common method for the estimation is to use a guard window that extends into the next window. The backward metrics are initialized to zero at the end of the guard window over which a backward recursion are then performed. The backward metrics obtained at the start of the guard window are then used as the initial values for the backward recursion of the SW. The drawbacks with this method are either a lower throughput due to the delay of the added backward recursion or additional hardware needs to be added to perform the initialization in parallel with the forward recursion.

In this implementation, another method in which the metrics from the previous iteration are used as the initial values is employed. In the first iteration the backward metrics at the end of the windows are initialized to zero and the values obtained at the end of the backward recursion are stored in memory and then used to initialize the backward metrics in the next iteration. In this method there is no added decoding delay but it requires additional memory to store the metrics between iterations. This additional memory is fixed and don't depend on the parallelization degree and will therefore have a low impact on the total area when the design is parallelized.

3.1.2 Branch Metric Unit

The BMU calculates the set of branch metrics for each trellis step according to (2.8). From (2.8) it is seen that the value of $\gamma_k(s', s)$ only depends on the associated codeword which in LTE consists of a pair of bits resulting in four unique branch metrics at each trellis step. Furthermore the codewords '11' and '10' are the negation of '00' respectively '01' and the implementation of the BMU therefore only requires three adders and are shown in figure 3.2.

3.1.3 State Metric Unit

The SMU calculates the state metrics by using eight of the add-compare-select(ACS) units shown in figure 3.3 in parallel. The recursive way of these calculations prevents the block from being pipelined and it thus sets an upper bound of the max-

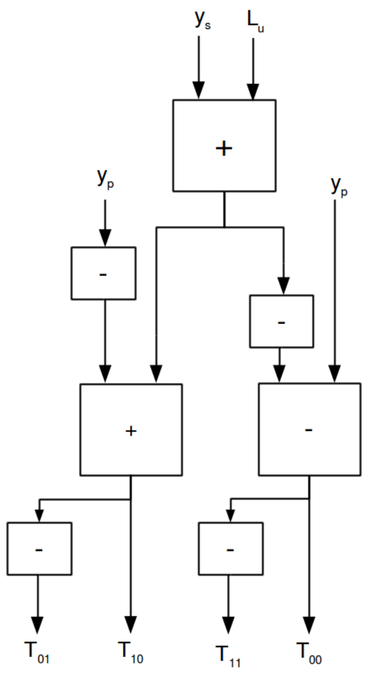


Figure 3.2: Architecture of the branch metric unit

imum achievable frequency of the decoder. It also tends to make the state metrics to grow in each step thus a normalization method are required to be employed to prevent errors arising due to arithmetic overflow. The normalization method chosen in this work are called modulo normalization and are a popular method since it has a small impact on the length of the critical path.

In modulo normalization instead of preventing overflowing the metrics are allowed to overflow, this works since it is the difference between the metrics that affects the LLR calculations and not their actual value. As long as the absolute difference between any two metrics are bounded by $\frac{C}{2}$ with C being a constant and by using the modified comparison rule defined in (3.1) the actual differences can be obtained[11].

$$z(\overline{m}_1, \overline{m}_2) = \overline{m}_{1_{MSB}} \oplus \overline{m}_{2_{MSB}} \oplus y(\hat{m}_1, \hat{m}_2) \tag{3.1}$$

3.1.4 Log-likelihood Ratio Unit

The LLRU calculates $L(u_k|y)$ by first calculating the sum $A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)$ for each branch. The maximum result associated with a transition due to +1 as well as -1 are then calculated by a tree compare-select(CS) network. The surviving result of the -1 is then subtracted from the result of the +1 to form the $L(u_k|y)$. The CS network is shown in figure 3.4.

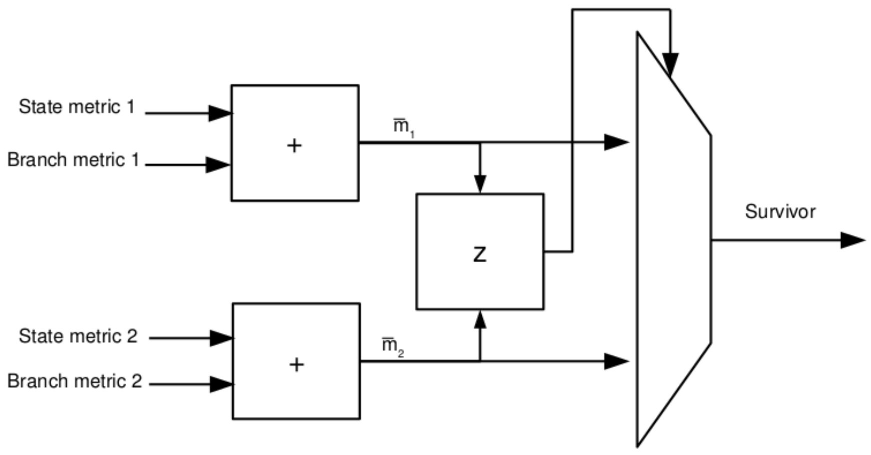


Figure 3.3: Architecture of a ACS unit of the state metric unit

3.1.5 Extrinsic Scaling

The absent correction term in the max-log-MAP compared to the log-MAP, as seen in 2.11, tends to make the output estimation of the max-log-MAP a bit too optimistic. To reduce the performance penalty of the missing correction term the extrinsic information can be scaled down after each half-iteration[10]. The optimal scaling factor ranges between 0.7 to 0.9 and depends on the SNR and the iteration number. In this implementation, a constant scaling factor of 0.75 which allows for a hardware friendly implementation has been chosen.

3.2 Interleaver

The interleaver consists of two address generator block, one which generates the addresses in the forward direction from (2.16) to (2.21) and one that generates the addresses in the backward direction according to (2.22) and (2.23). Since π and g are always smaller than K the forward address generator can be implemented as shown in figure 3.5 while the backward generator is implemented as in figure 3.6. During the forward recursion, the output of the interleaver is the address generated by the forward generator while in the backward recursion the address from the backward generator is outputted. At the end of the sliding window in the forward recursion, the π and g information of the forward generator is used as the π_0 and g_0 values to initialize the backward generator which will then output the addresses in the inverse order.

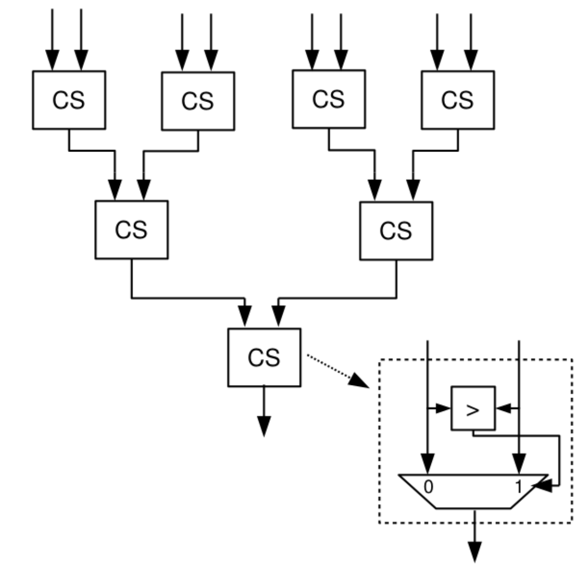


Figure 3.4: The compare-select network finding the maximum of the eight inputs

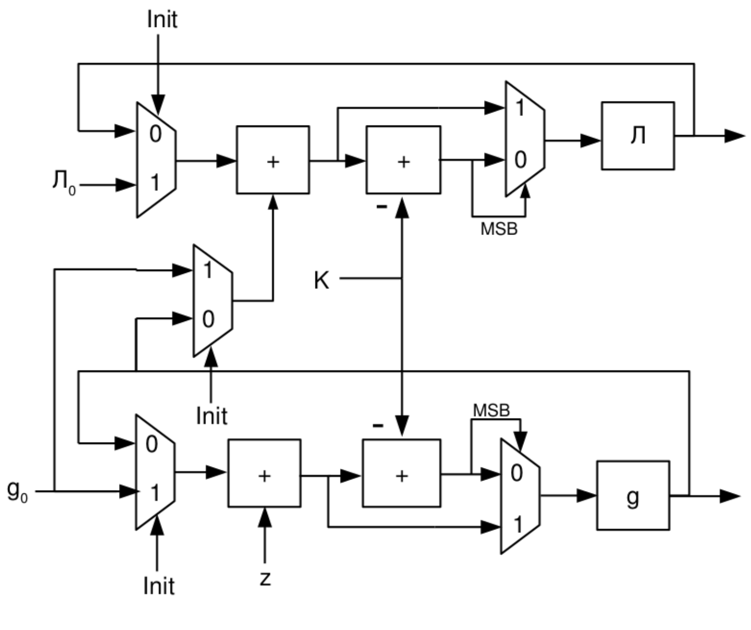


Figure 3.5: Address generator block for the forward direction

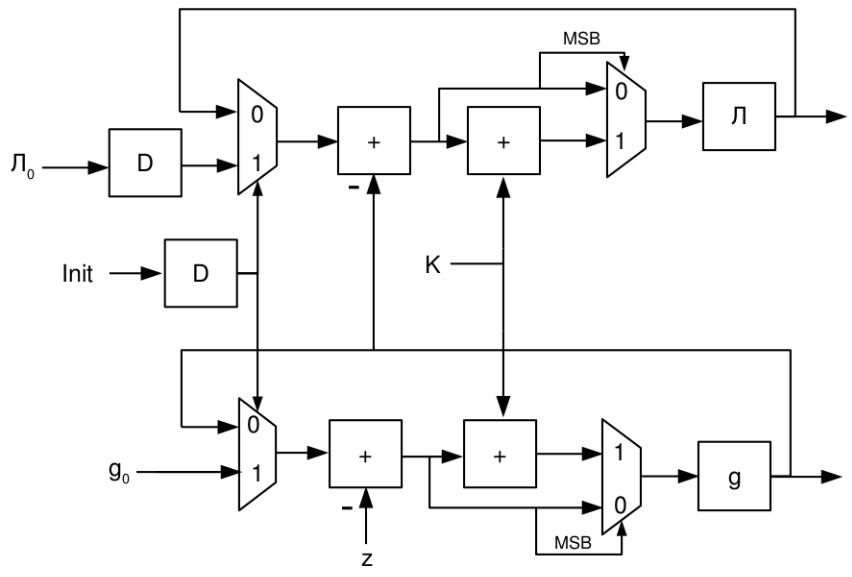


Figure 3.6: Address generator block for the backward direction

3.3 Turbo Decoder

With the max-log-MAP decoder and interleaver in place, the turbo decoder can then be implemented. The architecture of the turbo decoder that uses a single MAP decoder are shown in figure 3.7. It consists of an input memory bank which in turn consists of three memories, one for the systematic bit and one for each of the parity bits. The LLR memory stores the extrinsic information from the MAP decoder. The general structure of a turbo decoder as shown in figure 2.3 usually consists of two sub-decoders, however, the data dependency between them prevents them from being run in parallel and the turbo decoder are therefore implemented by using only one sub-decoder. During the first half of an iteration L_e are read and written from and to the memory in the consecutive order. The systematic bit and the parity1 bit are read in the same order as well. During the second half, L_e and y_s are read in the interleaved order while the parity2 bit is read in the consecutive order. This leads to the updated extrinsic LLR to be calculated in the interleaved order as well and the de-interleaving are performed simply by writing the memory using the interleaved address.

The a-posteriori LLR are only used during the last half iteration to performed the hard decision which is done during the backward recursion. Therefore no extrinsic information are stored in memory during the last half iteration.

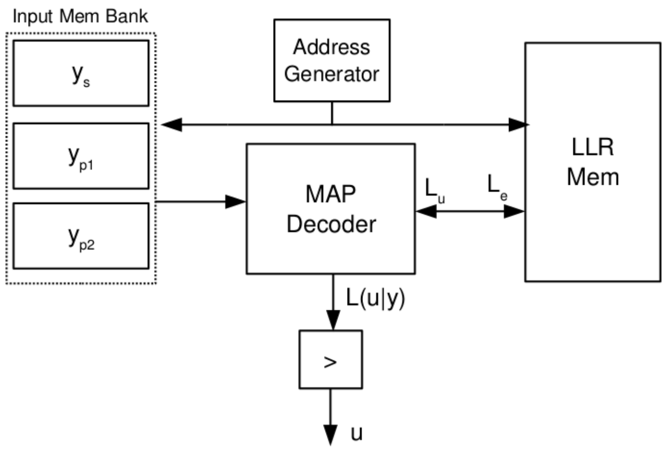


Figure 3.7: Block diagram of a turbo decoder with one MAP decoder

3.4 Parallel Structure

To increase the throughput the decoding can be performed in parallel by using multiple MAP decoders. In these implementations, the codeword is divided into equally long blocks where the number of blocks is equal to the number of MAP decoders. The input and LLR memories are needed to be split as well to prevent memory conflicts and for every added decoder an additional interleaver are required as well. If no memory conflicts occur due to the parallelization the throughput is increased by a factor equal to the number of used MAP decoders. As seen in the previous chapter each of the defined codeword sizes of LTE are even divisible by 2,4 and 8 and thus turbo decoders using these number of MAP decoders can be implemented without any memory conflicts.

If more than eight decoders are used some of them has to be deactivated if the current codeword size is not even divisible by the decoders used. Considering a turbo decoder using 32 MAP decoders, if the codeword size is less than 512 only 8 of the cores would be running while the others are idle and if the codeword size is between 512 and 1008 16 of the decoders would be active.

The architecture of a parallel turbo decoder using N MAP decoders is shown in figure 3.8. The parallel structure requires the addition of a interconnect network to route the data between memories and the MAP decoders as well as a block calculating the minimum address of the outputs of the interleavers.

3.4.1 Minimum Address Block

As seen in figure 2.15 each generated address will be separated by a factor of L and so the minimum address at each step can be used to address all the memories. The minimum address can be found using a compare-select structure whose length depends on the number of MAP decoders used. In figure 3.9 the min-

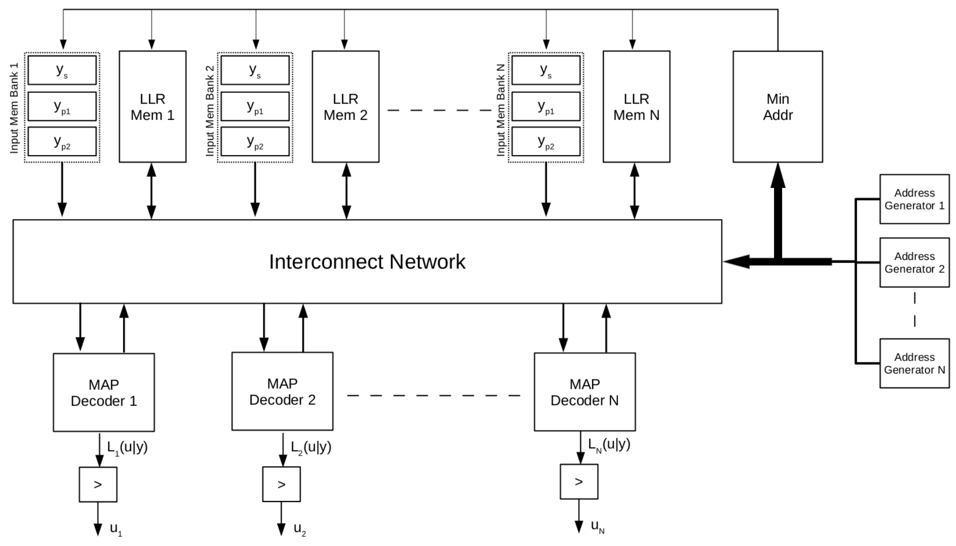


Figure 3.8: The turbo decoder using N MAP decoders.

imum address block for the case of 8 decoders are shown and three stages are needed to find the minimum of the 8 addresses. The minimum number of stages required to find the minimum address is given by $log_2(N_{MAP})$. The CS block for the minimum address block are implemented as in figure 3.10 and consists of a comparator and a mux and selects the minimum address out of the two inputs.

3.4.2 Interconnect network

During the forward recursion, the data read from the memories has to be routed to the correct MAP decoder and during the backward recursion the opposite holds. The routing is performed by a master-slave batcher network, in figure 3.11 the network for four MAP decoders are shown. The master network sorts the input addresses in ascending order by using two input sorting block implemented as in figure 3.12. If the lower input address is less than the upper the outputs are switched and the sel output is set to '1' otherwise the output and the input are equal. The slave network uses the generated select signals from the master network to apply the inverse sorting order to its input. The select block consists of two muxes as shown in figure 3.13 and just as in the sorting block the outputs are the switched inputs if the sel signal is high.

To route the LLR and systematic inputs to the MAP decoders they are provided as the input to the slave network. During the backward recursion the extrinsic LLR from the MAP decoders are provided as the input to the master network together with the generated addresses. It is the addresses that are used for the comparison and the L_e data are traveling along with the addresses.

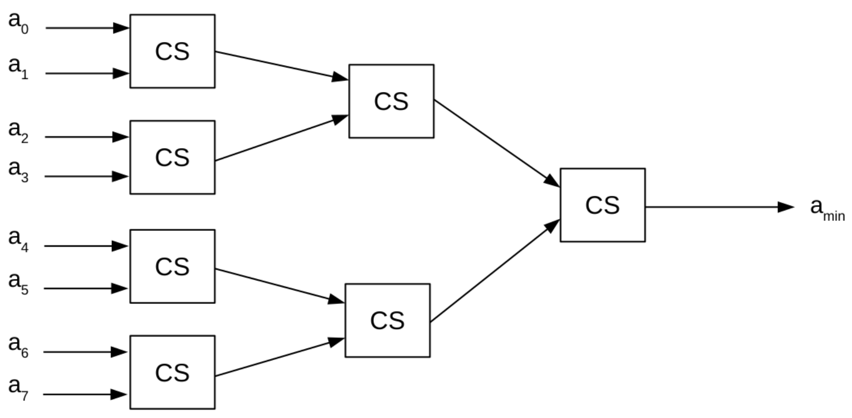


Figure 3.9: Block finding the minimum address out of eight inputs

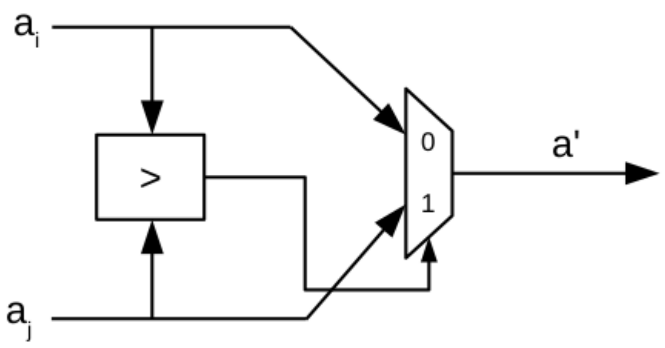


Figure 3.10: Block finding the smallest of the two inputs

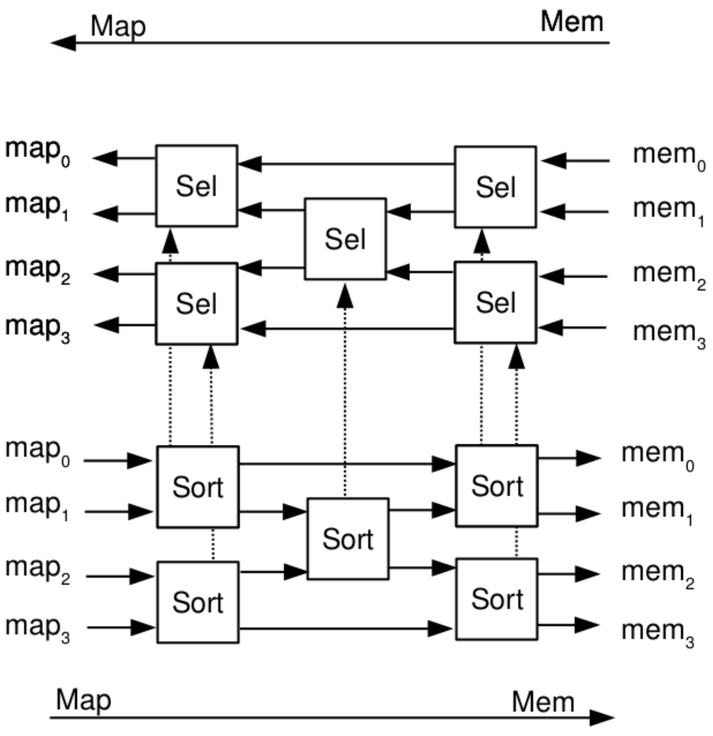


Figure 3.11: The master-slave interconnect network for 4 MAP decoders

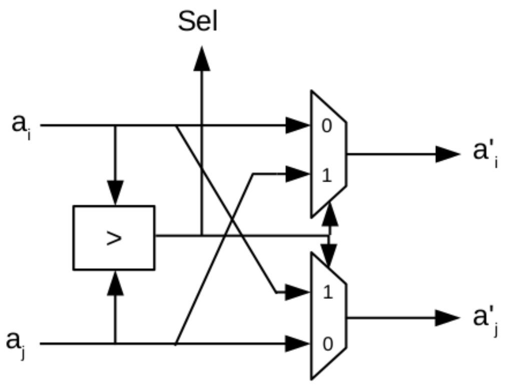


Figure 3.12: The two input sorter used in the master network. If the lower input is smaller than the upper the inputs are switched and the sel signal is set to '1'

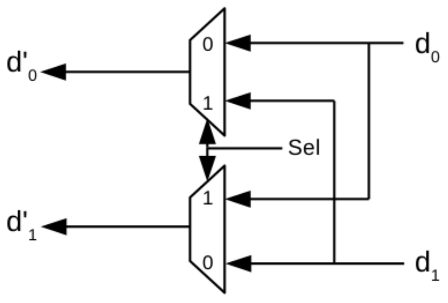


Figure 3.13: The select block in the slave network. When the input sel signal is high the input are switched

3.5 Stopping Criteria

The stopping criteria are implemented by modifying the decoders in figure 3.7 and 3.8 and the resulting architecture for the one MAP turbo decoder are shown in figure 3.14. The idea behind the stopping criteria in question is to stop the updating of the extrinsic information that is above a certain threshold and thus the decoder need a way to determine if the value should be written to the memory or not. This is achieved by attaching a status bit to every L_e and thus one extra bit are required to be stored in the LLR memory. As seen in figure 3.14 the extra hardware required, together with the extra memory bit, for the stopping criteria consists of a threshold compare unit(TCU) for every MAP decoder used and the early stopping unit. The TCU determines if the L_e shall be written to the memory or not. At the end of each half iteration, the output value of the early stopping unit is checked to determine if the decoding can be stopped. Some small changes are also done in the MAP decoder. The status bit is also provided in the input and is checked at every step. If the status bit is high then only the state metrics needs to be calculated which means that no value has to be stored in the $A\Gamma$ memory during the forward recursion and during the backward recursion no value has to read from the memory and the LLRU are idle as well.

The stopping criteria thus allow for potential lower total energy consumption in mainly two ways. Firstly, by the potentially reduced number of memory accesses to the LLR and $A\Gamma$ memory and the reduced number of calculations performed by the LLRU. Secondly, a reduced decoding time due to fewer iterations performed.

3.5.1 Threshold Comparison Unit

The status bit is updated during the backward recursion in the threshold comparison unit(TCU) which are shown in figure 3.15. If the status bit of the input is already high then nothing is done in this step and the block sets the write signal to false and nothing is written to the memory. Otherwise the absolute value of the extrinsic LLR are compared to the threshold and the result of the comparison are

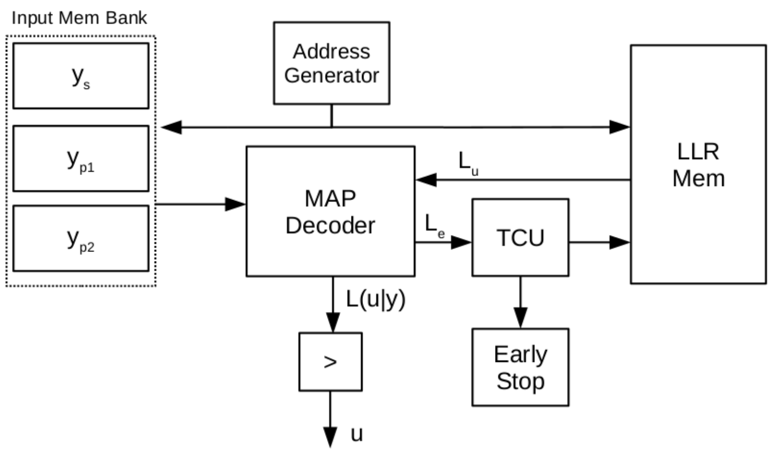


Figure 3.14: The turbo decoder with the stopping criteria.

set as the new status bit and the write signal is set to true so that the new status bit and the L_e are stored in memory.

3.5.2 Early Stopping Unit

The early stopping unit is a simple block that determines if all extrinsic values have reached the threshold in which case the decoding can be stopped. This can be implemented using a one-bit register, a two input AND gate, a N_{MAP} AND gate and a mux as shown in figure 3.16. The status bits outputted from the TCUs are set as the input to the N_{MAP} AND gate which's output are the input to the two input AND gate together with the register value. At the start of each half iteration the register is initialized to '1' and during the forward recursion it keeps its value. During the backward recursion the result of the AND operations becomes the new value of the register. If any of the L_e values haven't reached the threshold then the corresponding status bit are '0' and thus the result of the AND operation will be '0' as well while if every L_e is above the threshold the value of the register will be '1' when the end of the half iteration are reached.

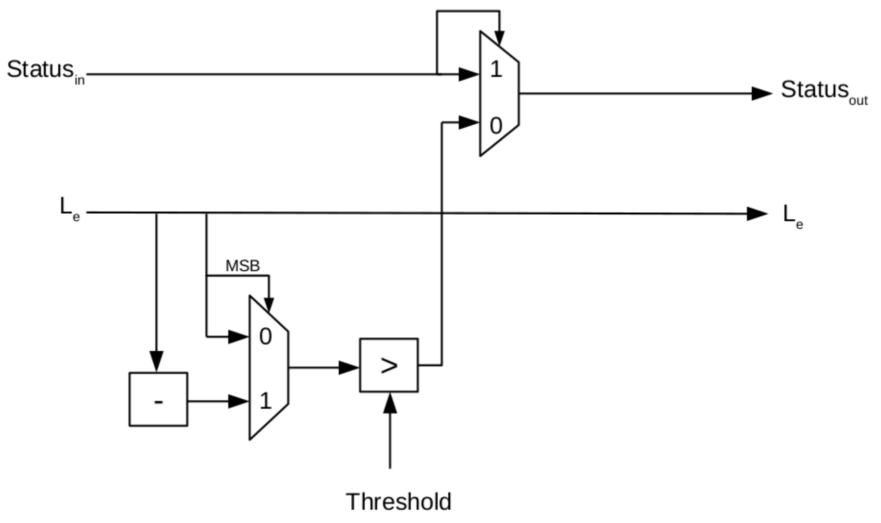


Figure 3.15: The threshold comparison unit. It compares the extrinsic LLR to the threshold and decides the next value of the status bit

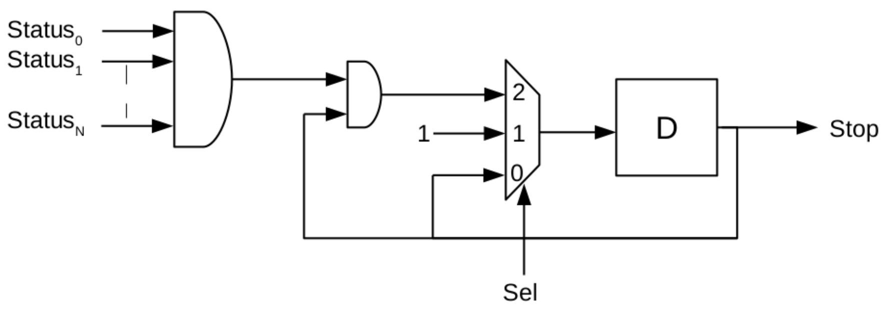


Figure 3.16: Block detecting if the decoding can be stopped

Chapter 4

Results

This work has resulted in two different implementations of an LTE compliant turbo decoder. One implementation includes the stopping criteria described in the previous chapters and the other does not use any stopping criteria. Both implementations can be configured to use 1, 2, 4, 8 or 16 MAP decoders in parallel thus reaching a maximum throughput of 110 Mbits/s running at a clock frequency of 200 MHz when performing 7 full iterations. The number of iterations, as well as the stopping threshold, can be configured at runtime.

The decoders are in the form of integer implementation using signed integers for representing the data. In this work, the soft input information is represented by 4 bits, the forward and backward metrics by 10 bits and the extrinsic LLR by 8 bits with no stopping criteria and 7 bits for the stopping criteria. The stopping criteria can thus take the values between -64 to 63.

4.1 Decoding Performance

The error correction performance is commonly measured by either the bit error rate(BER) or by the block error rate(BLER). BER measures the ratio between the number of wrongly decoded bits to the total amount of decoded bits whereas the BLER measures the ratio between information blocks with errors to the total amount of blocks. In many applications the data cannot be used if the information block contains any errors and therefore the BLER measure are used in this report. The modulation, de-modulation, rate matching and encoding have been done using the Matlab LTE toolbox and the AWGN channel have been simulated by the 'awgn' function in Matlab.

4.1.1 No Stopping Criteria

In figure 4.1 the BLER for the no stopping criteria turbo decoder are shown for different number of iterations. The simulations were performed for the native code rate of $\frac{1}{3}$ with the modulation scheme of QPSK. During the development, the turbo decoder in the Matlab LTE toolbox has been used as a reference. The result of the Matlab implementation for 8 iterations as well as no input quantization is also plotted in figure 4.1.

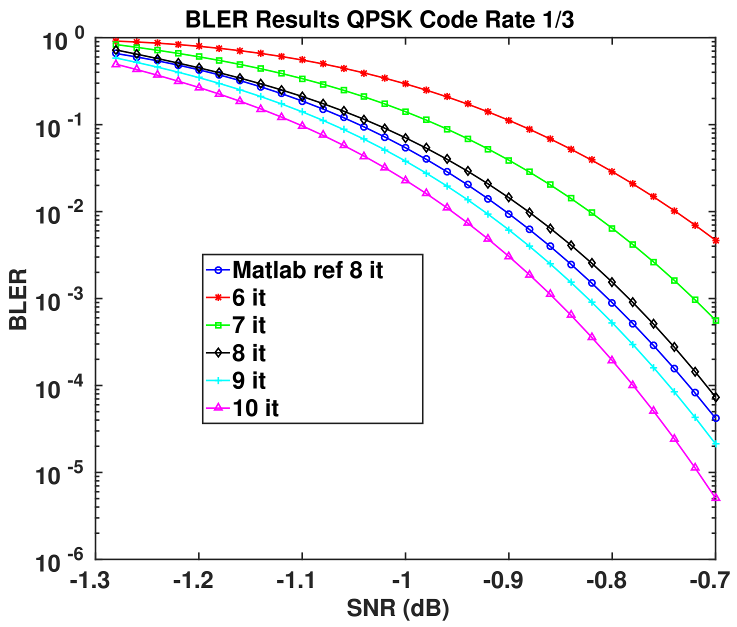


Figure 4.1: BLER for the turbo decoder without any stopping criteria for different iteration numbers for the code rate of $\frac{1}{3}$

Looking at figure 4.1 it is seen that the Matlab functional model performs a little bit better. Around the point at which the BLER is 10^{-1}, the hardware implementation performs about 0.02 dB worse than the Matlab model. With the higher input precision as well as being a purely functional model the Matlab decoder are expected to perform better thus the hardware implementation are in line with the expected performance.

From figure 4.1 it is also seen that the performance increases with each additional iteration which is expected, however it is also seen, by the decreasing distance between the curves, that the performance gained by increasing the number of iterations diminishes the higher the iterations gets. Each additional iteration also increases the decoding delay thus lowering the throughput which has to be taken into consideration when setting the number of iterations.

In the LTE standard, a code rate of $\frac{1}{3}$ with QPSK is only used in case of poor channel conditions as it provides the best error correction capabilities. In better channel conditions a better efficiency can be obtained by increasing the code rate and/or using a higher modulation scheme as 16QAM or 64QAM. In figure 4.2 the BLER for the same setup as in 4.1 are shown for a code rate of $\frac{1}{2}$. As seen in figure 4.2 increasing the code rate from $\frac{1}{3}$ to $\frac{1}{2}$ the SNR level for which the BLER is 10^{-1} is increased by around 2.4 dB. A part from the decreased error correction performance the behavior are similar to the $\frac{1}{3}$ case.

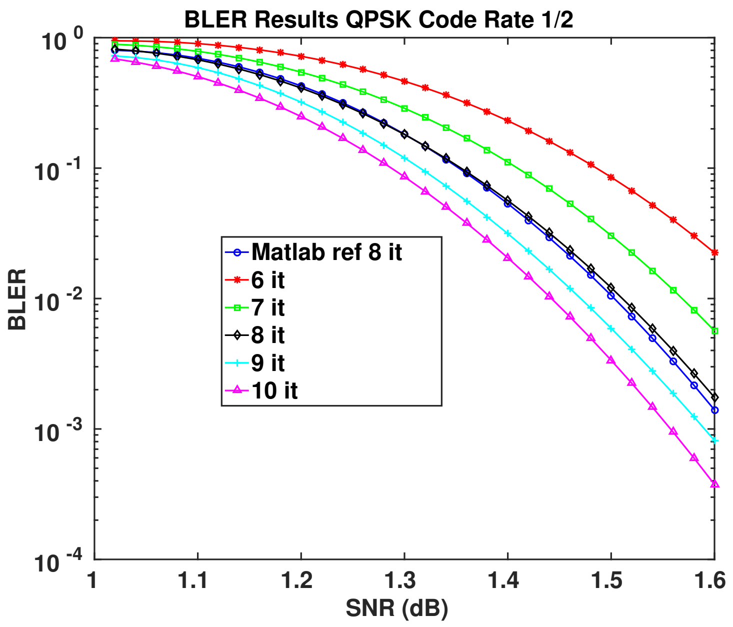


Figure 4.2: BLER for the turbo decoder without any stopping criteria for different iteration numbers for the code rate of $\frac{1}{2}$

4.1.2 Stopping Criteria

The stopping criteria under investigation will potentially lead to a reduced power and energy consumption by reducing the number of required memory accesses in conjunction with the potential to stop the decoding before completing the set amount of iterations. The effect of different thresholds on the memory accesses and performed iterations are shown in figure 4.3 and 4.4 with the SNR ranging between 2 to -0.8 dB. For these simulations, a code rate of $\frac{1}{3}$ with QPSK were used and the maximum number of iterations were sat to 7 i.e 14 half iterations.

Starting with figure 4.3 it is seen that as expected a lower threshold result in a greater reduction in the number of required memory accesses. At an SNR of 2, the required memory accesses ranges between 47 % to 53 % for the thresholds of 30 respectively 60. The amount of accesses then grows as the SNR decreases and at the SNR of 0 dB it now ranges between 51 % and 60 %. Beyond the SNR of 0 dB the growth accelerates and at an SNR of -0.8 dB the reduction in the number of memory accesses ranges between 38 % and 22 %.

Looking at figure 4.4 it can again be seen that a lower threshold results in larger savings. It is also seen that a threshold of 60 results in very low savings in performed iterations over the whole SNR range. At an SNR of 2 dB the threshold of 30 results in an average of only 5 half iterations being performed while the threshold of 50 results in 7 half iterations thus half the maximum set number of iterations. The required number of half iterations then grow as the SNR decreases

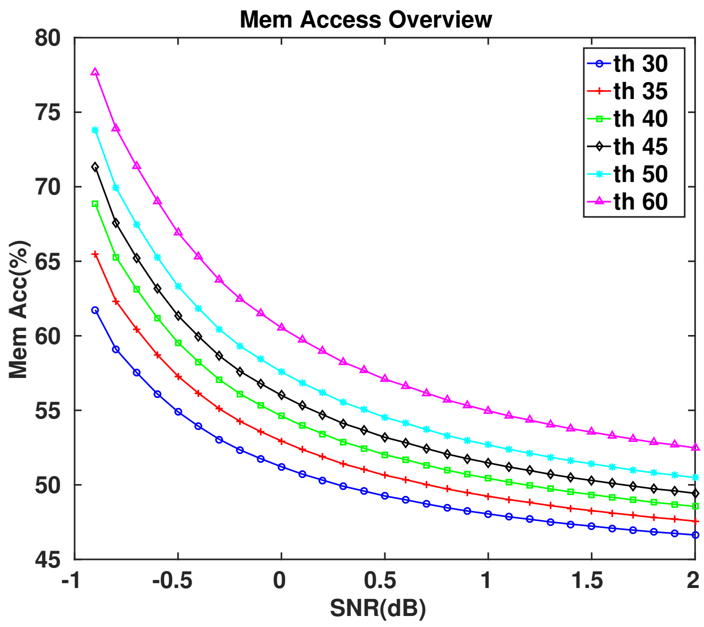


Figure 4.3: Average percentage of needed memory accesses for different thresholds for the code rate of $\frac{1}{3}$

and at -0.8 dB no reduction in the number of iterations are observed.

The BLER of the stopping criteria implementation for the same conditions as in figure 4.1 are shown in figure 4.5. The result of the no stopping criteria for 7 iterations is plotted as a reference.

Figure 4.5 shows that the lower thresholds of 30 and 35 results in a considerable performance loss at this SNR range. A threshold of 40 results in a performance loss of around 0.01 dB while for the rest of the thresholds no noticeable performance loss is observed.

In figure 4.6 the number of memory accesses are shown for the same setups as in 4.5. At this SNR range the maximum number of iterations are practically always performed as seen in figure 4.4 and are therefor not presented. However there is still gains in the amount of needed memory accesses as are shown in figure 4.6. At the SNR of -1 dB using a threshold of 45 results in a reduction of 25 % in the number of memory accesses needed and the threshold of 60 results in a reduction of 18 %.

The same simulations were also performed for the code rate of $\frac{1}{2}$ and in figure 4.8 and 4.7 the number of iterations performed and memory accesses are shown. The increased code rate makes the extrinsic LLR values to grow more slowly and the thresholds therefore have to be decreased.

As shown in figure 4.7 and 4.8 the behavior are similar to the $\frac{1}{3}$ code rate case apart from the increased SNR and reduction of the threshold magnitudes.

The BLER and average percentage of memory accesses for the different thresh-

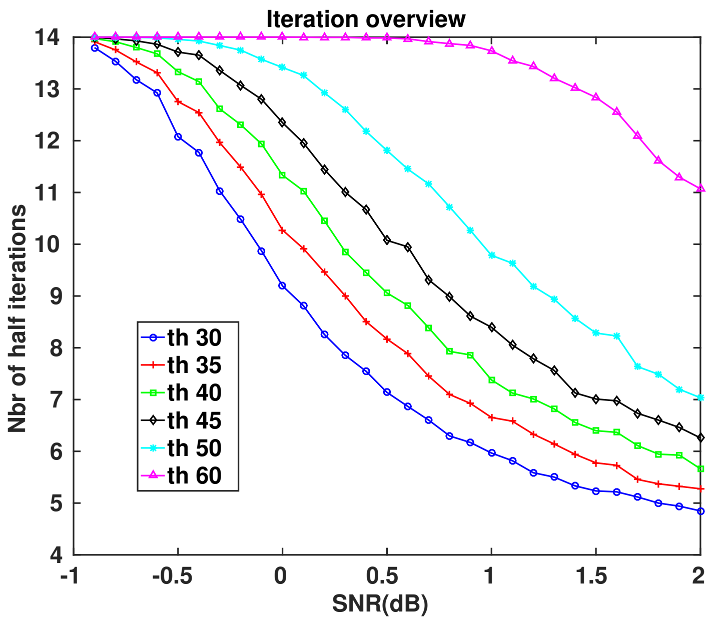


Figure 4.4: Average number of performed half iterations for different thresholds for the code rate of $\frac{1}{3}$. Maximum number of half iterations are 14

olds are shown in figure 4.9 respectively figure 4.10 for the code rate $\frac{1}{2}$.

The threshold of 20 results in a considerable performance loss while higher thresholds results in a very small or no performance loss as shown in figure 4.9.

4.2 Area Numbers

The high-level synthesis of the different configurations was performed in Catapult using 65 nm libraries from ST Microelectronics. The RTL output was then synthesized in Design Vision using low power 65 nm libraries. With the available memories limited the synthesis were performed without memories and the area were extrapolated from the closest matching memories. The synthesis was performed for the clock frequency of 200 MHz for all configurations.

In table 4.1 the area figures for the no stopping criteria implementation are presented for different numbers of MAP decoders. The area of the logic area i.e excluding the memories for the 32 and 64 MAP decoder configurations are extrapolated using a ratio of 1.95 which is the ratio between the 16 MAP and 8 MAP configuration.

Using 16 MAP decoders results in an area of around 1.5 mm^2 at 200 MHz. From table 4.1 it is seen that the main part of the area is occupied by memory. For the 1 MAP configuration, the memory occupies around 87 % of the total area, for

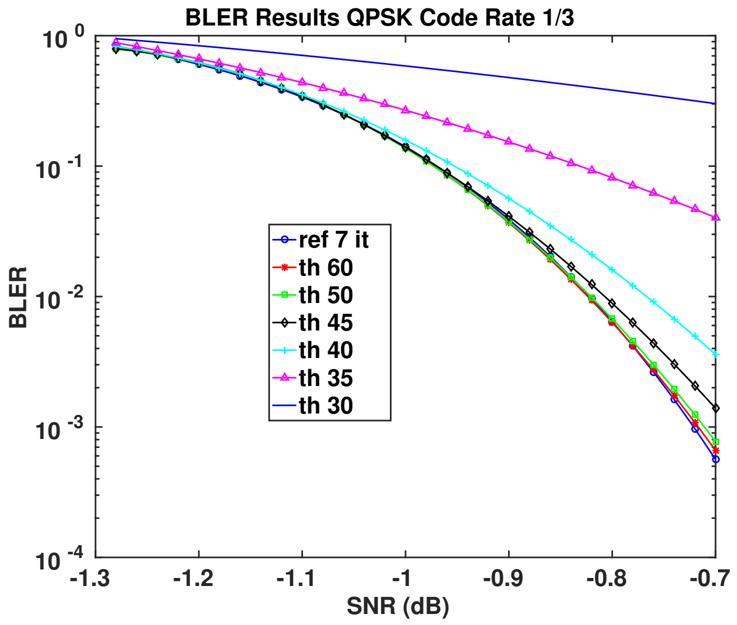


Figure 4.5: BLER for the stopping criteria turbo decoder for different thresholds. The code rate are $\frac{1}{3}$

the 2 and 4 MAP configurations around 80 % are occupied by memory and for all other configurations around 70 % of the area are occupied by memories.

In table 4.2 the area numbers for the stopping criteria are shown as well as the area penalty compared to the corresponding configuration without any stopping criteria. The amount of memory is the same as for the implementation without any stopping criteria.

Table 4.2 shows that excluding the memories the stopping criteria leads to an estimated area penalty of around 5 to 8 % depending on the number of MAP decoders used. When the memories are also included the resulting area penalty are less than 3 % for all configurations.

4.3 Throughput

The throughput numbers for the different configurations running at 200 MHz for 7 iterations are shown in table 4.3. At this frequency and iteration numbers, 16 MAP decoders are needed at minimum to reach the target throughput of 100 Mb/s. The throughput scales linearly with the frequency and the number of employed MAP decoders.

If higher throughputs are needed the design is synthesizable at 400 MHz when using general purpose libraries thus doubling the throughput and achieving a maximum throughput of 879.6 Mb/s if 64 MAP decoders are used and only

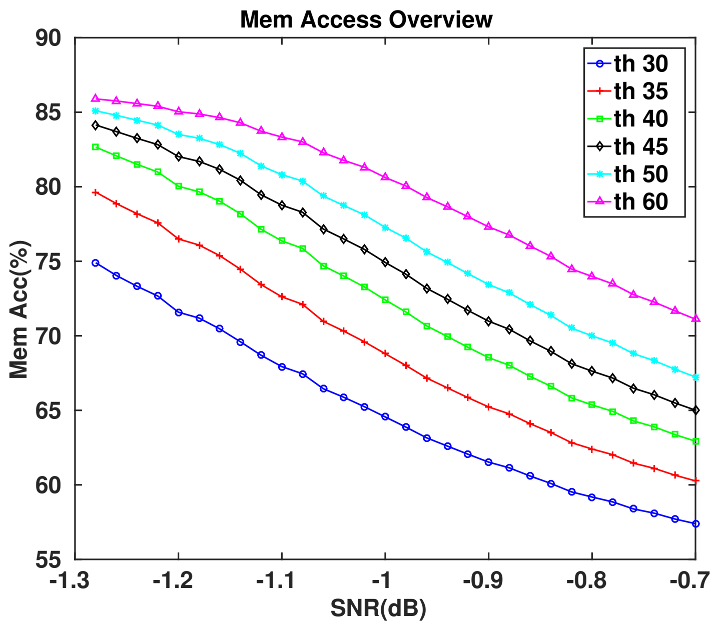


Figure 4.6: Average percentage of memory accesses for the different thresholds at low SNR and a code rate of $\frac{1}{3}$

8 MAP decoders are needed to reach the 100 Mb/s target.

4.4 Power Simulations

The power simulations were performed for the 8 MAP configurations for both implementations. To match the memory requirements with the available memories the sliding window length were increased from 64 to 128. The simulations were performed for the code word size of 3072 for 7 iterations over a time equal to the time required for the no stopping criteria implementation to finish. The average consumed energy for 7 different SNR levels are shown in figure 4.11 for a stopping threshold of 45.

As figure 4.11 shows the greatest reduction in the energy consumption are achieved at larger SNR levels as expected from the memory access result in 4.6. In the best case scenario(1.5 dB) a reduction in energy consumed by 63 % is achieved and at an SNR of -1 dB a reduction of almost 17 %. At the SNR of -1.5 dB only a small reduction in are observed. However, at this low SNR, the BLER is as seen in figure 4.5 almost equal to one and practically no code word are decoded correctly.

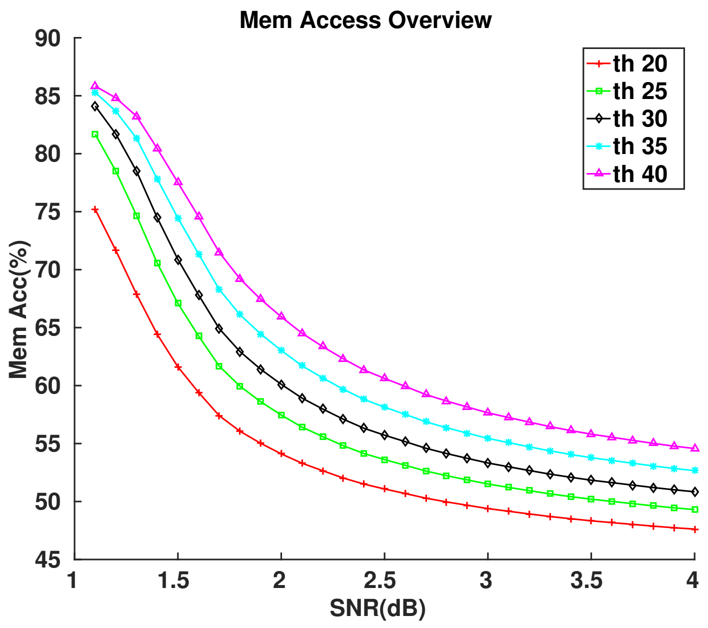


Figure 4.7: Average percentage of needed memory accesses for different thresholds for the code rate of $\frac{1}{2}$

Nbr MAP Dec	Area(mm^2)(no mem)	Est Area(mm^2)	ratio(to 1 MAP)
1	0.032	0.244	1
2	0.068	0.365	1.5
4	0.122	0.562	2.3
8	0.237	0.876	3.6
16	0.462	1.525	6.3
32	0.902	2.813	11.5
64	1.759	5.405	22.1

Table 4.1: Area numbers of the no stopping criteria implementation for different configurations synthesized at 200 MHz

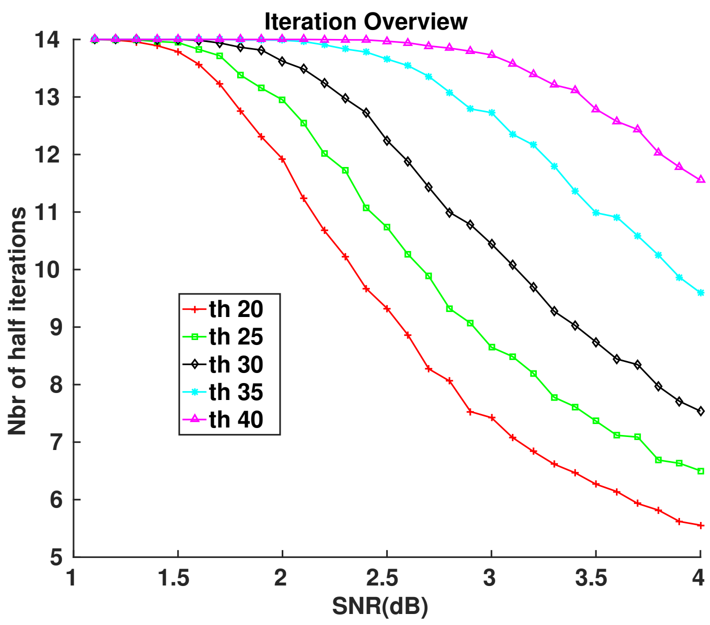


Figure 4.8: Average number of performed half iterations for different thresholds for the code rate of $\frac{1}{2}$. Maximum number of half iterations are 14

Nbr MAP Dec	Area(mm^2)(no mem)	Est Area(mm^2)	Area penalty(%)(no mem/mem)
1	0.033	0.245	4.7/0.4
2	0.069	0.366	1.3/0.3
4	0.128	0.569	4.9/1.2
8	0.255	0.894	7.6/2
16	0.498	1.561	7.8/2.4
32	0.971	2.882	7.6/2.5
64	1.893	5.483	7.6/1.5

Table 4.2: Area numbers of the stopping criteria implementation for different configurations synthesized at 200 MHz

Nbr MAP Dec	1	2	4	8	16	32	64
Throughput (Mbits/s)	6.9	13.7	27.5	55	110	219.9	439.4

Table 4.3: Throughput numbers for the different configurations at 200 MHz for 7 iterations using low power libraries

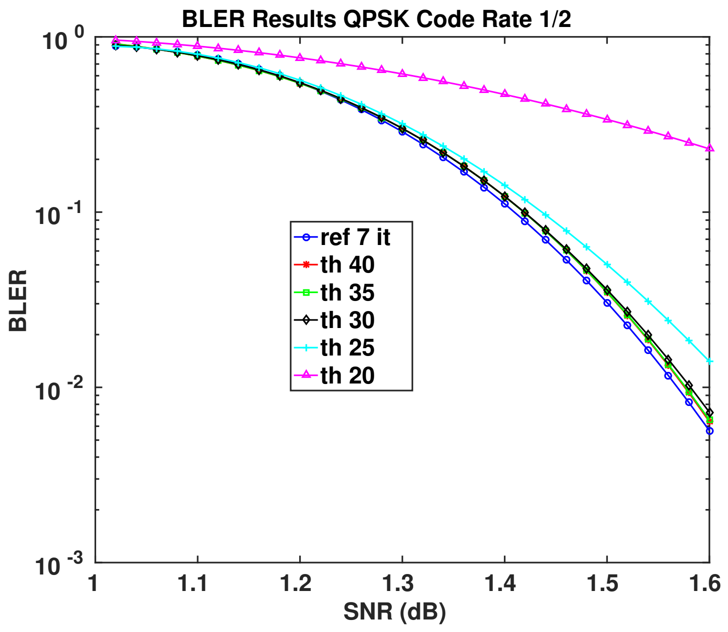


Figure 4.9: BLER for the stopping criteria for different thresholds for the code rate of $\frac{1}{2}$

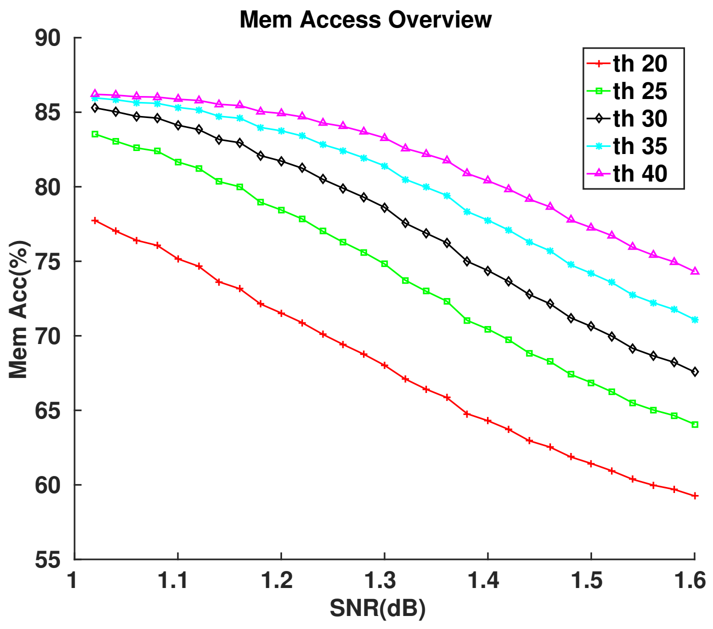


Figure 4.10: Average percentage of memory accesses for the different thresholds at low SNR with a code rate of $\frac{1}{2}$

Nbr MAP Dec	1	2	4	8	16	32	64
Throughput (Mbits/s)	13.75	27.5	55	110	219.9	439.8	879.6

Table 4.4: Throughput numbers for the different configurations at 400 MHz for 7 iterations using general purpose libraries

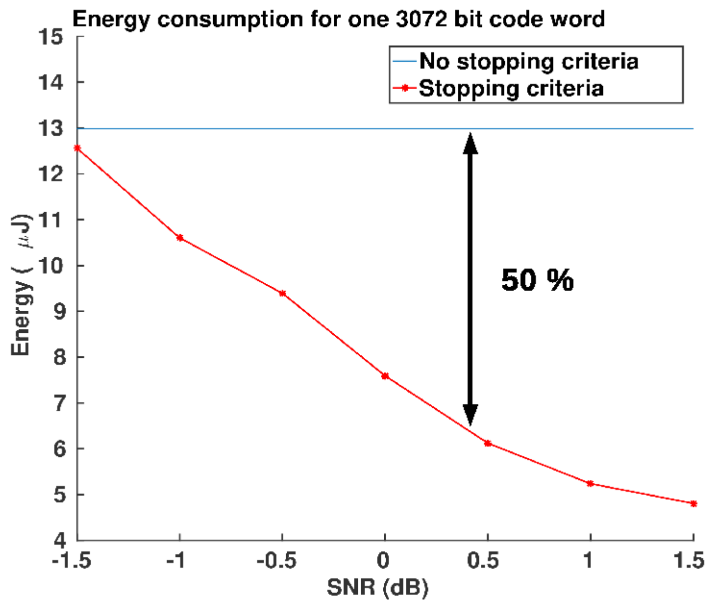


Figure 4.11: Energy consumption of one 3072 bit code word for both implementations with the stopping threshold set to 45.

Chapter 5

Discussion Analysis

The resulting stopping criteria implementation are summarized together with some existing turbo decoder implementations in table 5.1. With different implementations using different technologies and clock frequencies etc. the numbers can't be compared directly but they can provide a hint of the relationship between the implementations.

In [7] a turbo decoder using up to 64 MAP decoders using a 65 nm technology are presented. The maximum throughput is 1.2 GBits/s for 6 iterations at 400 MHz with an area of 8.57 mm^2. It also presents results for other MAP configurations and clock frequencies and in table 5.1 the numbers for the 16 MAP are used. In [9] a decoder with up to 8 MAP decoders with a throughput of 128 Mbits/s at 8 iterations implemented in 90 nm technology are presented. A turbo decoder using 8 radix 4 MAP decoders in 130 nm technology are presented in [8].

Since the implementation in [7] also uses 65 nm technology it is most suited for a comparison. The throughput for the implementation in this work for 6 iterations are 128 Mbits/s at 200 MHz and from table 5.1 it is seen that the decoder in [7] have a higher throughput for the same number of MAP decoders and equal clock frequency. This is due to a difference in the architectures. In [7] the forward recursion of the next sliding window are performed in parallel with the backward recursion of the current window. Thus when the backward recursion is complete the forward recursion of the next window are already finished and the backward recursion can be started immediately. This leads to a higher throughput compared to the architecture in this work where the recursions are performed one after another. Using the architecture in [7] the throughput are approximated as

$$Throughput \approx \frac{K * f}{I * (K/P + W)}$$

where K is the code word length, f is the frequency in MHz, I the number of performed half iterations, P the parallelism degree and W the length of the sliding window. Thus the throughput depends on the chosen window length. For the architecture in this work the throughput can be estimated as

$$Throughput \approx \frac{K * f}{2 * I * K/P}$$

which are independent of the window length. It should be noted that the difference in throughput between the two architectures decreases the more MAP

	This work	[7]	[9]	[8]
Standard	LTE	LTE	LTE	LTE
Radix	2	2	2	4
MAP Decoders	16	16	8	8
Technology	65 nm	65 nm	90 nm	130 nm
Clock Frequency	200 MHz	200 MHz	275 MHz	355 MHz
Throughput	128 Mbits/s (6 it)	230 Mbits/s (6it)	129 Mbits/s (8 it)	390 Mbits/s (5.5 it)
Area	1.6 mm^2	2.1 mm^2	2.1 mm^2	3.57 mm^2

Table 5.1: Comparisons to implementations in previous work

decoders that are employed since the difference between K/P and W decreases. For P equals 64 the decoder in [7] have a throughput of 640 Mbits/s at 200 MHz and the decoder in this work would have a throughput of around 512 Mbits/s.

To achieve the increased throughput the architecture in [7] requires an additional BMU, alternatively memory storing the branch metrics, and an extra SMU to be able to perform a forward and backward recursion in parallel. Furthermore, all the memories are required to be dual port memories since a read and write operation are required in each clock cycle. This should lead to an increased area which table 5.1 also shows. For 16 MAP decoders the area penalty of [7] are around 0.5 mm^2 and for 64 MAP decoders it is around 0.7 mm^2 or 1 mm^2 without the stopping criteria.

5.1 Stopping criteria

The main purpose of this thesis was to investigate the potential savings of the presented stopping criteria and it's potential drawbacks. In this section the effects of the stopping criteria on the performance, power/energy consumption and area will be discussed. The discussion will mainly focus on the case where the code rate of 1/3 is used which is done since the behavior at higher code rates are similar.

5.1.1 Performance VS Savings

In figure 4.5 that shows the BLER for different thresholds it is seen that for the case of a code rate of $\frac{1}{3}$ using the stopping thresholds of 50 and 60 leads to no visible performance degradation. The threshold of 45 leads to a very small performance loss at SNR levels above -1 dB while for lower SNR no loss is observed. The threshold of 40 results in a similar behavior with a more noticeable loss at lower SNR. Figure 4.5 also shows that thresholds of 35 and 30 leads to a considerable loss in the performance of the decoder and are thus not viable options at least for low SNR levels.

In figure 4.6 it is seen that a lower threshold result in a greater reduction in the number of performed memory accesses and figure 4.4 shows that the same holds for the number of performed iterations. However, a reduction in the number of performed iterations is only observed at SNR levels above -0.5 dB. For the memory accesses, a difference of the threshold magnitude of 5 results in a difference in the number of memory accesses of around 2 %. The implementation of the stopping criteria thus makes it possible to trade performance for a greater reduction of the energy consumption.

As there is no difference in the performance between the thresholds of 50 and 60 there is no real reason to chose 60 since the threshold of 50 leads to greater savings in the amount of memory accesses. Furthermore the threshold of 45 only results in a very small performance loss for SNR levels above -1 dB and might thus be an even better alternative in most cases.

As the SNR increases the BLER decreases and at some point, the BLER will become so small that even lower thresholds than 45 can be used with no loss in practice. The most optimal threshold thus depends on the channel conditions and should be set accordingly to achieve the best results. Still, fixing the threshold to 45 when the code rate of $\frac{1}{3}$ are used would result in a reduction in the amount of required memory accesses of between 15 % and 45 %.

5.1.2 Power and Energy

As seen in figure 4.11 using a threshold of 45 leads to a reduction in the energy consumption of around 17 % at the SNR of -1 dB. At the SNR of 0 dB the reduction in energy consumption has grown to around 40 % and around the 0.5 dB point of the reduction is almost 50 %. The stopping criteria thus introduce considerable savings in the amount of energy consumed by the turbo decoder with no or only a very small performance loss as previously discussed.

It shall be kept in mind that these simulations were performed using a sliding window length of 128, instead of the proposed 64, to be able to match the memory requirements of the implementation with the available memory sizes. The implementation thus uses a larger amount of memory which should lead to a higher power consumption. Therefore the amount of consumed energy is expected to be larger than if a sliding window of 64 were to be used and it is therefore the difference between the implementations that shall be compared and not the actual number.

The simulations were also done using the configuration with 8 MAP decoders running at 200 MHz which as seen in table 4.3 does not meet the throughput requirement of 100 Mb/s. The were several reasons that the 16 MAP configuration were not used. One was that it was not possible to match the memory requirements perfectly with the memories, another is the time requirements. Performing the high-level synthesis and RTL synthesis for 16 MAP takes several hours thus the turn around time for correcting problems are large. Furthermore, the memories used limits the maximum clock frequency to around 200 MHz thus it was not possible to reach the 100 Mb/s throughput for 8 MAP decoders.

5.1.3 Area

In table 4.2 it is seen that the estimated area penalty of the stopping criteria are less than 3 % for all configurations when the area of the memories is included. It should be noted that the area numbers presented are post-synthesis and the area of the memories have been estimated thus they are an estimation of the real area which would be obtained after performing the physical layout.

Excluding the memory area, table 4.2 shows that the area penalty due to the stopping criteria is less than 10 % for all configurations. Considering the potential savings in energy consumption ranging between 17 % to almost 50 % this area penalty is a small price to pay.

5.2 Conclusions

In this thesis an implementation of an early stopping criterion for a turbo decoder has been presented. The implementation of the stopping criterion can be configured so that no performance loss are observed compared to an implementation with no stopping criterion. Yet the energy consumption can be reduced by up to 50 % in good channel conditions while still providing a reduction of between 10 to 20 % for low SNR levels. The implemented stopping criterion only leads to a small area penalty of a few percent. Thus the presented stopping criterion can be implemented with the result of no performance loss while still providing a considerable reduction in the energy consumption and a small area penalty, all of which makes it an attractive choice.

5.3 Further Work

A continuation of this work would be to add support for the use of 32 and 64 MAP decoders in parallel to increase the throughput. Another part would also be to put the implementation through the whole ASIC design flow to get more accurate estimations on the area as well as the power and energy consumption. This would require access to a wider range and variety of memory sizes to allow for an as efficient implementation as possible. Another extension would also be to investigate the effect of the stopping criteria for other architectures like the one presented in [7] as well as the effect if a radix 4 architecture would be used. Another thing would also be to perform the power simulations for different thresholds as well as for different MAP configurations.

References

[1] 3GPP TS 36.212 V10.6.0 (2012-06)

[2] C.Berrou, A.Glavieux, P.Thitimajshima, Near shannon limit error-correcting coding and decoding: turbo-codes, in: IEEE International Conference on Communication, May 1993, pp. 1064 - 1070

[3] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, Optimal decoding of linear codes for minimizing symbol error rate, IEEE Trans. on Information Theory , pp. 284 - 287, March 1974.

[4] Silvio A. Abrantes, From BCJR to turbo decoding : MAP algorithms made easier, April 2004

[5] S Dolinar, D Divsalar, Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations, TDA Progress Report 42-122

[6] Johan Hokfelt, Ove Edfors, Torleiv Maseng, Turbo codes: correlated extrinsic information and its impact on iterative decoding performance, Vehicular Technology Conference, 1999 IEEE 49th pp. 1871 - 1875 vol.3

[7] Yang Sun , Joseph R. Cavallaro Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder, INTEGRATION, the VLSI journal 44 (2011) 305 - 315

[8] Christoph Studer, Christian Benkeser, Sandro Belfanti, Qiuting Huang, Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE, TO APPEAR IN IEEE JOURNAL OF SOLID-STATE CIRCUITS 2011

[9] C.-C. Wong, H.-C. C. Y.-Yu Lee, A 188-size 2.1mm^2 reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system, in Symp. VLSI circuits dig. tech. papers, Kyoto, Japan, June 2009, pp. 288 - 289

[10] M Andrei, L Trifina, D Tarniceriu, INFLUENCE OF EXTRINSIC INFORMATION SCALING FACTOR ON MAX-LOG-MAP DECODING ALGORITHM FOR TURBO CODES WITH TRANSMISSION ON CHANNEL AFFECTED BY MIDDLETON CLASS-A IMPULSIVE NOISE

[11] C. Bernard Shung, Paul H. Siege1, Gottfried Ungerboeck, Hemant K. Tliapar, VLSI Architectures for Metric Normalization in the Viterbi Algorithm

[12] J. Sun, O.Y. Takeshita, Interleavers for turbo codes using permutation polynomials over integer rings, IEEE Trans. Inform. Theory 51 (January) (2005) 101 - 119.

[13] Jorge Castinera Moreira, Patrick Guy Farrell, Essential of Error-Control Coding, chapter 1 p. 22 - 26

[14] Jorge Castinera Moreira, Patrick Guy Farrell, Essential of Error-Control Coding, chapter 6 p. 189 - 192