Contents

List of Figures

List of Tables

Abstract

Current and future digital communication standards highlight use of error correcting codes for reliable information transmission. In modern telecommunication applications, it may be desirable to have a flexible system able to support a variety of error correction standards or a variety of codes. Particularly, in software defined radio (SDR) paradigm where flexibility is a fundamental property of future receivers.

Thus in this thesis, we focused our research on the design of high performance flexible decoder architectures. In recent years many research activities have emerged proposing multiprocessor system on chip (MPSoC) implementations in order to achieve flexible and high throughput parallel iterative decoding. In such parallel architectures two building blocks are: processing elements (PEs) capable of supporting multiple codes and the interconnection network for iterative exchange of extrinsic information across these PEs.

The aim of the thesis was two fold: firstly to design and implement a low complexity flexible PE supporting different channel decoding algorithms (Viterbi, Turbo and LDPC) with maximum hardware reuse. As a subset to this goal some sub-optimal methodologies for low power low cost turbo decoder implementation were investigated. Furthermore, we presented a VLSI complexity analysis of datapath sharing across different FEC code families. In addition, we presented a novel parallel implementation of check node computations ("Tree-Way") using Min-sum algorithm for LDPC decoding, which is optimized for maximun reuse of turbo decoding kernel.

As the secondary goal interconnection network for iterative exchange of extrinsic information across these PEs was evaluated within the paradigm of "Intra-IP" communication. As a case study traffic pattern of MPSoC based iterative decoder over a 2-D Torus/Mesh network on chip (NoC) was evaluated to enable designers choose a right kind of NoC resources for achieving application specific throughput requirement.

Finally our quest for low power, low cost architectural solutions led us to find ways for a faster performance evaluation of sub-optimal choices we had to make. Given the multitude of options to be tested, traditional error performance evaluation present a bottleneck in terms of their longer simulation time contributing to significant increase in final design time. For this purpose a new metric for fast and efficient performance evaluation of iterative decoding algorithms was investigated. It was based on the estimation of distance like function between the *a-posteriori* probability (APP) decoded symbol of optimal and suboptimal decoding algorithms.

Chapter 1

Introduction and Overview

Contents

This thesis shall examine error control techniques from the point of view of VLSI designers; that is the individual who wishes to design and implement these techniques on a hardware platform for different telecommunication applications. Digital communication systems benefit from at least one form of error correction coding algorithms. Forward Error Correction (FEC) coding improves data reliability by introducing redundant information into a data sequence prior to transmission or storage which enables a receiver to detect and possibly to correct errors without requesting retransmission of the original information. There are many FEC codes widely used in wireless and wired communications like Reed-Solomon (RS), product, convolutional, Turbo and LDPC. However

this thesis deals only with latter three, while viterbi decoding algoritms for decoding convolutional codes have been used in communication systems for quite a long time due to their simple struture and easy implementability, only in 1993 at the IEEE International Conference on Telecommunications, C. Berrou and A. Glavieux presented a new scheme for channel codes: the turbo codes, and their associated turbo decoding algorithm [Berrou et al., May 1993]. Turbo codes made possible to get within a few tenth of dB away from the Shannon limit, for a bit error rate of 10^{-5}. Beside the major impact that turbo codes have had on telecommunication systems, they also made researchers realize that other capacity approaching codes existed. Hence, the low-density parity-check (LDPC) codes invented in the early sixties by Robert Gallager [Gallager, Jan. 1962], have been resurrected in the mid nineties by David MacKay [MacKay and Neal, 1995]. Like turbo codes, LDPC codes can also get very close to the Shannon limit by the mean of an iterative decoding. In the recent years there has been a growing need for devices that have the flexibility to support multiple modes in the emerging wireless standards. For example, the 802.16 standard supports high data rates with a variety of channel coding options. The mandatory scheme is a convolutional code. Convolutional turbo codes, turbo product codes, and in 802.16e, Low Density Parity Check (LDPC) codes are optional [802.16e, 2004]. It is imporatant to note that the design cost, and the requirements of the novel communication standards encourage the designers to exploit flexible structure which can be easily adapted to different working conditions. In the rest of this work, we will analyze and design flexible architectures for high performance communications, in particular we will focus on high performance channel decoders. Nevertheless, some basic consideration on the behavior of these decoding algorithms are required in order to define a unified notation.

1.1 Overview of the Thesis

This introductory chapter will conclude in next sections with a detailed presentation of three main error correction coding algorithm viz. convolutional, turbo and LDPC codes. SISO (Soft input soft output) decoding algorithm which will be used throughout this thesis for turbo codes is also derived using a real world example of codes used in WiMAX standard. On the decoding of LDPC codes an overview is presented of used iterative decoding algorithms with special emphasis on low complexity check node computations.

Chapter 2 will present the context of this research work in relation to the state of art in the field of a flexible FEC decoder design. A problem space of flexible channel decoder design is explored and contribution to the different subset of this space are highlighted. The problem space deals with not only the implementation issues rather it also incorporates algoritm design, optimization and validation aspects.

Chapter 3 introduces the research work peformed in the domain of algorithm optimization specifically on Turbo codes. The emphasis here is to analyze the performance area tradeoff by

applying innovative optimization techniques on an already existing design of turbo decoder. Our algorithm has a good performance complexity trade-off since it significantly reduces the complexity of the extrinsic memories in turbo decoder implementations, while keeping the bit error rate close to the optimal one.

Chapter 4 evaluates NoC (Network on Chip) for MPSoC (Multiprocessor System on Chip) based iterative decoding architectures. As a case study, we evaluate the performance of our MPSoC architecture for a 2-D Torus/Mesh interconnect topology. Evaluation results are presented based on the communication centric parameters that include network latency, network size, message injection rate etc. and can be extended to any other System on Chip (SoC) interconnect topology without loss of generality.

Chapter 5 presents hardware reuse in processing elements of a flexible FEC decoder. As an experimental work hardware reuse potential of different existing implementation schemes for check node processing in LDPC decoding is explored over a Max-Log-Map based turbo decoder datapath architecture. Furthermore, we also derived a novel parallel implementation scheme("Tree-Way") for check node processing, which fits very well with underline turbo decoding architecture and results in a faster check node computation.

Chapter 6 explores a new paradigm of performace evaluation of sub optimal iterative decoding algorithms. In this work a new metric for a quicker performance evaluation of iterative decoding algorithms is proposed. It was based on the estimation of distance like function between the *a-posteriori* probability (APP) decoded symbol of optimal and suboptimal iterative decoding algorithm. We apply the notion of entropy to evaluate this function.

A conclusion and some future perspectives are finally given at the end of this thesis.

1.2 Model of a Communication system

A simple model which emphasizes the role of error control in digital communication system is shown in fig 1.1 This model consists of a digital source, an error-control encoder, a noisy channel, an error control decoder and a sink or user, which are connected in cascade as shown. For this sytem all information transferred between system blocks must be in digital form (usually "1" and "0") in order for error control to be utilised in the manner shown. Since the output of the message encoder must be suitable for input to the error-control encoder, this output must certainly be digital. Thus if the message generator produces analog output, the message encoder must as a minimum posses analog to digital capability. Once the message has been digitized we introduce redundancy in the error control encoder. The redundancy is added at a fixed rate and in a known way which introduces structure to the error-control encoder output in the form of block, convolutional or a combination of these codes. The modulator maps the encoded digital sequences into a train of short analog waveforms suitable for propagation. Modulation can be performed by varying the amplitude, the

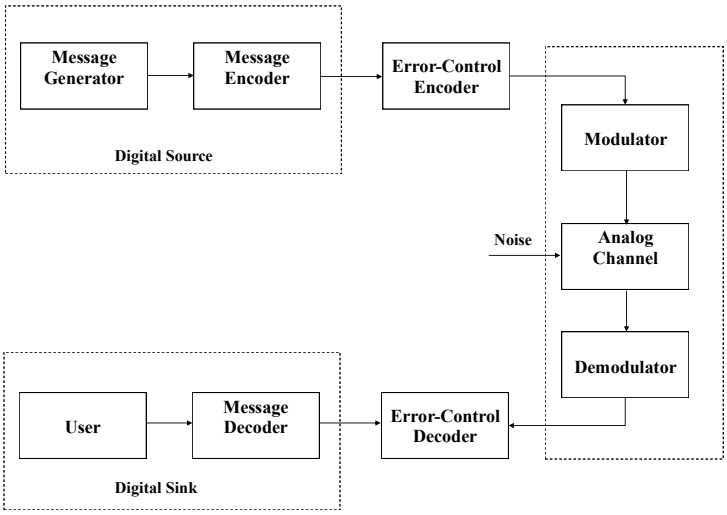

Figure 1.1: Model of Communication System

phase or the frequency of a sinusoidal waveform called the *carrier*.

Channels are transmission media used to carry or store information. Channel examples are wire lines, microwave radio links over free space, satellite links, fiber optic channels, magnetic recording media etc. Two major limitations of real channels are thermal noise and finite bandwidth. In the receiver, the demodulator typically generates a digital sequence at its output as the best estimates of the transmited codeword. The channel decoder makes estimates of actually transmitted message. The decoder process is based on the encoding rule and characteristics of the channel. The goal of the decoder is to minimize the effect of channel noise.

One important categorization is usefull to make here is; if the demodulator makes hard decisions and its output is a binary sequence; the subsequent channel decoding process is called hard decision decoding. Hard decisions in the demodulator result in some irreversible information loss. An alternative is to quantize the demodulator output to more than two levels or take samples of the analog received baseband signal and pass it on to the channel decoder. The subsequent decoding process is called soft decision decoding.

1.3 The Shannon limit

The aim of every digital communication system is to transmit more data as possible with little or no error spending a reduced amount of power. The signal bandwidth is a measure of its speed. The signals that change quickly in time have large bandwidth. On the other hand, every communication system has a limited bandwidth due to capacitances and inductances which prevent instantaneous change of signals. The system bandwidth B limits the speed of signal variations. For a given channel, there is a upper limit on the data rate related to signal to noise ratio and the system

bandwidth. In 1948 by Claude Shannon in his seminal work [Shannon, July 1948],[Shannon, Oct. 1948] proposed the concept of *channel capacity* C, as the maximum rate at which information can be transmitted over a noisy channel. For an additive white gaussian noise (AWGN) channel C is given by the formula

$$C = B \cdot log_2 \left(1 + \frac{S}{N}\right) bits/sec. \tag{1.1}$$

Shannon's channel coding theorem guarantees the existance of codes that can achieve arbitarly small probability of error if the data transmission rate r_b is smaller than the channel capacity. This fundamental result showed that noise sets a limit on the data rate but not on the error probability as widely believed before and resulted in research of finding explicit methods, called codes, for increasing the efficiency and reducing the net error rate of data communication over a noisy channel to near the limit that Shannon proved is the maximum possible for that channel. These codes can be roughly subdivided into data compression (source coding) and error-correction (channel coding) techniques. In the latter case, it took many years to find the methods Shannon's work proved were possible. While data compression removes as much redundancy as possible, an error correcting code adds just the right kind of redundancy needed to transmit the data efficiently and faithfully across a noisy channel.

To conclude Shannon limit of a communication channel is the maximum rate supported by this channel for information transfer. Since this discovery, many scientists and engineers have tried to get as close as possible to this limit. Though the theorem does not indicate how to design specific codes achieving maximum possible data rate at arbitarly small error probabilities, it motivated the development of a number of error control techniques.

1.4 Convolutional code overview

Convolutional codes have been widely used in applications such as space and satellite communications, celluar mobile, digital broadcasting etc [MacWilliams and Sloane, 1978], [Roman, 1992]. Their popularity stems from their simple structure and availabilty of easily implementable maximum likelihood soft decision decoding methods.

1.4.1 Structure of Convolutional codes

The two most important types of convolutional codes are the *non-systematic* (NSC) and the *recursive systematic* convolutional (RSC) codes. In figure 1.2 a simple example of a non-systematic convolutional code is shown. A convolutional code takes k input bits and generates n output bits; the code rate is defined as $R = k/n$. The output bits are linear combinations of the present input bit and delayed input bits. The encoder has rate $R = 1/2$ because it uses one input bit and generates two output bits. The constraint length of the encoder is the number of bits that the encoder

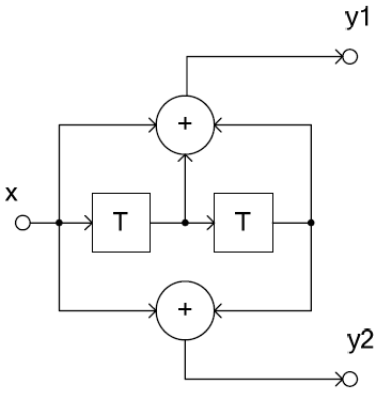

Figure 1.2: Non systematic convolutional code (NSC).

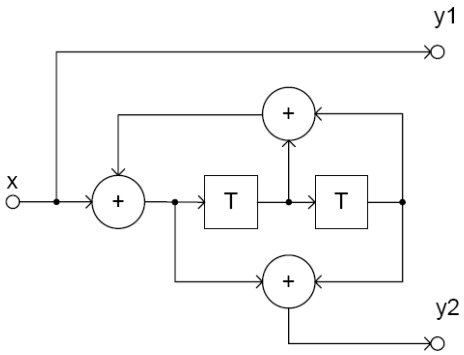

Figure 1.3: Recursive systematic convolutional code (RSC).

depends on. The encoder in the example below have constraint length $k = 3$. The second type of convolutional codes is the recursive systematic convolutional code in figure 1.3. One of the output bits is systematic ($y1$), the parity bits ($y2$), depend on all previous input bits due to the feedback loop in the encoder.

The inner state of the encoder is defined as the values of the memory cells. The simple encoders in the examples above can take four different states; 00_2, 01_2, 10_2 and 11_2. The transitions between the different states and the output from the encoder can either be described by a state diagram or by a trellis. Figure 1.4 represent state diagram to describe the behavior of the NSC code in the example above. It can be seen in the figure that if current state is 00_2 and the input is 0_2, the next state will be 00_2. Furthermore, from the state diagram the output can also be found; e.g. in this particular case, the output is *y1y2*= 00_2.

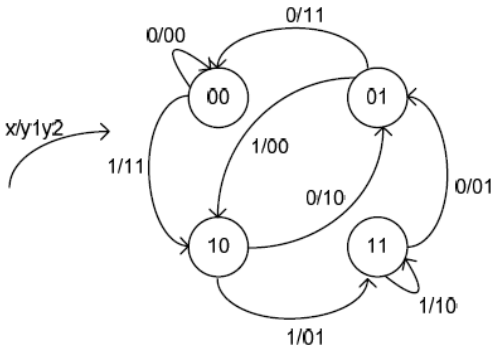

Figure 1.4: State diagram of the encoder.

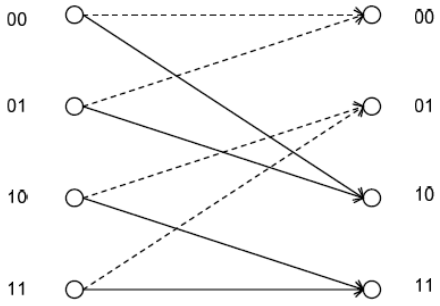

Figure 1.5: Trellis diagram of the encoder.

1.4.2 Trellis Diagram

A *trellis diagram* is derived from the state diagrams by tracing all possible input/output sequences and state transitions. Figure 1.5 shows the trellis diagram for the encoder described by the state diagram in Figure 1.4. There are two branches emanating from each state corresponding to two different symbols. There are also two branches merging into each state. The transitions caused by an input symbol "0" are indicated by dotted lines while the transitions caused by an input symbol "1" are indicated by continous lines. It should be noted that NSC encoder described above is one of the base component in turbo code encoders which will be detailed in the next sections.

1.5 Turbo Codes and Decoding Algorithm

1.5.1 Turbo Encoder

A Turbo encoder is formed by concatenation of two RSC encoders separated by a random interleaver. The concatenation could be either serial or parallel, however in this section we will talk about parallel concatenation as in this thesis we have used extensively such kind of turbo codes. For more detail on these two schemes interested readers are refered to [Boutillon et al., June 2007]. In parallel concatenation two encoders operate on the same set of input bits, rather than one encoding the output of the other, this encoding process is explained for turbo codes used in WiMax standard as we will be dealing with them in our work. The encoder uses a double binary Circular Recursive Systematic Convolutional (CRSC) code. The basic difference between binary and duo-binary turbo codes is that in former each RSC component code has one input while in later it has two inputs. The advantage of duo-binary turbo codes are better convergence of the iterative decoding, large minimum distances (i.e. large asymptotic gains), less sensitivity to puncturing patterns, reduced latency, and robustness towards the flaws of the component decoding algorithm [Berrou and Jezequel, Jan. 1999]. The encoder structure illustrated in the Figure 1.6 is Parallel Convolutional Concatenated Codes (PCCC).

First, the encoder 1 (ENC1)(after initialization by the circulation state[1] S_{c1}) is fed by the sequence in the natural order with the incremental address $i = 0 \dots N-1$. This first encoding is called $C1$ encoding. Similarly encoder 2 (ENC2) (after initialization by the circulation state S_{c2}) is fed by the interleaved sequence with incremental address $j = 0 \dots N-1$. This second encoding is called $C2$. In order to achieve various coding rate, puncturing is introduced to the mother code. Puncturing is the process of removing some of the parity bits after encoding. This has the same effect as encoding with an error-correction code with a higher rate, or less redundancy. However,

[1] Circular state is an alternative to tail bit method [Berrou et al., 2005]. For a given information block, there is one and only one state S_c (circulation state) such as $S_c = S_0 = S_k$. Although the tail bit method is easy to implement, the transmission of additional bits is needed and initial and final states are singular states. Instead in circular state approach coding rate remains unchanged and the trellis can be regarded as a circle without any singularity. However a pre-coding step is required for the circular state.

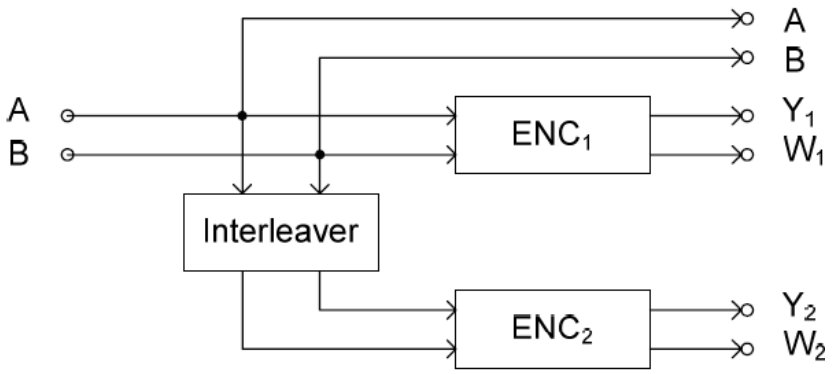

Figure 1.6: WiMaX encoder structure.

with puncturing the same decoder can be used regardless of how many bits have been punctured, thus puncturing considerably increases the flexibility of the system without significantly increasing its complexity. The order in which the encoded bit shall be fed into the subpacket generation block is:

$$A, B, Y_1, Y_2, W_1, W_2$$

$$=$$

$$A_0, B_0, .., A_{N-1}, B_{N-1}, Y_{1,0}, Y_{1,1}, .., Y_{1,N-1}, Y_{2,0}, Y_{2,1},$$

$$...$$

$$Y_{2,N-1}, W_{1,0}, W_{1,1}, .., W_{1,N-1}, W_{2,0}, W_{2,1}, .., W_{2,N-1}$$

The encoding block size shall depend on the number of subchannels allocated and the modulation specified for the current transmission. Concatenation of a number of subchannels shall be performed in order to make larger blocks of coding where it is possible, with the limitation of not passing the largest block under the same coding rate (the block defined by 64-QAM modulation). For a complete description see [802.16e, 2004]. This code guarantees better convergence, better performance, especially at low SNR and high data rate. The polynomial characteristics for the encoder are (figure 1.7):

$$G_0(D) = 1 + D_1 + D_3 \tag{1.2}$$

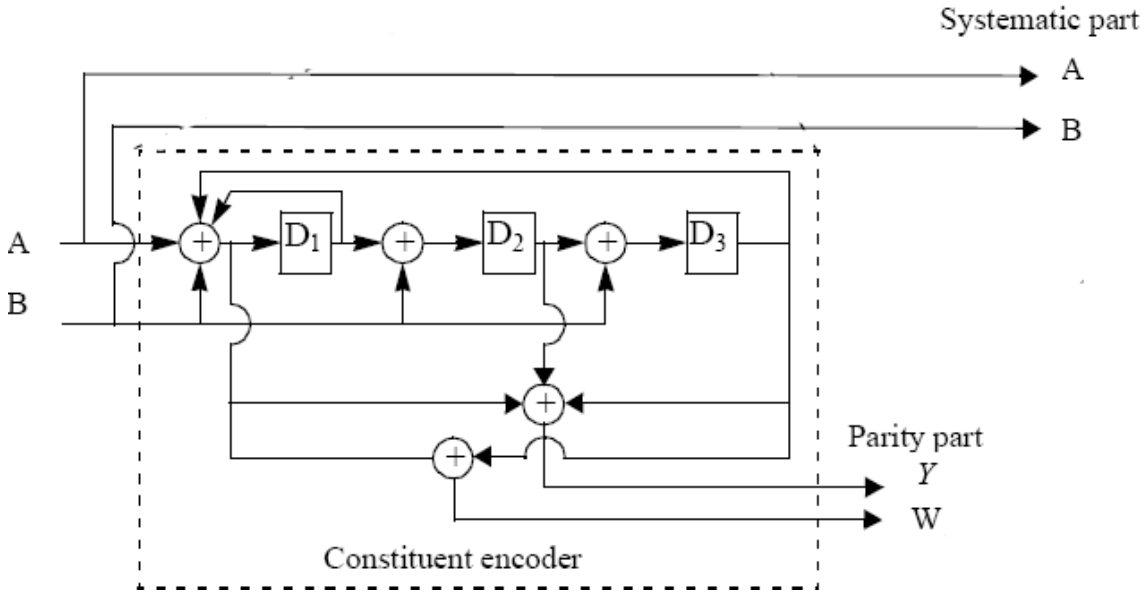

Figure 1.7: WiMAX constituent encoder internal structure.

$$G_1(D) = 1 + D_2 + D_3 \tag{1.3}$$

$$G_2(D) = 1 + D_3 \tag{1.4}$$

The equation 1.2 is for the feedback branch, the 1.3 is for the Y parity bit and the 1.4 is for W parity bit. These three equations are useful for the calculation of the trellis of every encoder. As explained in the figure 1.8 each trellis contains eight initial states $(000, 001, 010, 011 \ldots$ etc) and eight final states. Four arrows leave each state (corresponding input $00, 01, 10, 11$) and four arrows arrive in each final state. Inputs and outputs of the transition are represented along these arrows.

1.5.2 Interleaving

The channel interleavers that normally are included in communication system are used to spread out burst errors of several FEC coding blocks in order to enable correction. The reason for interleaving in turbo coding is to rearrange the input stream in a manner so that the correlation between the rearranged and original data is minimized; i.e. the correlation of the parity bits from $C1$ and $C2$ is minimized. There are several methods for interleaving and the way as it is performed influences the noise performance of the turbo code. For the WiMAX standard the permutation law (*interleaving law*) is given as follows:

```
for j=0 ... K-1
switch j mod 4:
case 0: P(j) = (P0j+1) mod K
```

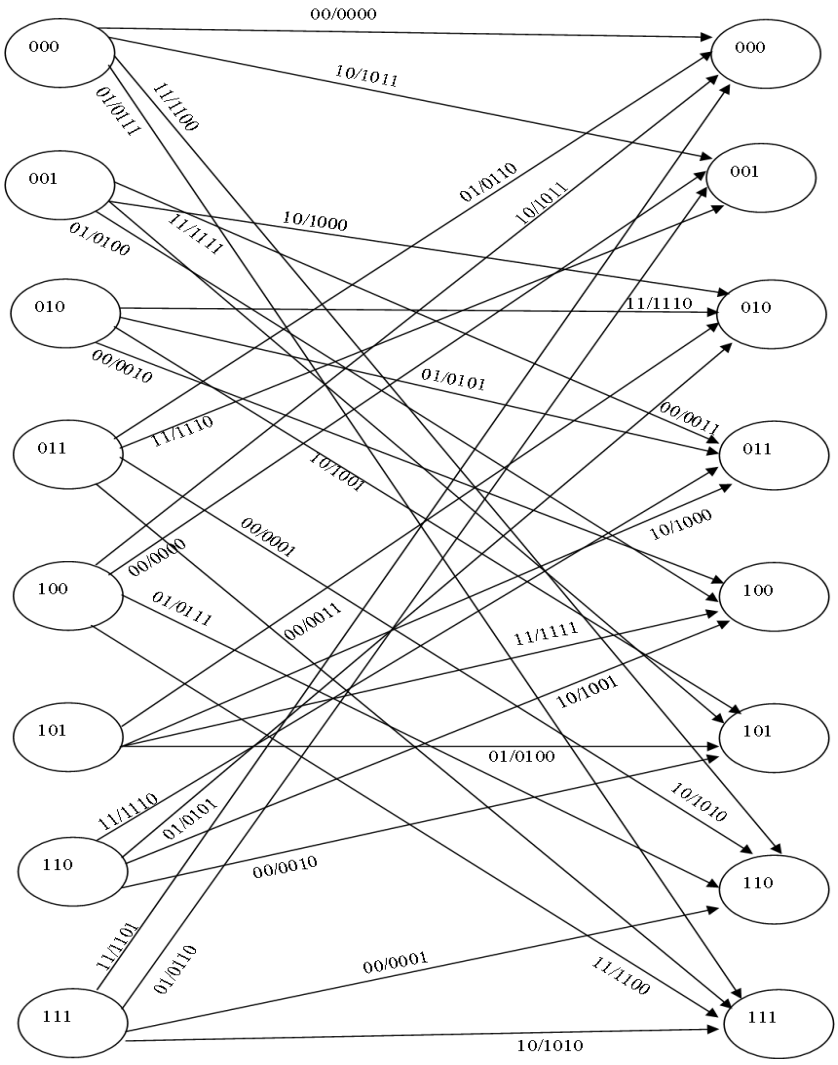

Figure 1.8: WiMaX CTC encoder trellis.

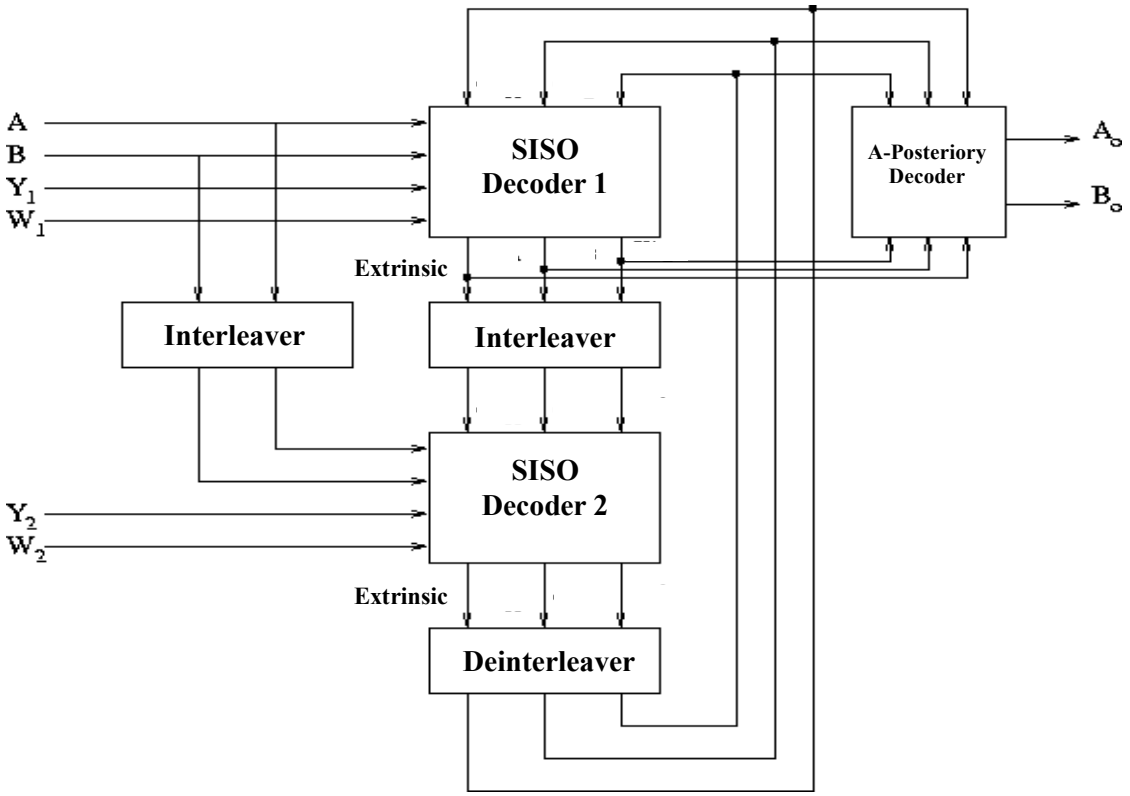

Figure 1.9: WiMaX CTC logical decoding process.

```
case 1: P(j) = (P0j+1+K/2+P1)mod K
case 2: P(j) = (P0j+1+P2) mod K
case 3: P(j) = (P0j+1+K/2+P3) mod K
end
```

where P_0, P_1, P_2 and P_3 are constants related to block length of the code K (see [802.16e, 2004]). This kind of algorithm is conceived to facilitate parallel implementation of interleavers.

1.5.3 Decoding process

In this section the decoding process is explained with reference to WiMaX convolutional turbo codes. The soft input from the demodulator is initially de-punctured by padding zeros in the positions of the punctured bits. The zeros added in the puncture/de-puncture are spread out in the parity sub-blocks. The systematic bits are never punctured. The input data are then processed by the SISO unit, which implements the BCJR algorithm [Bahl et al., Mar. 1974] (first half iteration), the results called *extrinsic information* are interleaved according to the permutation law used in the encoder and then processed by another SISO (second half iteration) as shown in Figure 1.9. The input to the decoder is kept constant, the decoding is performed several times, and only extrinsic informations are passed between the SISO. After the predetermined number of iterations, (typically

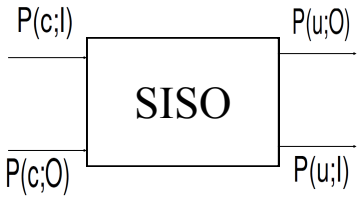

Figure 1.10: Generic SISO Decoder

$4-8$ depending on BER and FER requirement), a final decision is made by using the extrinsic information from the two SISO decoders and the systematic soft bits from the demodulator. There are also algorithms that use an early stopping criterion, which means the decoding is stopped when certain rules are fulfilled, e.g. the same estimation of the output for two or three consecutive iterations. Early stopping increases the throughput of the decoder however they will not be detailed here as they are out of the scope of this thesis, interested reader can refer to [Matache et al., Aug. 2000], [Shao et al., 1999].

1.5.3.1 BCJR algorithm

The BCJR algorithm, used in SISO module, basically receives as input blocks of soft information associated to each processed bit (LLR) and it is capable to refine them, generating as output new reliable soft bits. To derive a basic understanding of how this algorithm works, a general structure of SISO is presented Fig.1.10 where u are the input symbols of the encoder while c are the coded outputs. Moreover P(u; I) and P(c; I) are respectively, the estimations of the probability distributions of the encoder input and output symbols; while P(u;O) and P(c;O) are the refined values for these distributions after BCJR algorithm computations. These probability distributions are related to the aforementioned extrinsic information reported in Figure 1.9. The basic equations for calculating refined probability distributions in the original form are:

$$P_k(c;O) = H_c \sum_{e:c(e)=c} A_{k-1}[s^S(e)]P_k[u(e);I]B_k[s^E(e)] \tag{1.5}$$

$$P_k(u;O) = H_u \sum_{e:u(e)=u} A_{k-1}[s^S(e)]P_k[c(e);I]B_k[s^E(e)] \tag{1.6}$$

where:

- Index k indicates the time step and runs on the whole transmission length, while symbol s represents a code state;
- e is a generic trellis edge, while $c(e)$ and $u(e)$ are the output and input coder symbols associated to the edge e;
- $s^S(e)$ and $s^E(e)$ indicate the starting and the ending states for the generic trellis edge e;
- H_u and H_c are normalization constants.

A_{k-1} and B_{k-1} are probability distribution accumulated (*path metrics*) in the forward and backward direction along the trellis, according to the following updating relations:

$$A_k(s) = \sum_{e:s^E(e)=s} A_{k-1}[s^S(e)]P_k[u(e);I]P_k[c(e);I] \tag{1.7}$$

$$B_k(s) = \sum_{e:s^S(e)=s} A_{k+1}[s^E(e)]P_{k+1}[u(e);I]P_{k+1}[c(e);I] \tag{1.8}$$

It is clear from the direct inspection of these equations that their straightforward implementation in a digital architecture would give rise to at least two serious problems:

1. The updating of the involved metrics implies that the whole sequence of transmitted data (block) has been received and stored. This has the effect to increase the decoder memory.
2. A large number of multiply operations are involved, which has huge impact on overall decoder complexity.

The first problem is solved using an approximated version of the original algorithm, largely known as *sliding window* BCJR algorithm [Blankenship et al., June 2005b]. In the *sliding window* approximation, the block of data to be decoded is divided into a number of segments, or windows (length N_W) and the decoding algorithm is applied to each of them in sequence. It has been proved that with a good choice N_W the performance loss due to this approximation is negligible. For the second problem the usual solution is provided by re-formulating the algorithm in the logarithmic domain. In fact, by processing logarithmic likelihood ratios (LLR's) rather than the original reliability metrics allows to replace multiplications with additions. Considering the logarithm of $P_k(u;O)$ (and the same for $P_k(c;O)$) we can write $\log(P_k(u;O))$ as:

$$\pi_k(u(e);O) = const + \underbrace{log(\sum_{e:u(e)=u} \exp^{\alpha_{k-1}[s^S(e)]} \exp^{\pi_k(c(e);I)} \exp^{\beta_k[s^E(e)]})}_{a} \tag{1.9}$$

where:

- $\alpha_{k-1}[s^S(e)] = log(A_{k-1}[s^S(e)])$
- $\beta_k[s^E(e)] = log(B_k[s^E(e)])$
- $\pi_k(c(e);I) = log(P_k[c(e);I])$ $(\pi_k(u(e);I) = log(P_k[u(e);I]))$
- $\pi_k(u(e);O) = log(P_k(u(e);O))$ $(\pi_k(c(e);O) = log(P_k(c(e);O)))$

Therefore the new BCJR equation obtained with this new transformation are based on a generic function a. This operation is defined as:

$$a = log\left[\sum_{i=1}^{n} exp(a_i)\right] \tag{1.10}$$

and for VLSI implementation requires a proper approximation [Masera, 2005]. The two most widely known techniques to approximate equations 1.10 are max* and max. Recalling the Jacobian operator, equation 1.10 can be evaluated recursively applying the new operator max* as:

$$log(\exp(a_1) + \exp(a_2)) = max^* = max(a_1, a_2) - log(1 + exp(-|a_1 - a_2|)) \tag{1.11}$$

The max* operator requires in general two sums and a look-up table. The look-up table size depends on the required accuracy. However in the max approximation whole correction term can be avoided, giving rise to a simpler and suboptimal version of the SISO (max-sum or max-log-MAP):

$$log(\exp(a_1) + \exp(a_2)) \approx max(a_1, a_2) \tag{1.12}$$

To compensate the effect of neglecting the look-up table several strategies are possible, like scaling or offsetting the messages. These techniques are described in [Masera et al., Sept. 1999], [Wu et al., April 2005]. Coming back to the equations we can consider that $LLR's = \log\frac{P_i}{P_j}$ and therefore we can define :

$$\lambda_k[u;O] = \pi_k[u(e) = u;O] - \pi_k[u(e) = \tilde{u};O] = \max_{e:u(e)=u}{}^{*}\{b(e)\} - \max_{e:u(e)=\tilde{u}}{}^{*}\{b(e)\} - \lambda_k[u;I] \tag{1.13}$$

where $\tilde{u}$ is an input symbol taken as a reference (usually $\tilde{u} = 00$). With duo-binary three extrinsic LLR's are produced, then in general the terms $\lambda_k[u;O]$ and $\lambda_k[u;I]$ are vectors. The term $b(e)$ in

1.13 is defined as:

$$b(e) = \alpha_{k-1}[s^S(e)] + \gamma_k[e] + \beta_k[s^E(e)] \tag{1.14}$$

$$\gamma_k[e] = \pi_k[u(e); I] + \pi_k[c(e); I] \tag{1.15}$$

The $\pi_k[c(e); I]$ is computed by adding together the $\lambda_k[c; I]$ produced by the soft demodulator as:

$$\pi_k[c(e); I] = \sum_{i}^{n_c} c_i(e)\lambda_k[c_i(e); I] \tag{1.16}$$

where $c_i(e)$ is one of the coded symbols associated to e and n_c is the number of bits forming a coded symbol. On the other hand, we can write $\pi_k[u(e), I]$ as:

$$\pi_k[u(e), I] = \begin{cases} 0 & if\ u(e) = (\prime 0\prime, \prime 0\prime), \\ \lambda_k^{\overline{A}B}[u(e); I] & if\ u(e) = (\prime 1\prime, \prime 0\prime), \\ \lambda_k^{A\overline{B}}[u(e); I] & if\ u(e) = (\prime 1\prime, \prime 0\prime), \\ \lambda_k^{AB}[u(e); I] & if\ u(e) = (\prime 1\prime, \prime 1\prime) \end{cases}$$

For path metrics, the same method was adopted and the result obtained are:

$$\alpha_k[s] = \max_{e:s^E(e)=s}{}^{*} \left\{\alpha_{k-1}[s^S(e)] + \gamma_k[e]\right\} \tag{1.17}$$

$$\beta_k[s] = \max_{e:s^S(e)=s}{}^{*} \left\{\beta_{k+1}[s^E(e)] + \gamma_{k+1}[e]\right\} \tag{1.18}$$

where $\alpha_k[s^S(e)]$ and $\beta_k[s^E(e)]$ are the forward and backward metrics associated to $s^S(e)$ and $s^E(e)$.

1.6 LDPC Coding and Decoding

Low Density Parity Check (LDPC) Codes, are a class of linear block codes, with parity-check matrices H very sparse, i.e. the number of 1's are small percentage of the zero elements [Gallager, Jan. 1962]. Moreover, "code dimensions" in terms of information word and codeword sizes may be in the order of hundreds or thousands digits (from 600 up to 64000 bits). R. Gallager defined an

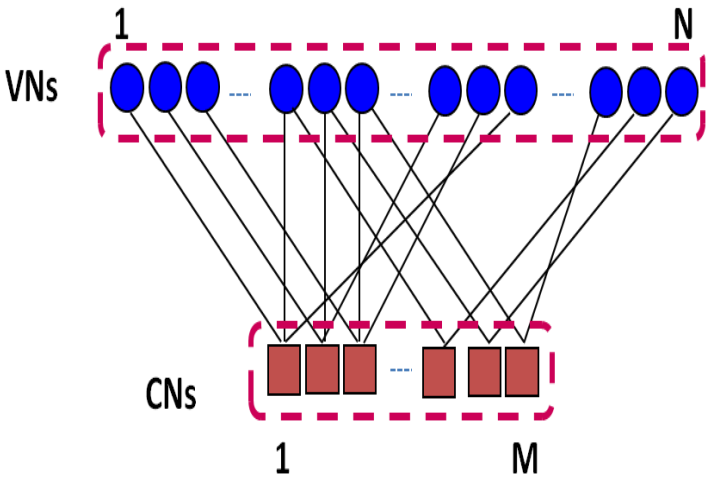

Figure 1.11: Tanner Graph for LDPC code.

(N, m, n) LDPC code as a block code of length N having a small fixed number (m) of ones in each column of the parity check matrix H, and a small fixed number (n) of ones in each rows of H. In the Gallager's original LDPC code design, there is a fixed number of ones in both the rows (n) and the columns (m) of the parity check matrix: it means that each bit is implied in m parity check constraints and that each parity check constraint is the exclusive-OR (XOR) of n bits. This class of codes is referred to as *regular* LDPC codes. On the contrary, *irregular* LDPC codes do not have a constant number of non-zero entries in the rows or in the columns of H (e.g. WiMAX LDPC codes).

1.6.1 Tanner Graph representation

An LDPC decoder architecture is strictly related to the Parity-Check Matrix H; in fact, due to the sparseness of this matrix, the decoder can be represented in term of a bipartite graph called Tanner Graph [Tanner, May 1981.]. A bipartite graph is a graph where the elements of a first class can be connected to the elements of a second class, but not to the same class. In a Tanner graph for binary LDPC codes (Fig. 1.11) there are two classes of Processing Elements (PEs) that are related to the rows and columns of the H matrix.

The N nodes related to the rows, i.e. the length of the codeword also known as "code-block", are usually reported, in LDPC notation, as Variable (or Bit) Nodes (*VNs*); on the other hand, there are $M = N\text{-}K$ nodes (K is the information word size), called Check Nodes (CNs) that are related to the columns of the H matrix, i.e. the M parity check equations of the code. Moreover, an edge e_{ji} on the Tanner Graph connecting a Variable Node VN_j with a Check Node CN_i (i.e. messages exchanged between this two nodes) corresponds to a "1" in the H matrix between row j and column i. The Tanner graph representation of error correcting codes is very useful since their decoding

algorithms can be explained by the exchange of information along the edges of these graphs.

1.6.2 Optimal decoding Algorithm

From the Tanner graph representation of LDPC matrix in the previous section the decoding of LDPC codes can be easily explained. The VNs receive the intrinsic information λ from the channel and update them depending on the results of the parity check equations compute at the CNs; this process is iterate several times until a converge criterion is met. This criterion may be that a codeword was decoded or that a maximum number of iterations were reached. Since a great number of messages are exchanged among Variable and Check Nodes, this algorithm is also called "Two Phase Message Passing" (TPMP), but since these messages are reliability information, it is better known as Belief Propagation Algorithm (BPA). Analyzing in more details this algorithm, it can be seen clearly that VNs receive likelihood functions (i.e probabilities) from the channel; moreover also the messages from the CNs are reliability values that are also probabilities. This means that the updated versions computed at the Variable Nodes are obtained by means of probability multiplications since this will give information on how reliable were the received data. On the other hand CNs have to compute parity checks; this means that CNs must sum up all the incoming messages from the VNs. Only if the sum is zero then the data stored in the VNs are correct and a codeword was found. This is basically the "Sum-Product" implementation of the Belief Propagation Algorithm, but recalling that we avoid multiplications we have to work in logarithmic domain so multiplications becomes additions and sums become more complex functions. Considering the k^{th} iteration and a generic Variable Node VN_j that send a message Q_{ji} to a generic Check Node CN_i on edge e_{ji} , the updated outgoing message (Q_{ji}) is obtained from the channel intrinsic information LLR λ_j and from the messages ($\mathrm{R}_{\alpha j}$) from all the CNs connected to VN_j excluding the one (R_{ij}) received on the same edge e_{ji} (i.e. extrinsic information computation). This behavior is described by equation

$$Q_{ji} = \lambda_j + \sum_{\alpha \in C[j]/\{i\}} R_{\alpha j}\left[k-1\right] \tag{1.19}$$

where $\mathrm{C}_{[j]}$ is whole set of incoming edges for VN_j and k is the iteration.

A generic Check Node CN_i, on the other hand, receives messages $\mathrm{Q}_{\gamma i}$ from all the VNs that are connected with it and computes a parity check equation. Then it sends back to the VNs some reliability measures depending on the results of these parity checks. In particular it sends on the edge e_{ji} a message (R_{ij}) to the Variable Node VN_j that depends on all the messages $\mathrm{Q}_{\gamma i}$ it has received excluding the one (Q_{ji}) that was received on the same edge e_{ji} (i.e. other extrinsic information computations). This message updating is described by eq.

$$R_{ij}\left[k\right] = \psi^{-1}\left[\sum_{\gamma \in R[i]/\{j\}} \psi\left(Q_{\gamma i}\left[k\right]\right)\right] \bullet \delta_{ij} \tag{1.20}$$

where $R_{[i]}$ is the set of all messages received by the CN_i from the all VNs connected to it.

The $\psi(x)$ operator in equation 1.20 is an non-linear function which actually performs the CN processing according to the Belief Propagation algorithm. Its general expression is shown by equation

$$\psi(x) = -\ln\left(tanh\left|\frac{x}{2}\right|\right) \tag{1.21}$$

Moreover, the term δ_{ij} in equation 1.20 is obtained from the sign of all the incoming messages Q_i (excluding Q_{ji}) as in :

$$\delta_{ij} = (-1)^E \left[\prod_{\gamma \in R[i]/\{j\}} sgn\left(Q_{\gamma i}[k]\right)\right] \tag{1.22}$$

where E is the number of incoming edges at Check Node CN_i (including, in this case, also Q_{ji}).

When the convergence criterion is met, then Variable Nodes can evaluate the corrected output accordingly to equation:

$$\Lambda_j = \lambda_j + \sum_i \in C_{[j]} R_{ij}[k] \tag{1.23}$$

where all the messages from CNs are taken into account; in fact to compute the decoded value all the information from the CNs are needed since they are related to the result of the parity check equations. The "hard decoded" output bit $\widehat{x}_j$ (an estimation of the actually coded bit x_j) is obtained from the sign of Λ_j as in eq:

$$\widehat{x}_j = sgn\left(\Lambda_j\right) \tag{1.24}$$

while $|\Lambda_j|$ (magnitude of Λ_j) is its reliability [Hagenauer et al., Mar. 1996].

From above mentioned elaborations, the Belief Propagation Algorithm can be easily sketched: VNs update the intrinsic information from the channel with the messages from the CNs and send the new extrinsic information back to CNs. Check Nodes compute new parity check equations and send to VNs new extrinsic informations that are the results of these parity checks. All this process is applied iteratively until a convergence criterion is met and then the estimated codeword $\widehat{x}_j$ is obtained taking the sign of the updated likelihood functions Λ computed at the VNs. Actually, since data are represented in 2-complement notation, a positive value has a sign represented with a binary digit '0' while negative ones as sign-digit equal to '1'. Since positive value are related to a transmitted value equal to '1' and negative likelihood values represent code digit equal to '0', then the sign of Λ_j have to be inverted via logic NOT to obtain the actual decoded bit $\widehat{x}_j$. The algorithm that we have briefly described was proved to be a Maximum Likelihood algorithm under the assumption that the Tanner Graph is "cycle-free" (or acyclic).

A graph is said to be acyclic if and only if there are no closed paths starting from a node and ending on the same one (cycle). In actual codes, there are, however, some cycles of certain length 4:

nevertheless it was also proved that if the length ("girth") of these cycles is greater than 4 (i.e. more than 4 edges in a cycle), then BPA will still behaves closely to a Maximum Likelihood algorithm, [MacKay, Mar. 1999]. Moreover, regularities in the code construction will lead to poorer decoding performances since some limitations in the sparseness of the LDPC code are posed, [MacKay, Mar. 1999], [Richardson et al., Feb. 2001]. For these reasons, irregular codes have better error-correcting properties rather than regular ones, even if this means that actual hardware implementations of both encoder and decoder will have higher complexity due to this irregularity.

1.6.3 Sub-optimal decoding algorithm

Analyzing in more details equations 1.20 and 1.21 mentioned in previous section it can be seen that the $\psi(x)$ function is highly non-linear and a direct mapping into an hardware resource will require the adoption of Look-Up Tables (LUTs). But since significantly large number of bits are necessary in order to cope with finite precision issues, this will lead to the adoption of big LUTs with lots of entries, a significant cost in term of hardware complexity. To deal with finite precision issue reduction and at the same time the area requirements, different solutions were proposed in literature trying to avoid evaluations of the function. The simplest suboptimal check node algorithm is the well known Min-Sum algorithm [Fossorier and Imai, May 1999]. Only the magnitude part of the check node update rule differs from the iterative algorithm explained in previous section and is give as following:

$$R_{ij}[k] = \min_{\gamma \in R[i]/\{j\}} |Q_{\gamma i}[k]| \bullet \delta_{ij} \tag{1.25}$$

This is an important simplification in the BP-based algorithm since the check node update is replaced by a selection of the minimum input value. But this simplification is made at the expense of a substantial loss in performance. There are other similar approaches in this class, like λ-min [Guilloud et al., Sep. 2003] or average min-sum [Axvig et al., Sep. 2008].

In [Mansour and Shanbhag, Aug. 2002], a different approach is suggested: instead of applying the classical BPA updating equations a Turbo-like solution is adopted. In particular Check Node elaborations are computed by means of a modifications of the classical BCJR, [Bahl et al., Mar. 1974] algorithm. Mansour et al. proved that the adoption of a BCJR-like decoding scheme is able to achieve remarkable error-correcting properties even when finite precision is taken into account. A parity-check matrix $H_{m \times n}$ of an LDPC code containing r_i ones per row, $i = 1, \ldots, m$, can be represented using m trellises corresponding to $(r_i, r_i - 1)$-SPC codes. Using this representation, it is possible to describe an LDPC code using an alternative graph based on the bipartite graph of the code. In this graph, a check node of degree r_i is replaced by the trellis of an $(r_i, r_i - 1)$-SPC code, while the bit nodes are removed. The edges incident on a check node in the bipartite graph are connected to the appropriate sections of the trellis corresponding to that check node, and the edges incident on a bit node are connected directly to each other. If two rows in H overlap in q

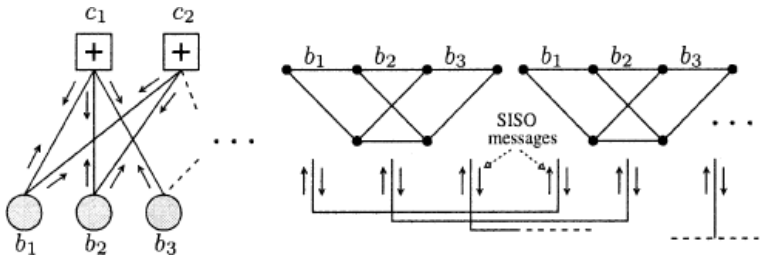

Figure 1.12: Trellis representation of a portion of bipartite graph.

column positions, then their trellises are connected by edges at those overlapping positions. Figure 1.12 shows a portion of the bipartite graph represented using trellises connected through edges.

The bit-to-check and check-to-bit messages associated with the nodes of a bipartite graph are replaced by a single type of message typical of turbo decoding. Each $(r_i, r_i - 1)$-SPC code associated to one row, processed in the SISO using the simplified form of BCJR algorithm. This approach is also known as forward backward method. The key equations of the algorithm for any section of such trellis reduce to:

$$\alpha_1^{'} = ln(e^{\alpha_1} + e^{\alpha_1 + \lambda}) \tag{1.26}$$

$$\alpha_2^{'} = ln(e^{\alpha_1 + \lambda} + e^{\alpha_2}) \tag{1.27}$$

$$\beta_1^{'} = ln(e^{\beta_1} + e^{\beta_1 + \lambda}) \tag{1.28}$$

$$\beta_2^{'} = ln(e^{\beta_1 + \lambda} + e^{\beta_2}) \tag{1.29}$$

$$\Lambda = ln(e^{\alpha_1 + \beta_2} + e^{\alpha_2 + \beta_1}) - ln(e^{\alpha_1 + \beta_1} + e^{\alpha_2 + \beta_2}) \tag{1.30}$$

where λ is the input prior and intrinsic channel reliability value of the code bit associated with that trellis section, Λ is the updated output reliability of that code bit, and α, β are intermediate forward and backward state metrics, respectively. Note that these equations are pretty similar to equations used in BCJR algorithm for decoding of Turbo codes. The difference is that in Turbo case

we can have trellis with 4, 8 or 16 states, here we have trellis with two states and also the length of the trellis here is the number of bits per check node whereas for turbo it is the window size (in the case of sliding window BCJR N_W). Equation 1.26, 1.27, 1.28, 1.29 can be approximated using max^*.

1.6.4 Structured Codes and Scheduling

From the previous section it is evident that an LDPC code is completely specified by its H matrix. Thus a proper construction for this matrix is advisable for achieving significant coding gains. These matrices can be "constructed" according to some constraints. In particular, some codes are designed in order to greatly simplify the decoder architecture. Most of these codes, in fact, are based on blocks of sub-matrices obtained by permutations of the rows of Identity Matrices leading to the so-called structured codes [Zhang and Parhi, Nov. 2001,]. These particular structures allows the implementation of simplified decoders since dispatching of messages from VNs to CNs could follow some simple rules (like indices permutation). Adding some regularity, like structured matrices, may degrade the code's decoding performance; thus particular care needs to be taken in selecting proper constructing rules that allow remarkable error correcting capabilities.

In this section as an example of such codes, we discuss about LDPC codes used for WiMaX application which we will be using extensively in our thesis. The entire H matrix is composed of the same style of blocks with different cyclic shifts, which allow structured decoding and reduces decoder implementation complexity. The H_{BASE} matrix defined as:

$$H_{BASE} = \begin{bmatrix} \Pi_{0,0} & \Pi_{0,1} & \dots & \Pi_{0,N} \\ \Pi_{1,0} & \Pi_{1,1} & \dots & \Pi_{1,N} \\ \vdots & \vdots & \vdots & \vdots \\ \Pi_{i,0} & \Pi_{i,1} & \Pi_{i,j} & \Pi_{i,N} \\ \vdots & \vdots & \vdots & \vdots \\ \Pi_{M,0} & \Pi_{M,1} & \dots & \Pi_{M,N} \end{bmatrix}$$

is associated to a parity check matrix H. It has M block rows and N block columns. The H_{BASE} is expanded, in order to generate H matrix, by replacing each of its entries $\Pi_{i,j}$ with a Z-by-Z permutation matrix, where Z is the expansion factor. The permutation matrix can be formed by cyclically shifting right the Z-by-Z identity matrix. The set of shifts in the base matrix are used to determine the shift sizes for all other code lengths of the same code rate.

Standard defines various code rates (eg. R=1/2, R=2/3, R=3/4). The base matrix is defined for the largest code length ($N = 2304$) of each code rate. Table 1.1 collects some details about the different WiMaX LDPC codes.

Table 1.1: H_{BASE} parameters for different WiMaX LDPC codes.

Code Rate	**1/2**	**2/3**	**3/4**
H_{BASE} matrix	12×24	8×24	6×24
Type	irregular	irregular	irregular
Number of block rows [M]	12	8	6
Maximun row-column weight	$7-6$	$11-6$	$15-4$

The scheduling of the BP algorithm for LDPC codes is the order in which the messages of the graph should be propagated. For practical codes, whose graphs are not cycle-free, the flooding schedule is used. The most commonly used schedling in the literature are as follows:

1.6.4.1 Flooding Schedule

The flooding schedule is the classical way of scheduling the BP algorithm. In this schedule, mode of operation consists of two stages. At first all the variable nodes are processed to generate the updated messages destined for check nodes and when all the messages are sent then the check nodes are updated all together performing the parity check function [Kschischang and Frey, 1998]. The update for a type of node can be made either one node at a time (serially) or in parallel without any change in the output messages.

1.6.4.2 Shuffled Schedule

In [Zhang and Fossorier, Nov. 2002] authors proposed a shuffle BP algorithm which converges faster than the BP algorithm. The idea is to update the information as soon as it has been computed, so that the next node processor to be updated could use a more up to date information. This schedule operates along the variables: it means that all the variable node are processed one after the other, hence it is also called vertical shuffle since the check node are processed in a shuffle order.

On the other hand, the authors in [Mansour and Shanbhag, Aug. 2002] introduced the concept of Turbo Decoding Message Passing (TDMP, also referred as layered decoding in [Hocevar, Oct. 2004] or horizontal shuffle) where block of rows (check nodes) are seen as supercodes. This approach is dealt with, in detail here, as we will be using it primarly in our work. Each constituent supercode of the H matrix is an even parity-check code having support only in positions where there are non-negative numbers in the corresponding H_{BASE} matrix. The layered decoding algorithm decodes a codeword iteratively in a number of sub-iterations which are equal to the number of supercodes, with one sub-iteration per constituent supercode, using a constituent SISO decoder and an interleaver similar to decoding a serially concatenated turbo code. The logical dataflow of the decoding process is shown in Figure 1.13.

The SISO decoder accepts in a sub-iteration, intrinsic input reliability messages (denoted by λ)

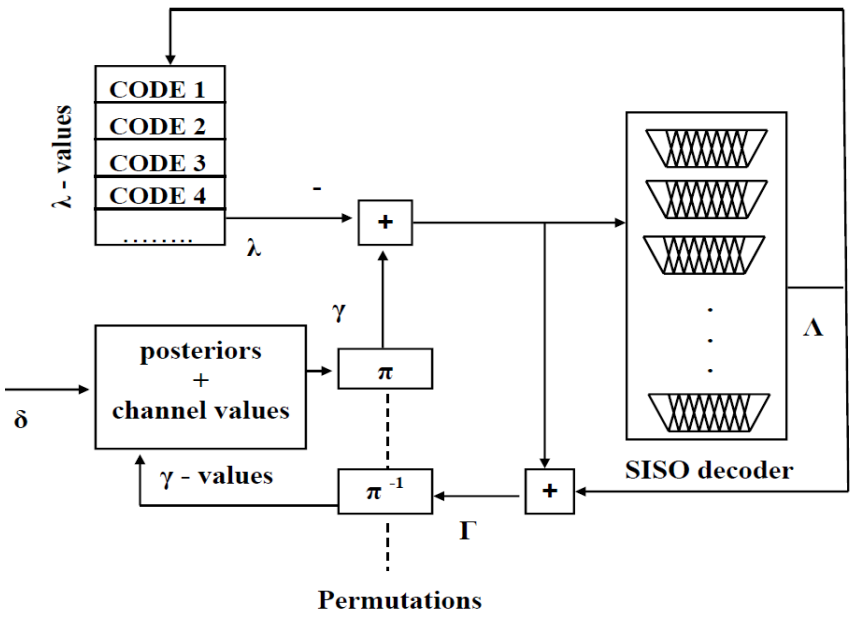

Figure 1.13: LDPC logical decoding flow using TDMP algorithm.

generated from all other previous sub-iterations. Intrinsic λ-messages pertaining to the supercode under consideration are excluded to minimize correlation between messages from different sub-iterations. The sum of all intrinsic λ-messages in addition to the channel values δ is denoted by γ :

$$\gamma = \delta + \sum_{super-codes} \lambda \tag{5}$$

The SISO generates as output updated extrinsic messages (denoted by Λ) corresponding to the constituent supercode being decoded, as well as updated posterior messages (denoted by Γ). At the end of a sub-iteration, these Λ and Γ-values are stored as λ and γ-values to be used as inputs in the next sub-iteration. The next CN subset (supercode) will thus receive newly updated messages which improves the convergence speed and therefore increases communications performance for a given number of iterations.

1.7 Conclusion

The purpose of this chapter was to introduce the application domain in which this thesis is focussed i.e. digital communication. We presented the different elements playing a role in the digital transmission over a channel and in particular element of channel decoding was detailed. Some of the widely used channel code families were explored, including their coding and decoding algorithms

in the context of real codes employed in WiMAX application. Challange of creating architecture platforms, supporting all together these different codes and their decoding, motivates us to explore the flexible architecture paradigm which will be detailed in the next chapter with extensive study of the related state of art.

Chapter 2

Context

Contents

This chapter presents the motivation of the thesis work along with the state of art in the area of different subset of flexible channel decoder design space. A brief overview of the flexibility paradigm is presented and then a problem space is defined for the flexible channel decoder architecture implemetation. Each subset of this problem space is confronted with already existing solutions and then at the end our contribution to them are highlighted.

2.1 Introduction

Since the number of radio standards is growing increasingly fast and the diversity among the standards is increasing, there is a need for a processing solution capable of handling as many standards as possible and at the same time not consuming more chip area and power than a single-standard product. In the last few years, the technology market was dominated by the quest of even better communication infrastructures. Typical example is the mobile phone market where non-voice traffic is becoming dominant over traditional voice communication. This led to the introduction of

novel standards, like UMTS (3G) which is able to support higher data rates with higher reliability. Beside mobile phone, novel data wireless networks, like Wireless Local or Metropolitan Area Networks (WLAN or WMAN) are increasing their appeal to the final user that could have access to data without being tethered to any static infrastructure. In a near future digital TV (DVB-H) and the successor to 3G, Long Term Evolution (LTE) will also be included in feature rich handsets. Furthermore the idea of software defined radio has thrilled many people over the last decades. SDR promotes flexibility and hardware reuse which makes it very appealing for companies seeking to implement flexible multi-standard radios in the future. Supporting all these different standards, obviously, should be achieved without any degradation in term of achievable data rate or transmission reliability. In order to achieve the required flexibility to support all these standards and to reach optimal solutions, flexible platforms are necessary.

In the last few years, the improvements in the micro-electronic technology have made conceivable the integration on the same Integrated Circuit (IC) millions of MOS transistors and logic gates. This makes possible the design of novel integrated architecture with enhanced capabilities compared to those designed in the first years of the nineties. This revolution was partially driven by the astonishing growth of the telecommunication market where the need of more services implies higher bandwidth and data rates. To tackle with these requirements, novel telecommunication infrastructure should be investigated, even if they require augmented computation capabilities. These capabilities could be achieved by means of the evolution of the micro-electronics circuits. The increased integration density, in fact, allows to implement on the same silicon die, different circuits. However, these augmented possibilities requires novel design paradigms in order to catch all of them. In particular the augmented design cost, and the augmented requirements of the novel communication standards impose to exploit flexible structures able to easily adapt to different working conditions. Nevertheless, before analyzing these architectures, some brief analyses of this novel design paradigm of flexibilty is necessary.

2.2 Flexibility as a Design Paradigm

Traditional Application Specific Integrated Circuit (ASIC) design flows have always tried to maximize the circuits' operating frequencies, while keeping low the hardware complexity, intended as the whole number of logic gates and the required circuit area. Moreover the augmented number of battery-supplied portable devices impose taking into account also the power consumption leading to more power-conscious design flows. Summarizing, we can state that typical design constraints are frequency, complexity and power. The improvements of the technology lead to MOS transistor with minimum features that shrink to extremely short dimensions in the order of the nanometer scale. Current gate length, for commercial technologies are in fact equal to 90 nm and 65 nm while in the next generation it is foreseen that it will become equal to 45 nm or even less [ITRS, 2003].

These minimum features allow integrating several millions of transistors on the same chip allowing, at the same time, to design novel ASICs with increased computation capabilities, working at higher frequencies and with limited area and power requirements.

Even if the current technologies enable the design of advanced architectures, they nevertheless pose some severe limitations in the integrated circuit design too. In particular, Non-Recurrent Engineering (NRE) costs related to design and masking costs have increased to such a point that ASICs production become economically viable only for mass productions. Moreover, time-to-market is shortening, reducing the design time and limiting the possibility of correct design errors in successive iterations. Additionally, in current telecommunication standards different operating modes are prescribed; thus, the more different working conditions a single circuit can tackle, the more market this circuit will gain and the more time it will be present (i.e. it will last) in this market. This means that this circuit should be reusable without any changing in his internal architecture. This will also extend is lifetime allowing mass production and the absorption of non-recurring costs. Thus, a novel design paradigm, the flexibility puts aside the traditional ones in the design flow [Benedetto et al., Sept. 2004.], [Masera and et al., Oct. 2004].

Hereinafter we will give a more formal definition of flexibility even if it can be hardly quantitatively measured in practical implementations. Flexibility is defined in terms of ability of the hardware platform to adapt to the different algorithms and its re-configurability towards different scale of an algorithm [Polydoros, Sept. 2008].

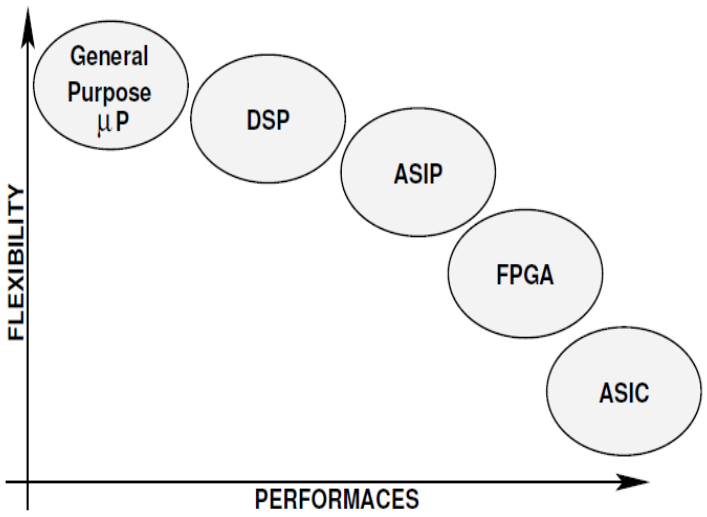

Figure 2.1: Performances vs. Flexibility of Different Digital Circuits

Fig. 2.1 shows the diagram relating flexibility and performances for some digital architectures . We can see that usually, the more flexible is an architecture the less performances it can achieve; in fact, their internal structures are not optimized (i.e. customized) to any specific application. A general purpose microprocessor (μP) is a digital computation unit able to perform different

task without any specialization. They are highly adaptable architectures, and their flexibility is given by the code running on them. However this high flexibility comes at the cost of overall performance. A Digital Signal Processor (DSP) is a particular kind of microprocessor with some specialized unit and instruction dedicated to signal processing. Thus, these specialized units reduced the overall flexibility since some instructions are optimized while the other are not, but allow better performances when signal processing computations should be performed. Like μPs, also DSP are adaptable circuits, since different task can be performed simply varying the code running on them.

An Application Specific Instruction-set Processors (ASIP) is processor with some specialized instructions and computational unit specialized for a specific task, [Coware]. Respect to DSP, ASIPs are optimized for only a specific task, so they are less flexible than DSPs. Moreover also their internal structure (bus width, arithmetic units and so on. . .) are designed in order to accelerate computations of the specific application. Thus, when an ASIP does elaborations related to the application it was designed for, it could achieve high performances. However, there are still some general purpose instructions and an ASIP could afford also different elaboration even if the obtainable performances are degraded.

FPGAs are reconfigurable circuits, since different circuits (i.e. this is where the flexibility came from) can be mapped rearranging the internal structure of the FPGAs themselves. On the other hand Application Specific Integrated Circuits or ASIC is a digital circuit design and optimized in order to perform a specific task (i.e. application)with best possible performances [Weste and Eshraghian, 1992]. Typically, ASIC is tailored around a specific application and may not exhibit any kind of flexibility.

2.3 Flexible Channel Decoding

As can be inferred from the previous analysis, a design flow that take into account flexibility from the beginning (i.e. from the definition of the problem that the architecture should tackle), will lead to circuits that can be easily adapted to novel and different working conditions. This is particularly true in the case of the telecommunications where novel standards, with novel enhanced operating modes are introduced. In particular, each new standard tries to increase the data rate, i.e. the number of information transmitted in a second, while keeping low the occurrence of errors in the transmissions. Moreover, depending on some external conditions, each standard provides different profiles. Thus, an integrated circuit designed for telecommunication purposes has to exploit a certain degree of flexibility in order to tackle all these profiles. Multi-standard devices moreover, are able to support different telecommunication standards and they may exhibit a quite long life-cycle since a single IC can be adapted to different conditions. More flexible architectures can have also support for future out-coming standards.

Among the different functionalities of such standards, one of the most demanding operations

is channel decoding which is the central processing part in the outer modem of telecommunication systems. Convolutional codes (CC), binary turbo codes (bTC), duo-binary turbo codes (dbTC), and low-density parity-check codes (LDPC) are established channel coding schemes. Especially the decoding algorithms of turbo codes and LDPC codes have a very high computational complexity. It is shown in [Lin et al., 2006], [Pan et al., 2003] and [Hosemann et al., 2003] that channel decoding contributes atleast 40 % to the total computational complexity of the physical layer of a wireless system, depending on the implementation platform. However, multiple variations of these algorithms are employed in standards. More specifically, every standard uses a different configuration of an algorithm which makes it unique to that standard. For example, the Turbo coding algorithm used in W-CDMA employs a different polynomial, block size, coding rate and termination method from that used in WiMAX. This translates to a further increase in complexity of a flexible channel decoding plateform. In the recent years, novel decoding schemes, able to improve transmissions close to the theoretical limits were presented and adopted in different standards (UMTS, DVB-S2, DVB-RCS, 802.16, . . .). These decoding algorithms are able to achieve high performances also in terms of data rate. However, novel codes with enhanced properties with better overall performances are still investigated and these novel solutions might be adopted in future standards. Thus, flexible architectures able to support as much options as possible seems the only viable solution for reducing overall costs.

2.4 Problem Space Formulation

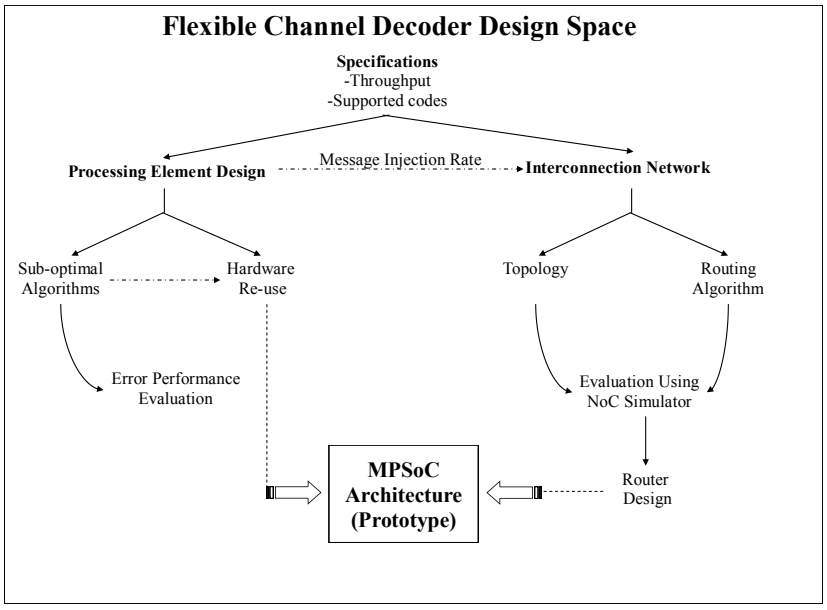

Figure 2.2: Problem space of flexible channel decoder design

The implementation of complex (iterative) channel decoding systems became essential to reach the performances now required in term of quality of transmission. Dedicated hardware architectures (i.e. ASIC) implementing parts of these systems are already tackled in several academic and industrial research teams. However, the requirements of: very low error rate, very high throughput and increased flexibility of the implementation, make the resort to adequate multiprocessor architectures inevitable. In this context, Multi-Processor System-on-Chip (MPSoC) architectures are being widely investigated these last years in order to accommodate the increasing throughput and flexibility requirements of emerging wireless communication standards. MPSoC architectures for iterative decoders are detailed in *chapter 4* of the thesis. In multi-processor implementations several independent data blocks can be simultaneously decoded on different processors, this approach multiplies the costs (memories, area, and power consumption) along with the throughput. Exploiting the inherent parallelism of the iterative decoding algorithms like turbo and LDPC enables a far more efficient partitioning of the decoding task. The block to decode can be divided into several sub-blocks. Decoding each sub-block on an individual processor significantly reduces latency as a critical parameter in many communication applications, and also the memory overhead. Here application specific processing element design capable of supporting different channel decoding algorithms and their different scales constitute challaneging subset of flexible channel decoder architecture design problem (shown in Figure 2.2). It can be seen that, other than the different supported codes, the system specification requirement of the required decoder throughput is one of the important factor to keep in mind while designing the processing element. The need to achieve higher throughput pushes the demand for sub-optimal algorithms design of related code encoding and decoding procedures for low latency implementation, which however need to be effciently validated for their error performance robustness. In addition a reuse of hardware resources (datapath, memories) across different decoding algorithm implementations needs to be explored while designing a flexible proecessing element.

Furthermore due to the iterative exchange of data between processing elements each processor working on the same data block has to communicate with each other, yielding only limited locality. A communication network has to support the communication demands of the different processing elements without degrading the throughput of the overall system, which is again an additional subset of the flexible decoder design problem space as seen in Figure 2.2. Conventional on-chip buses become inefficient in large systems and the nanotechnology integration issues (propagation delay, crosstalk, etc.) make their use no more practical. In this context, Network-on-Chip (NoC) has recently emerged as a new paradigm allowing to cope with these major design issues, and more particularly with the on-chip interconnection issues, and to accommodate future onchip integration of several hundreds of components. However, incorporating NoC paradigm in flexible decoder implementation is not straightforward, requirement of decoder throughput determines the network size and the configurations of designed processing elements impacts the selection of right topology

and efficient routing algorithm for achieving application throughput. Designing of the processing elements and the communication structure involve the exploration of several alternatives that need to be investigated, functionally validated, characterized and compared in terms of cost and performance, after all these steps only the final prototype can be obtained. In the following sections we present state of art work for these different subsets of problem space of flexible channel decoder design.

2.4.1 Sub-Optimal Algorithms

Challanges of designing flexible processing elements include low power, low cost implementation of individual decoding algorithm supported. Various suboptimal approaches are proposed in the literatures for low power and low cost implementations of turbo and ldpc decoders. Memory being dominant part of decoder implementations many efforts have been focussed on reducing the memory footprint. As explained in *chapter 1* the optimal decoding algorithm for CTC is the iterative decoding using the maximum *a posteriori* probability (MAP) algorithm [Bahl et al., Mar. 1974]. For convenience of implementation, all the computations are typicallly performed in logarithm domain, which is known as log-MAP [Robertson et al., Jun 1995]. Research on the implementation of log-MAP has focussed on power reduced memory orgnization [Schurgers et al., April 2001], [Wang et al., Dec 2002], [Lee and Park, June 2006] and sub-optimal decoding methods such as Max-log-MAP algorithm [Woodard, Nov 2000], enhanced Max-log-MAP [Vogt and Finger, Nov 2000], and linear approximation [Hsiung et al., June 2008]. When CTC is extended from single binary to Duo-binary, the basic decoding method remains the same, however more extrinsic informations are generated at the receiver, increasing the memory requirement of extrinsic storage significantly.

The size of the extrinsic memory, complexity of the interleaver, and the communication resources of the network for these implementations, scale with the reduction in the bit width of the exchanged extrinisic information. In addition to this, a reduced energy consumption is also achieved. There have been previous efforts in finding out the optimal quantization of extrinisic information to be exchanged. In [Montorsi and Benedetto, May 2001], [Jeong and Hsia, Sept 1999], [Castellon et al., May 2005] results for optimal quantization of extrinsic information were presented with the objective of complexity and memory size reduction of the processing element. In [Park et al., May 2008] an algorithm was proposed to decreases the necessary bit description width of the extrinsic information by employing a pseudo floating point representation. The work in **chapter 3** furthers this exploration by proposing a novel extrinsic bit-width reduction methodology.

Similar to turbo decoders various suboptimal LDPC decoding algorithm (Min-sum, λ-min, A-min etc.) exists in literatures as mentioned in *chapter 1*. Quest of power efficient architectures is driving the research for finding efficient sub-optimal LDPC algorithms, however in thesis we do not deal with them.

2.4.2 Interconnection Network for Flexible decoder implementation

In the context of designing interconnection network for flexible decoder implementation it should be noted that, MPSoC Architectures for iterative decoders present the Intra-IP communication paradigm, which is significantly different from inter IP communication in heterogeneous SoCs [Vacca et al., 2009]. Efficient NoC architectures like Mesh, Ring, Benes, hypercube etc. have been evaluated for inter-IP communication in heterogeneous SoC paradigm where it is much easier to classify the traffic because of non uniformity and locality.

In parallel turbo decoding, extrinsic information is iteratively and concurrently exchanged between component decoders. For each iteration, these exchanges become more and more massive with decoder level parallelism. In addition, the communication traffic profile depends on the turbo code interleaving rule. Since turbo code interleaving rule varies from one standard to another and/or one mode to another; the requirement of a fully flexible on-chip communication network interconnecting the two component decoders implies its ability to support the intensive interleaved memory accesses induced by parallel turbo decoders and to convey any permutation from the network input ports to its output ports. To that purpose, several application specific on-chip communication networks were recently introduced. Topologies based on Benes, Butterfly, 2D Mesh, chordal ring, and de Bruijn were explored, designed, and integrated for an MP-SoC decoder implementation [Moussa et al., 2007], [Thul et al., 2002], [Moussa et al., May 2008].

Similarly to parallel turbo decoders, partially-parallel code-independent LDPC decoders, which offer the best compromise in terms of area, throughput, and flexibility [Masera et al., June 2007], introduce a problem of conflicting accesses to the memories containing the exchanged messages. To manage this issue, the authors in [Tarable et al., Sept. 2004] propose an efficient algorithm which can compute a collision-free memory mapping of interleaving laws with no constraint imposed on the code itself. Thus, a versatile implementation requires for each supported code a pre-processing step to recompute the corresponding memory mapping. Besides, considering the various block sizes and codes rates increase significantly the flexibility requirement. In addition, the computations being relatively simple, implementation issues mainly come from the communication structure between the VNPs and CNPs. Indeed, the communications rapidly become intensive because they depend on the number of nodes, the node degrees and the number of iterations. In this situation, a flexible on-chip interconnection network must be designed with the aim of fully exploiting the parallelism of the LDPC/turbo decoder architecture by reducing the message latency, alleviating the memory conflicts and efficiently routing any permutation from the network input ports to its output ports.

However, none of these works deal with performance/complexity tradeoff of NoC versus different parallelism degrees, message injection rates and routing algorithms etc. used in MPSoC based iterative decoder depending on the target throughput requirements. Such an analysis is needed for minimizing the network area overhead for a target decoder throughput. The work in **chapter 4**

provides such a case study using real life iterative decoders supporting WiMAX standard.

2.4.3 Hardware Reuse in Flexible decoder design

The problem of designing flexible processing elements also includes sharing of datapath and memories across different forward error correction (FEC) decoder implementations as seen in Figure 2.2. Since quite a long time, issues of flexible architectures for codes within a FEC family have been dealt with in research. In [Muller et al., Mar. 2006] a flexible and high performance Application-Specific Instruction-set Processor (ASIP) model for turbo decoding was proposed which can be configured to support all simple and double binary turbo codes up to eight states. On design of flexible LDPC decoder plateform in [Brack et al., Sept. 2006] a highly flexible LDPC decoder architecture capable to process all specified WiMaX LDPC codes was proposed.

In last few years however many research activities have emerged proposing implementations in order to achieve flexible and high throughput decoder architectures for supporting the flexibility across different FEC code families. Few approaches combining the turbo and Viterbi decoding have been reported. Bickerstaff et al have proposed a unified architecture designed for UMTS base stations [Bickerstaff et al., Nov. 2002], [Thomas et al., April 2003]. The main emphasis is on the multi-channel aspect, and the flexibility in coding schemes has not been handled in this work. Mainly the memories are shared between the turbo and Viterbi modes. In [Kreiselmaier et al., Feb. 2004], another combined architecture is suggested for wireless terminals. In this architecture the datapath is shared as well as the memories. A MAP algorithm is used for decoding both convolutional and turbo codes. This is, however, only possible when the throughput requirement for Viterbi decoding is much lower than that of turbo decoding (e.g. 12.2 kbps for Viterbi and 384 kbps for turbo). In another effort to combine the two types of decoders soft Viterbi decoding is used for the turbo iterations and hard output Viterbi decoding is used for convolutionally encoded signals [Cavallaro and Vaya, April 2003]. A unified decoder architecture for LDPC and turbo codes has been presented in [Sun and Cavallaro, Oct. 2008] where multi-mode decoding is achieved by employing a flexible add-compare-select (FACS) unit. By representing LDPC codes as parallel concatenated single parity check (PCSPC) codes, they have tried to efficiently reuse the turbo decoding infrastucture for LDPC decoding functions.

In the later years when focus was shifted towards ASIP architecture authors like [Vogt and Wehn, Mar. 2008.], have presented the FlexiTreP ASIP family which supports trellis based channel codes, i.e., convolutional, binary and duo-binary turbo codes for various standards. In [Alles et al., Sept. 2008] a memory sharing across turbo and LDPC code in an application specific processor (ASIP) was explored. The FlexiTreP core is merged with an LDPC ASIP core into a single ASIP, named FlexiChaP, a flexible channel coding processor, which is capable to support convolutional codes, binary/duo-binary turbo codes, and structured LDPC codes. It was shown that because of the efficient memory sharing across turbo and LDPC decoding area increases in FlexiChap

($0.39mm^2$) only slightly compared to FlexiTrep ASIP ($0.31mm^2$). However sharing of datapath logic area (which is about 25% of the complete ASIP) across these two families were considered insignificant. Very recently in [Naessens et al., Oct. 2008] authors proposed the application-specific instruction programmable architecture addressing in a unified way the emerging turbo and LPDC coding requirements of 3GPPLTE, IEEE802.11n, IEEE802.16(e) and DVB-S2/T2 where datapath and memory reuse across different FEC families has been used.

From the analysis of the state of the art in hardware reuse in flexible channel decoder design two important points can be gathered. First, memory sharing across different codes definitely is the bigger player in overall reuse scenario, however datapath logic which could be up to 25% of the whole decoder complexity do present an interesting area of exploration for hardware reuse, mostly because of a multiplicity of ways to implement different FEC decoding algorithm. Secondly, it can be seen that a comprehensive analysis of datapath reuse providing the designer with the choice to decide on the different possible ways of implementing basic FEC computation units is still lacking. The work in **chapter 5** deals with some interesting aspects of such an analysis.

2.4.4 Performace Evaluation of Sub-optimal Iterative Decoding Algorithms

A large number of sub-optimal algorithms to be tested in quest of flexible, low power, low cost implementation of iterative deocders. The problem of finding a good performance-complexity trade-off is thus a complex task. Moreover, the evaluation of each configuration generally requires a CPU extensive Monte-Carlo simulation in order to have an accurate estimation of the BER. In order to avoid such simulation, in [Boutillon et al., June 2007] authors propose an efficient reduced monte carlo simulation (RMCS) method which provides an improvement of simulation speed by a factor of 10^3. The key steps of the proposed approach are as follows:

- **step 1**: Define the space of the search by defining the range of search for each parameter of the decoder. Define a model of complexity for each parameter. Define also the maximum allowable complexity of the design.
- **step 2**: Define the "worst case" configuration by individually setting the parameters to the value that degrades performance most.
- **step 3**: Using this configuration, perform a Monte Carlo simulation at the SNR of interest. Each time a received codeword fails to be decoded, store the codeword (or information to reconstruct it, i.e., the seeds of the pseudo random generators) in a set S. Stop the process when the cardinality of the set S is high enough (typically around 1000). Note that this operation can be very CPU consuming but it has to be done only once.
- **step 4**: Perform an optimization in order to find the set of parameters that minimize the BER (or the FER) over the set S with the constraint that the overall complexity of the decoder

remains below a given value.

- **step 5**: Perform a *normal* Monte Carlo simulation in order to verify *a posteriori* the real performance of the selected parameters. Go to step 1 with a different scenario of optimization if needed.

It can be seen that the method is an intelligent adaptation of traditional MC simulation method and gives faster evaluation as a result of its operation on smaller set of codewords. However it essentially operates on BER domain and from APP to BER, a huge quantity of information is suppressed. This observation leads us to a question of whether it is possible to utilise this APP information for speeding up the performance evaluation of iterative decoders of different complexity? The work in **chapter 6** explores possible answers to this question.

2.5 Contribution of the Thesis

Based on above classification of subsets in the problems space of flexible channel decoder design our contribution can be summarized as following:

- **Suboptimal algorithms**: The methodology for bit-width reduction of extrinsic information was developed which not only reduces memory also it has imact on complexity of permutation network which are used to piggback extrinsic information between iterations in a turbo decoder [Singh et al., Sept 2008].

- **Interconnection Network**: Evaluation of NoC parameters for MPSoC based Turbo and LDPC decoder architectures is done. By a detailed traffic analysis we not only showed that the overall processing throughput is impacted by the network's limited communication bandwidth, in addition choice of parallelism degree and design of processing unit also play an important role [Scarpellino et al., August 2008].

- **Hardware Reuse**: Design and implementation of a flexible FEC kernel which supports the various channel decoding algorithm used in current wireless standards viz. Viterbi, Turbo and LDPC [Singh et al., 2010].

- **Performace Evaluation**: Developement of a new metric for fast and efficient performance evaluation of iterative decoding algorithms [Singh et al., July 2009].

2.6 Conclusion

In this chapter, we have tried to explore flexible channel decoder design problem space. This space features various subsets related not only with algorithm design and their validation methodologies,

but architectural choices and their efficient implementation aspects are also dealt with. This classification of subsets help to describe different dimensions of the complexity of the flexible decoder. The combinations of the different components of this model enables us to describe the state of the art flexible channel decoder architectures and methods and also build the motivation for different contributions of the thesis. The chapter then presents these contributions, which will be fully detailed in the following chapters.

Chapter 3

Bit Width Optimization of Extrinsic Information in Turbo Decoder

Contents

Soft Input Soft Output (SISO) decoders iteratively exchanging intermediate results (extrinsic information) between themselves lie at the core of turbo decoder architectures. In this chapter, first a brief overview of two kinds of turbo codes: serial concatenated convolutional code (SCCC) and parallel concatenated convolutional code (PCCC) is presented along with their decoding aspects. Then various implementation architectures of these codes are categorized as either serial, parallel or network on chip (NoC) based. For each of these categories role of the extrinsic information's

bit-width on implementation complexity is analyzed. Finally a technique for bit-width reduction of exchanged extrinsic information in turbo decoders is presented and the impact of it for different implementation architectures is analyzed. The methodology is investigated over turbo decoding system based on the Max-Log-MAP algorithm. The material of the second part has been partly presented during the Turbo Symposium conference [Singh et al., Sept 2008].

3.1 Introduction

Extrinisic information in typical turbo decoding indicates a degree of confidence associated with each bit. There have been previous efforts in finding out the optimal quantization of extrinisic information to be exchanged. In [Montorsi and Benedetto, May 2001], [Jeong and Hsia, Sept 1999] and [Castellon et al., May 2005] results for optimal quantization of extrinsic information were presented with the objective of complexity and memory size reduction of the processing element. In [Park et al., May 2008] an algorithm was proposed to decrease the necessary bit description width of the extrinsic information by employing a pseudo floating point representation. Instead, we explore a new approach for optimal quantization of exchanged extrinsic metrics, where the underline assumption is that optimal fixed point representation has already been obtained for the component decoder's internal signals. We propose a technique in which most significant bit (MSB) clipping combined with least significant bit (LSB) drop (at transmitter) and append (at receiver) is used as a way for bit-width reduction across the communication structure.

3.2 Turbo Codes and Decoding

This section presents a brief description of the two classes of turbo codes, SCCC and PCCC along with a general structure of their encoding and decoding architecture. These codes are introduced here as the bit-width optimization methodology on extrinsic information will be evaluated for their decoder implementations.

3.2.1 Parallel Concatenated Convolutional Codes

The concept of parallel concatenation (PC) traces back its origin to the introduction of turbo codes in 1993. A general structure of PCCC encoder and decoder is shown in Figure 3.1. In PCCC the message to be transmitted is encoded two times: first encoder processes the information in its natural order, while the other encoder processes it in the interleaved order. Analyzing Fig. 3.1.a it appears clear that in PCCC encoders the two constituent codes work on the same set of input data u even if it they are a "scrambled version" for the second RSC block.

The decoder of PCCC processes LLRs at their inputs $\lambda(\ ;\ I)$ and outputs $\lambda(\ ;\ O)$(see Figure 3.1.b. Each SISO decoder computes the extrinsic LLRs related to information symbols $\lambda(\ u_1\ ;$

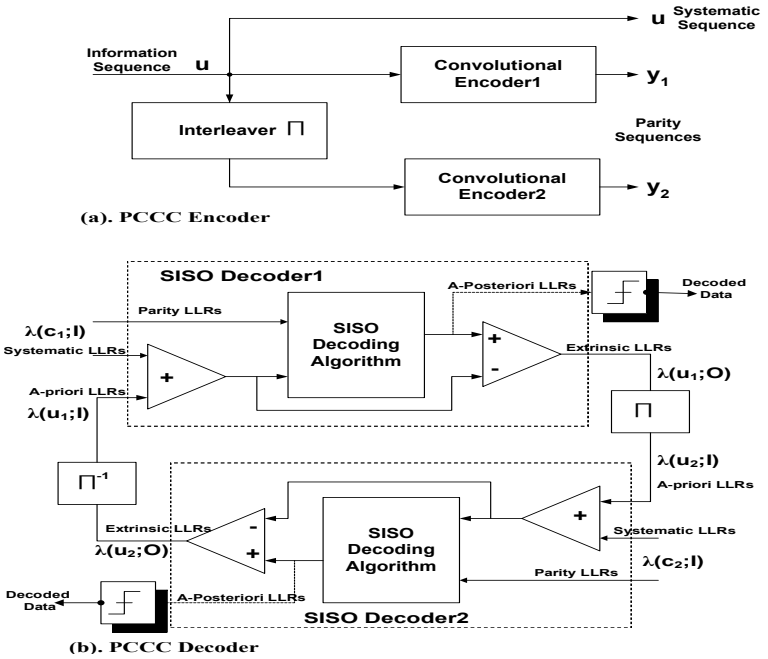

Figure 3.1: Turbo codes: (a).Parallel Concatation Convolutional Code Encoder (b).Parallel Concatation Convolutional Code Decoder

O) and $\lambda(\ u_2\ ;\ O)$, using the observation of the associated systematic and parity symbols coming from the transmission channel $\lambda(\ c_1\ ;\ I)$ and $\lambda(\ c_2\ ;\ I)$, and the *a priori* LLRs $\lambda(\ u_2\ ;\ I)$ and $\lambda(\ u_1\ ;\ I)$. Since no *a priori* LLRs are avalaible from the decoding process at the beginning of the iterations, they are then set to zero. For the subsequent iterations, the extrinsic LLRs coming from the other decoder are used as a priori LLRs for the current SISO decoder. The decisions can be computed from any of the decoders. In the PC case, the turbo decoder structure is symmetrical with respect to both constituent decoders. The decoder structure is symmetrical with respect to both constituent decoders. The two SISO processes execute in sequential fashion, for example, after SISO1 processing is finished, SISO2 starts the processing and so on [Masera, 2005] or could work in parallel in interleaved and natural domain [Juntan and Fossorier, Feb. 2005]. In the latter approach also known as "shuffled decoding" nevertheless, to preserve BER performance, additional iterations are required.

3.2.2 Serial Concatenated Convolutional Codes

Figure 3.2 shows the general structure SCCC encoder-decoder pair. As shown in Fig. 3.2.a the output of the outer encoder (c_o) becomes the input of the subsequent inner encoder (u_i). It is evident that the decoding structure is no more symmetrical. The inner SISO decoder computes extrinsic LLRs $\lambda(u_i;O)$ related to the inner code information symbol, using channel information $\lambda(c_i;I)$ and extrinsic LLR coming from other SISO decoder ($u_i;I$). The outer SISO decoder uses

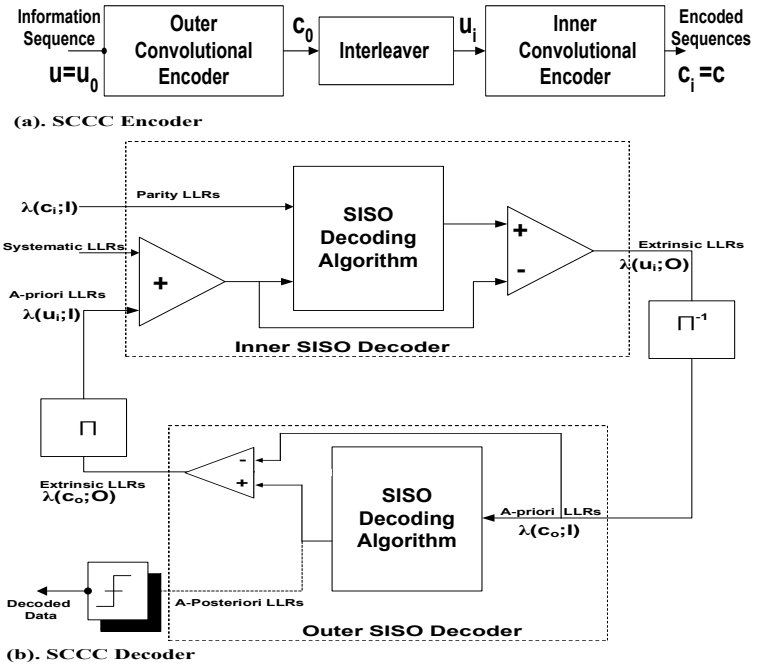

Figure 3.2: Turbo codes: (a).Serial Concatation Convolutional Code Encoder (b).Serial Concatation Convolutional Code Decoder

extrinsic LLRs provided by the inner decoder and computes the extrinsic LLRs (c_o;O). The outer SISO decoder computes the decisions related to information symbols as a posteriori LLRs (u_o;O).

3.3 Turbo Decoding Architectures

In this section turbo decoder implementation architectures are classified into three broad categories: serial implementations, parallel implementations with deterministic interleavers and network on chip based parallel implementations. The impact of the extrinsic information's bit-width on implementation complexity is also analyzed for each of them.

3.3.1 Serial Architecture

The simplest decoding architecture to decode a block of codeword is based on the use of a single SISO decoder, which alternatively processes both constituent codes. This serial architecture requires three storage units: a memory for the received data at the channel and decoder outputs (LLR_{in} memory), a memory for extrinsic information at the SISO output (EXT memory), and a memory for the decoded data (LLR_{out} memory). An instantiations of this architecture in the PCCC case is shown in Figure 3.3. In the PCCC scheme, the decoding architecture is the same for both component codes, since they play the same role in the overall decoding process. The SISO decoder decodes code 1 or 2 using the corresponding channel data from the LLR_{in} memory and the a priori information

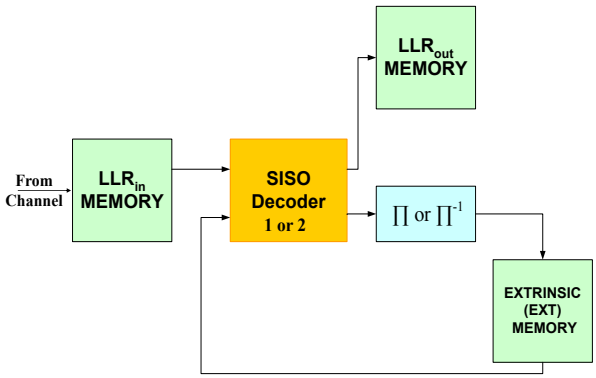

Figure 3.3: Turbo decoder serial implementation(PCCC)

stored in the EXT memory at the previous half-iteration. The resulting extrinsic information is stored in the EXT memory and then used as a priori information at the SISO input when the other code is processed, at the next half-iteration. The decoded data are written into the LLR$_{out}$ memory at the last half-iteration of the decoding process. In the SCCC scheme, the architecture elements are same however they are used in the slightly different fashion for inner and outer code processing [Boutillon et al., June 2007].

The important thing to note in both kinds of turbo-code decoding architectures is that the extrinsic memory has an implementation complexity proportional to the number of bits to be stored. Bit width reduction of extrinsic information not only results in memory area saving, a reduced energy consumption is also achieved, as memory power dissipations consititutes a major part of total turbo decoder power consumption budget depending on implemented architecture. For example in [Schurgers et al., April 2001] authors presented memory power dissipatation to be 50% of the total power consumption of their implemented architecture.

3.3.2 Deterministic Permutation Network Based Parallel Architecture

For high throughput application, parallel architectures of turbo decoders are required. The block of data to be decoded of frame size K is sub divided into P sub-blocks, of size M. Each of the sub-blocks is processed by P independent SISO processing units (see Figure 3.4). Access of any memory bank is facilitated by the use of its associated address generator (AG). The extrinsic informations are exchanged across multiple SISO processing units using a permutation network. Some of the most commonly used network for this purpose are Benes and De-brujin. An appropriate initialization is required for the forward f_m and backward b_m metrics [Blankenship et al., June 2005a], [Giulietti et al., Feb. 2002]. It can be observed that in the deterministic permutation network based parallel implementation, permutation network has an implementation complexity proportional to the number of bits to be routed. Reduction of bit width of exchanged extrinsic not only reduces

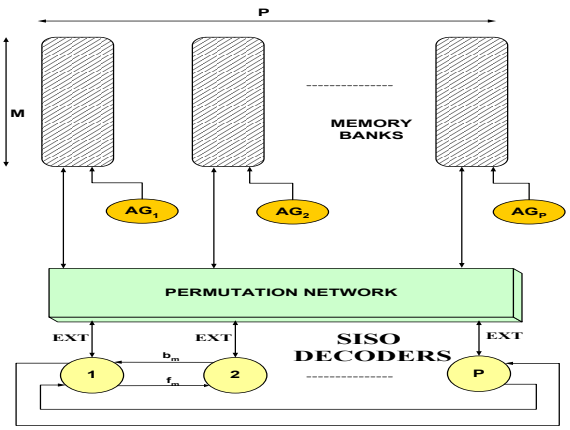

Figure 3.4: Turbo decoder parallel implementation

the memory bank size, it also significantly reduces the complexity of the permutation network. Furthermore, as a result of reduced memory size, corresponding column read/write hardware of memory is also reduced.

3.3.3 Network on Chip based Parallel Architecture

In [Neeb et al., May 2005] networks-on chip capable of resolving access conflicts at run-time is detailed to support arbitrary interleavers without any pre-processing. Such an approach offers great flexibility with respect to implementing various permutation patterns for standard compliant decoders. NoC based architectures are treated with great detail in *chapter 4*. In this section as a brief in Figure 3.5 the general NoC based parallel turbo decoder architecture is shown, where each processing element (PE) is a SISO processing unit with corresponding memories. The extrinsic information exchange across different PEs is facilitated by the NoC. Choice of the interconnect topology determines the achievable decoding throughput. In [Scarpellino et al., August 2008] it was

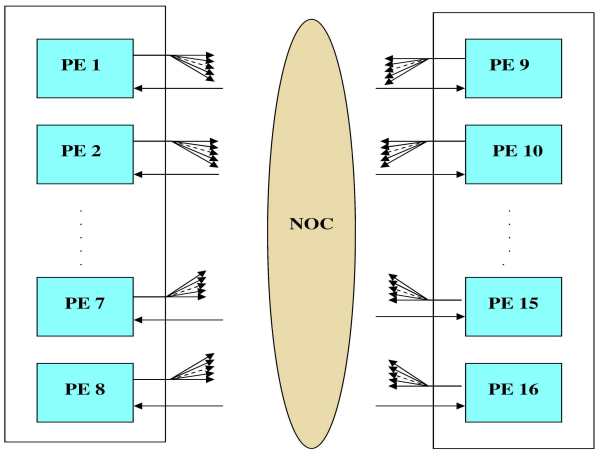

Figure 3.5: Turbo decoder NoC based implementation

shown that for a given interconnect topology, due to the limitation of routing resources (switching and interconnect wires), accepted traffic by the network tends to saturate after a certain value of injection load, resulting in reduced decoding throughput. Reducing the bit-widths of exchanged extrinsic information is an effective way to allocate more communication resources and increase the accepted traffic saturation point, thus increasing the decoding throughput.

3.3.4 Special Case of Duo-Binary CTC

In the context of impact of extrinsic information's bit-width on implementation complexity, the special case of Duo-Binary CTC needs to further detailed. Duo-Binary CTC decoder can be implemented using any of the three architectures mentioned above, however important thing to note here is that 3 LLRs per symbol are generated and iteratively passed acrossed component decoders. In the three architectures mentioned above employing duo-binary CTC decoding extrinsic bit-width reduction will result in higher complexity reduction (by a factor of 3) than respective architecures for binary turbo codes.

3.4 Bit-Width optimization

The problem of fixed-point implementation is important as hardware complexity increases linearly with the internal bit width representation of the data. The trade-off between hardware complexity and error correcting efficiency leads to the minimum bit width internal representation that results in an acceptable degradation of performance. However, in this work we assume that the internal bit width representation of the data has already been derived for minimal performance degradation. We investigate another paradigm where we focus our efforts to decrease the bit width of the extrinsic information sent over the communication structure between component decoders, without much degradation in BER performance of the decoder. With a C based simulator which is capable of simulating whole operation of data transmission for turbo encoding and decoding some results were obtained to estimate the distribution pattern of the extrinsic information exchanged between inner and outer siso. The simulator starts from data creation, goes on with encoding, puncturing, transmission in a noisy channel, de-puncturing, to finish with the whole decoding process, including parallel siso-decoding and interleaving/de-interleaving. Table 3.1 provides the parameters of the simulated model. The serial code used is from the implementation of a very high speed (1Gbps) adaptive coded modulation modem for satellite applications [Benedetto et al., April 2005]. Figure 3.6 shows the distribution pattern of extrinsic information in SCCC turbo decoder, where parameter "Frequency" measures the number of extrinsic informations for a given value. The distribution plots shows the Gaussian nature of the extrinsic values as observed in [Isukapalli and Rao, Nov. 2003]. The evolution is gradual with number of iterations.

Table 3.1: Parameters for Extrinsic Distribution Simulations.

Parameters	**Value**
Block size	510
Permutation size	768($RateOuter$: 0.662338)
Code size	1540($RateInner$: 0.499350)
Rate	0.331
Number of symb.	770
Modulation	$4PSK$
Spectral eff.	0.662
Es/N0	$0.80dB$
Iterations	8
Bits EXT	8

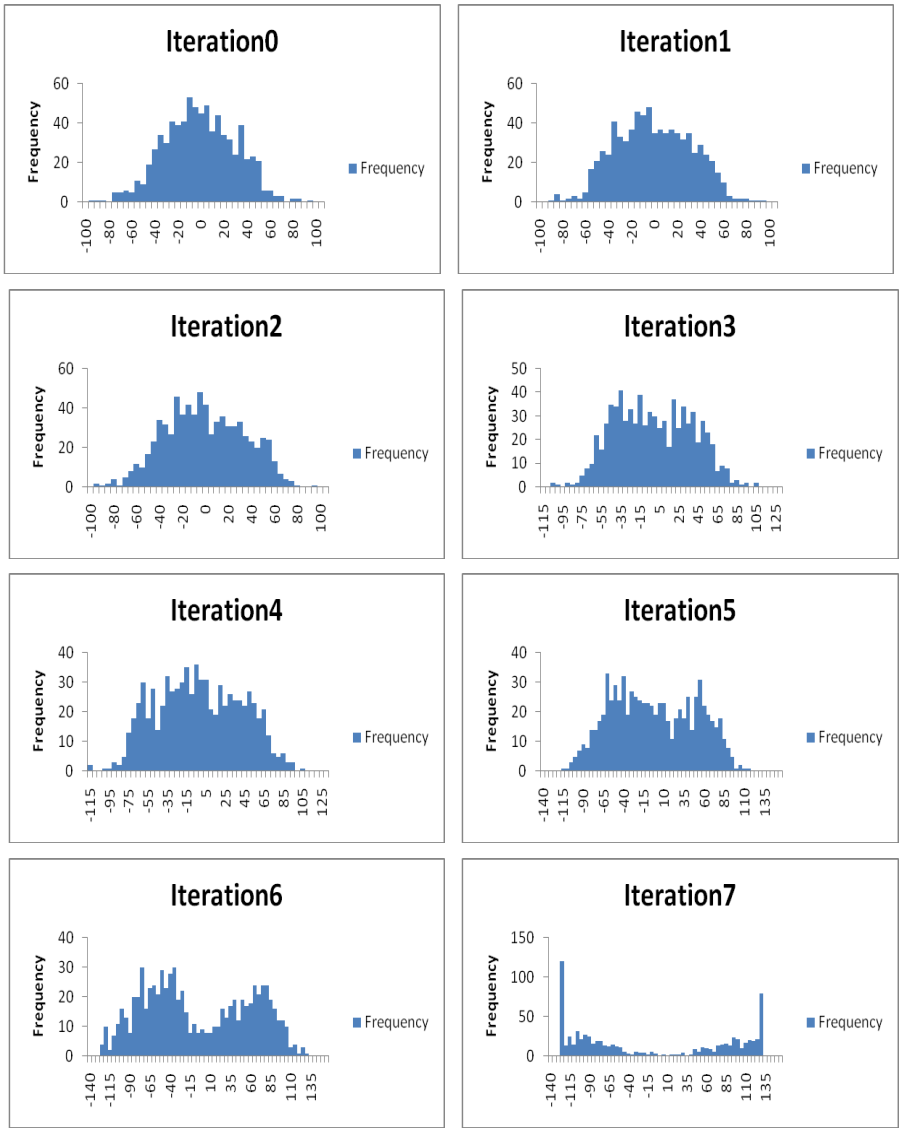

Figure 3.6: Extrinisic distributation over iterations with parameters given in Table 3.1

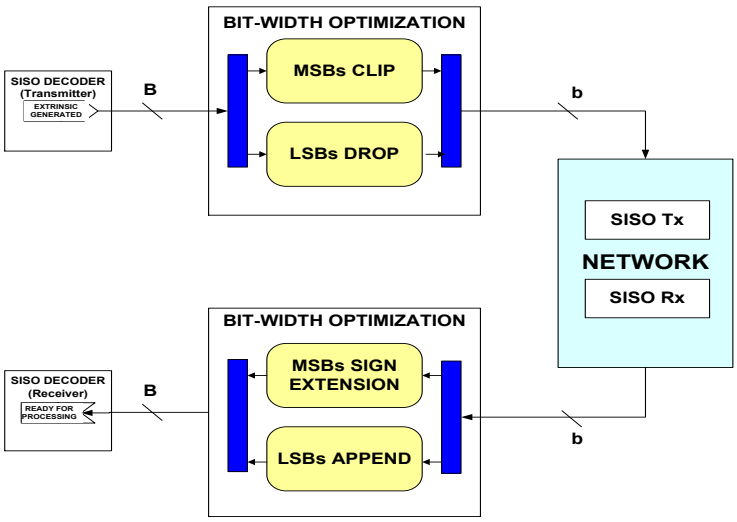

Figure 3.7: Bit-width Optimization Process

Our goal is to obtain the BER performance results using smaller bits to represent the extrinsic information (originaly represented by 8 bits) taking into consideration their Gaussian distribution and the evolution of the pattern over iterations. As the extrinsic distribution evolution from one iteration to other is not uniform any linear bit-width reduction strategy will result in significant performance degradation. Thus a need of adaptable methdods arises, some of such strategies are discussed in the following sections.

3.4.1 MSB clipping and LSB drop-append

We analyze the impact of MSB clipping and LSB drop-append across the communication structure and resulting impact on the error performance. In Figure 3.7 a Unified Modeling Language (UML) based activity diagram model of our methodology is presented. A parallel implementation is considered with a network for extrinsic information exchange across multiple SISO units. After the first half iteration, extrinsics with bit-width B are generated at the transmiter SISO decoder which are then reduced to b bits ($B > b$) and sent across the network. At the input of the receiver SISO decoder, b bits are coverted back to the original B bit-width for SISO processing. The methodology can be equally applied to serial implementation architectures of the turbo decoder mentioned in the previous section without loss of generality. In this case, the network block in Figure 3.7 is replaced with extrinsic memory EXT.

3.4.1.1 MSB Clipping

The MSB bits are clipped at the transmitter component decoder and sign extension is done at the receiver component decoder. Let x_B (with B bit-width) be the extrinsic information at the output of the transmitter component decoder. Clipping is applied to cut M MSBs, mapping x_B

into $(x_B)^M$, which is sent across the network. At the input of the receiver component decoder, $(x_B)^M$ is transformed back to B bits as $\hat{x}_B$, which is fed to the component decoder for processing. The transformation can be formalised as Algorithm 1.

Algorithm 1: MSB Clipping

$x_B \Longrightarrow (x_B)^M \Longrightarrow \hat{x}_B$
If $\{|x| < (2^{(B-N-1)} - 1))\}$
$(x_B)^M = x_B$
ElseIf $\{x_B \geq 0\}$
$(x_B)^M = (2^{(B-N-1)} - 1)$
Else
$(x_B)^M = -2^{(B-N-1)}$
EndIf
$\hat{x}_B = (x_B)^M$;

3.4.1.2 LSB Drop-Append

Certain number of LSB bits at the transmitter are dropped and zeros are appended at the receiver component decoder. Let x_B (with B bit-width) be the extrinsic information at the output of the transmitter component decoder. LSB dropping is applied to remove L LSBs and mapping x_B into $(x_B)_L$, which is sent across the network. At the input of the receiver component decoder, N zeros are appended to obtain the original bit-width of B, mapping $(x_B)_L$ into $\hat{x}_B$, which is fed to the component decoder for processing. The transformation can be formalised as Algorithm 2, where $\lfloor x \rfloor$ represents the integer part of x.

Algorithm 2: LSB Drop Append

$x_B \Longrightarrow (x_B)_L \Longrightarrow \hat{x}_B$
$(x_B)_L = \lfloor \frac{x_B}{2^N} \rfloor$
$\hat{x}_B = (x_B)_L \times 2^N$

3.4.2 Error Performance Analysis

Applying the methodology of previous section, extensive simulations were performed to evalute the error correction performance of the turbo decoder with reduced word size allocated for exchange of extrinisic information. Different wireless communication services have different requirements on the transmission quality, e.g., for speech-services a BER of approximately 10^{-3} is sufficient, whereas for data-services BERs down to 10^{-6} are necessary in applications where data is delay sensitive. Therefore, the performance in the area of these functional points is of particular interest. The characteristics of the evaluation system are given in Table 3.2.

Table 3.2: Simulation Parameters. (SCCC and WiMAX PCCC Turbo codes)

Code	K	R	Modulation	Channel	Extrinsic Bits
SCCC	1022	0.332	*4PSK*	*AWGN*	8
PCCC	960	0.333	*BPSK*	*AWGN*	8

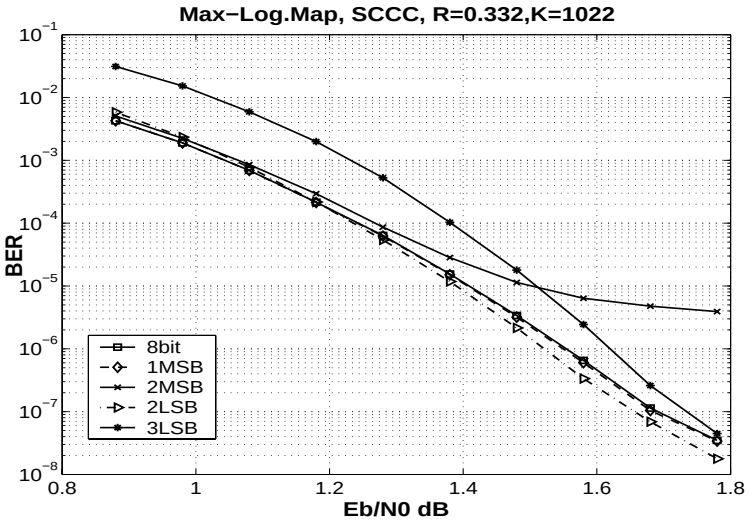

Figure 3.8: Separate MSB clip and LSB drop-append (SCCC Code at the end of 7 iterations)

The received channel values are 6 bit quantized with 5 bit for the integer part and 1 bit for the sign information. To improve the performance of the Max-Log-Map algorithm a correction term implemented as a look-up table is used for max function calculation [Benedetto et al., April 2005]. Figure 3.8 shows the BER performance (10^{10} bits simulated) for different bit-width choices at iteration 7. It can be observed that MSB clipping of 2 bits starts to give performance degradation of more than 0.1 dB at a BER of 10^{-5} and greater. Instead, there is more tolerance to LSB drop and append strategy. We can reduce 2 LSB bits without significant degradation in BER performance. For SNR higher than 1.25 dB, the LSB drop and append strategy of 2 bits provides a slight performance gain over the conventional system. We can hypothesize that this behaviour is related to the optimality of the extrinsic information. To decode Low Density Parity Check (LDPC) codes, in the Offset Min-Sum (OMS) algorithm [Chen and Fossorier, Mar. 2002], a multiplied or additive correction factor is directly applied to the check-node output of the original Min-Sum algorithm in order to improve the decoding performance. Inherently, the impact of drop-append technique could be compared to the offset value of OMS algorithm, however in drop-append technique the offset is not fixed and depends on the value of extrinsic information.

Figure 3.9 shows the BER results for different combinations of MSB clip and LSB drop-append strategies. At a BER of 10^{-4} opting 2 MSB clip and 2 LSB drop-append we get less than 0.1 dB performance degradation with respect to original 8 bit wordsize case. At higher BER this encoding

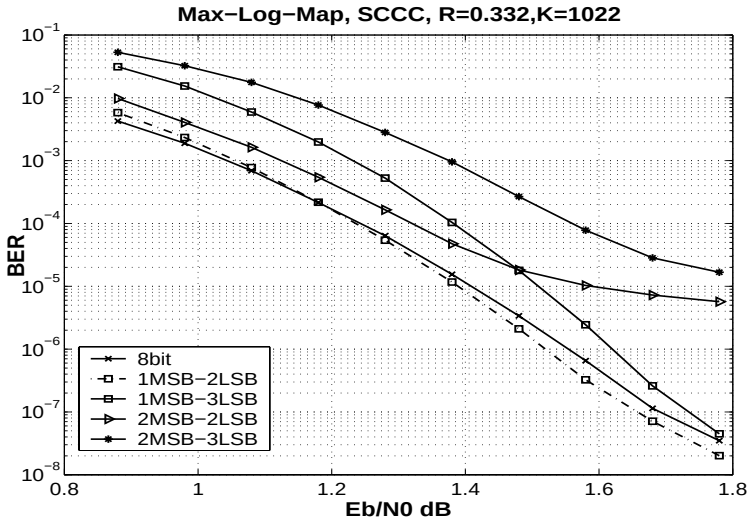

Figure 3.9: Combined MSB clip and LSB drop-append (SCCC Code at the end of 7 iterations)

starts to give greater loss, however the loss can be contained well within the 0.1 dB by switching to 1MSB clip and 3 LSBs drop-append strategy. Such a switching can be actuated based on the SNR information available at the receiver, resulting in a 50 percent reduction of the extrinisic information bit-width.

Figure 3.10 shows the BER performance (10^9 bits simulated) for different bit-width optimization for duo-binary codes used in the WiMax standard. The characteristics of evaluation system are given in Table 3.2. To improve the performance of the Max-Log-Map algorithm a scaling factor of 0.75 is used for the extrinsic scaling at each iteration. The BER values are at the end of iteration 7. The recieved channel values are 6 bit quantized with 3 bits for the fractional part and 2 bits for the integer part. The MSB represents the sign of the channel information. It is evident from the results that MSB clipping is not an option for wimax codes, but LSB (drop-append) methodology is effective. At a BER of 10^{-6} a 3 bit reduction in bit-width is possible if the loss of 0.2 dB is tolerable. Considering the fact that in duo binary turbo decoder three extrinsic information are passed on between component decoders, combined bit-width reduction is significant. It can be observed that the proposed technique provide good result on very diferent cases, such as a duo-binary PCCC and a binary SCCC. Thus this technique is not an ad-hoc trimming of the decoding algorithm, rather a general method to optimize turbo decoder implementations.

3.4.3 Memory and Power Consumption Reduction

As explained in section 1.3 memory area dominates the total implementation area of the turbo decoder and the decoder contains three main memories. They are dedicated respectively to soft input information, the soft extrinsic information, and state metrics. The width of the extrinsic information memory is $K \times x_B$ bits. In the SCCC decoder case, for a block size $K = 1024$, a

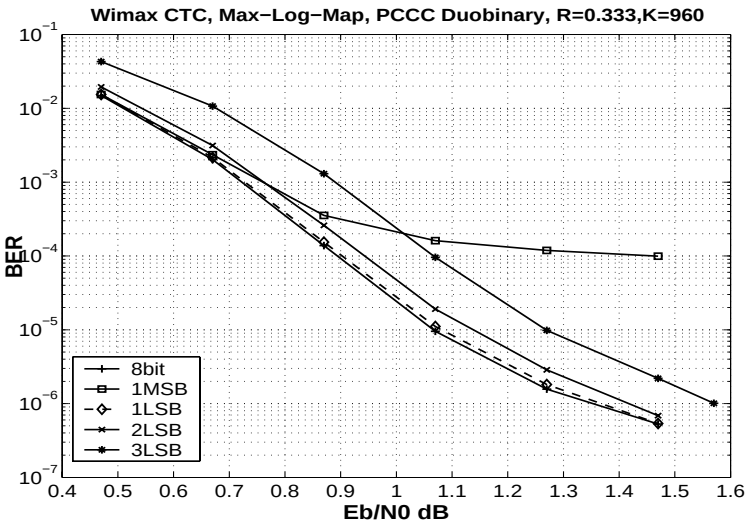

Figure 3.10: WiMax Bit-Width optimization at the end of 7 iterations

reducton of 4 bits of extrinsic bit width corresponds to a memory size reduction from 1 KBytes down to 0.5 KBytes. In the case of a WiMAX PCCC decoder, as the extrinisic information vector contains three extrinsic information related to the decoded symbol, the memory saving is more visible. For example, for a block length of 2400 couples, a reducton of 3 bits of extrinsic bit width corresponds to a memory size reduction of 7 KBytes downto 4.4 KBytes.

Static (leakage) power P_{lkg} is becoming a dominant source of power dissipation in next generation integrated circuits. It can be defined as :

$$P_{lkg} = V_{DD} \times I_{lkg}$$

where V_{DD} and I_{lkg} denote the power supply voltage and the leakage current respectively. Leakage power dissipation depends on the memory size and increases exponentially with submicron technologies [Mamidipaka et al., Sep. 2004]. Bit width reduction also reduces the amount of switched capacitances inside the memory device and on the driven bus lines, resulting in reduced dynamic power dissipation. Hence a reduction of the memory size has also a significant impact on the total power consumption of the turbo decoder.

3.5 A Further Effort at Bit-width Reduction

As an extension to our previous approach of bit-width reduction, some other algorithms were proposed and their impact on BER performance was evaluated. The new approaches were focussed on reducing bitwidth of extrinsic information in WiMAX duo-binary turbo decoder as three extrinsic information LeA, LeB, LeC are exchanged across SISO decoders, thus giving us a bigger scope for reducing overall extrinsic memory size. In the proposed approaches instead of dealing with extrin-

sic information *LeA*, *LeB*, *LeC* individually, we try to perform optimization on the whole extrinsic vector. The idea is to sort LeA, LeB, LeC and then assign more number of bits to highest value and lesser bits for remaining. So if *EXT1*, *EXT2*, *EXT3* are the three extrinsic in the decreasing order of value, then for extrinsic vector of 16 bit:

Scheme 1:

- 3 Bits for : Sorting Information of *LeA*, *LeB*, *LeC* .
- 2 Bits for : Sign Information of *LeA*, *LeB*, *LeC*.
- 5 Bits for : *EXT1*
- 3 Bits for : *EXT2*
- 3 Bits for : *EXT3*

Scheme 2:

- 3 Bits for : Sorting Information of *LeA*, *LeB*, *LeC*.
- 2 Bits for : Sign Information of *LeA*, *LeB*, *LeC*.
- 5 Bits for : *EXT1*
- 3 Bits for : *EXT1-EXT2*
- 3 Bits for : *EXT2-EXT3*

The above mentioned schemes can be better explained using an example. Assuming *LeA* =-27, *LeB*=78, *LeC*=12, then after sorting, *EXT1*=78, *EXT2*=12 and *EXT3*=-27. All different possible sorting results can be calculated as 3! = 6, which could be depicted by 3-bit binary representation. Now to represent the sign information of these ordered three elements *EXT1*, *EXT2*, *EXT3* we actually need 2 binary bits, as there would be only 4 possible cases: "+,+,+", "+,-,-", "-,-,-" and "+,+,-". Our example corresponds to fourth case i.e. "+,+,-". Now we can see that 5 bits are used-up to store sorting and sign information of three extrinsics. Given a target bit-vector of 16 bits we have 11 remaining bits to actually represent the magnitude of these three extrinsic. In the first scheme we try to assign more bits for extrinsic with higher value, while for the second scheme different bits are assigned to highest extrinsic and to differential extrinsic values as shown above. The extrinsic vector is transmitted using above mentioned one of the two schemes, across SISO half iterations, and before starting the new half iterations *LeA*, *LeB* and *LeC* are reconstructed using *EXT1*, *EXT2* and *EXT3* as well as the sign and sorting information.

Simulation for these two schemes were performed using a entropy inspired distance (EID) metric for performance evaluation of iterative decoders presented in *chapter 6*, for quicker observations.

Table 3.3: EID Metric Simulation for WiMAX PCCC Turbo codes)

Sub-optimal Cases	EID Distance
1 LSB drop-append	936
2 LSB drop-append	1434
3 LSB drop-append	1938
Scheme1	2677
Scheme2	3341

Results obtained are summarized in Table 3.3 along with corresponding results of drop-append strategy mentioned in previous section for a comparative analysis (results correspond to a simulation set of N=10000 codewords at SNR of 1.47 dB). As can be seen from the results that proposed schemes seem to degrade the error performance of the decoder significantly (presented distance values correspond to more than 0.5 dB degradation in BER scale). One of the possible explaination to such behaviour could be derived from the extrinsic distribution shown in Figure 3.6 especially their variation across iterations. A more dynamic algorithm needs to be formulated taking into account the extrinsic spread across iterations.

3.6 Conclusion

A methodology for extrinsic message size reduction was proposed which results in bit-width reduction of 8 downto 4 bits for SCCC code with less than 0.1 dB performance loss at BER of 10^{-6}. For the WiMax CTC code bit-width reduction of 8 downto 5 bits is possible if loss of 0.2 dB at BER of 10^{-6} is tolerable [Singh et al., Sept 2008]. Cost, area and energy consumption of the turbo decoder implementation scales with the bit-width of extrinsic information. The presented results show that there exists a definite scope for memory size reduction by using suitable quantization algorithms for extrinsic information, without serious degradation of the bit-error performance.

Chapter 4

Network on Chip (NoC) Evaluation for Flexible Iterative Decoders

Contents

This Chapter explores Turbo and LDPC decoding application modelling over a NoC based Multi Processor SoC (MPSoC) platform. An overview of NoC paradigm is presented along with associated parameters. Then as a case study the performance evaluation of MPSoC architecture of a WiMAX (802.16e) based turbo and LDPC decoder for a 2-D Torus/Mesh interconnect topology is explored. Evaluation results are presented based on the communication centric parameters that include network latency, network size and can be extended to any other System on Chip (SoC) interconnect topology without loss of generality. The material of the second part has been partly presented in the ISSSTA symposium [Scarpellino et al., August 2008].

4.1 Background

In many telecommunication applications, it may be desirable to have a flexible system able to support a variety of error correction standards or a variety of codes, particularly in software defined radio (SDR) paradigm where flexibility is a fundamental property of future receivers. In recent years many research activities have emerged proposing multiprocessor implementations in order to achieve flexible and high throughput parallel iterative decoding. In order to obtain architectures that achieve both high throughput and flexibility, MPSoC is an effective solution. Together with flexible and high throughput processing elements, a MPSoC architecture must also include a flexible and high throughput interconnection network. The Network-on-Chip (NoC) approach has been proposed to interconnect processing elements in iterative decoder architectures designed to support multiple standards [Muller et al., Mar. 2006] [Brack et al., Sept. 2006].

4.2 MPSoC based Turbo Decoder architecture

A parallel turbo decoder can be modeled as P processing elements that need to read from and write to memories. Each processing element, often referred to as soft-in-soft-out (SISO) module, performs the BCJR algorithm [Bahl et al., Mar. 1974], whereas the memories are used for exchanging the extrinsic information among the SISOs. MPSoC based turbo decoder architectures are based on block level parallelism. Data blocks are divided into a certain number of windows that are processed sequentially. All the SISO decoders work in parallel on different windows of bits. Decoding of a block of data is done in iterative fashion. In the first half iteration estimates of decoded bits are calculated and send it over for further processing in the second half iteration. This sending over between iterations is called interleaving as data are sent in interleaved order. As the demand of high throughput is increasing from such kind of decoders, degree of parallelization is increasing, i.e. blocks are sub divided into sub blocks and each sub block then being processed by independent processing elements. With the increase in degree of parallelization a question of concurrent interleaving comes into the picture. Which is crucial given the improved performance of such decoder is mainly due to presence of interleaving aspect. In order to support arbitrary interleaving law, NoC seems to be an interesting solution [Boutillon et al., June 2007].

The upper half of the Figure 4.1 shows how the block is subdivided and processed independently by different processing elements (PEs). All the processing elements are same in their behavior i.e. they are running the same processing algorithm. As a reference a detailed MPSoC based parallel implementation of turbo decoder is presented in [Martina et al., April 2008], where in order to achieve high decoding throughput, a parallel architecture has been designed with 4 SISO decoder units.

As depicted in the lower half of the Figure 4.1 in the whole decoding flow following parameters

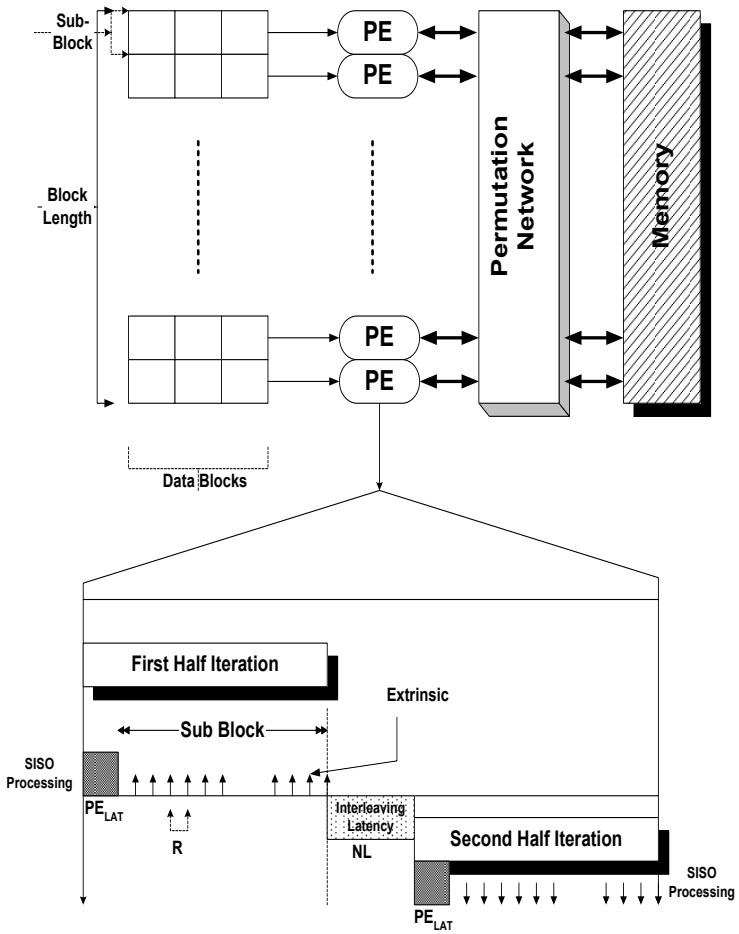

Figure 4.1: MPSoC Architecture for Turbo Decoder

can be classified:

- **Processing Start Time ($\mathbf{PE}_{LAT}$):** It is the time which processor takes to perform internal processing before starting to emit messages to permutation network (SISO Latency in the above example). It should be noted that for the duration of one half iteration time, occurrence of PE_{LAT} is just once.

- **Sending Period(R) :** It is the time interval at which each message (corresponding to each data of the sub block under consideration) is injected in the network. In the best case it is equal to the average number of cycles per calculated data on a PE. After the Time (PE_{LAT}), total number of messages sent across the network with a periodic interval of (R) is equal to the sub block length. It is worth noting that at each interval (R) packet destinations are different and are dependent on the interleaving law.

- **Network Latency (NL) :** It is the extra amount of time (Interleaving Latency in the above example) which a processing element has to wait so that all the expected messages are written on respective memory location before it can start the next cycle of processing.

4.2.1 Throughput Modelling

To quantify network traffic, we first estimate the data rates originating from interleaving traffic for turbo case with respect to a single processing unit. Given clock frequency (F), number of iterations (It) and processing unit latency (PE_{LAT}), the time employed by a processing unit to complete the sending of K processed messages can be estimated as:

$$T_{SISO} = \frac{It}{F}\left(K \times R + PE_{LAT}\right) \tag{4.1}$$

where R is the average number of cycles per calculated data on a node processor and K is the turbo block length. The simplest solution to implement decoder is to use one processing unit that performs in two half iterations. Given the single processing unit architecture we can estimate its throughput (D), as the number of decoded bits over the time employed to actually decode them ($2T_{SISO}$:

$$D = \frac{s \times K}{2T_{SISO}} = \frac{s \times K \times F}{2 \times It(K \times R + PE_{LAT})} \tag{4.2}$$

where s represent the size of the symbol and $s = 1$ for binary codes while $s = 2$ for double binary codes. In case of Parallism of P and with the assumption that the interconnection structure

(permutation network) latency is negligible T_{SISO} becomes:

$$T_{SISO} = \frac{It}{F}\left(\frac{K}{P} \times R + PE_{LAT}\right) \tag{4.3}$$

and the throughput (D) is given by:

$$D = \frac{s \times K \times F \times P}{2 \times It\,(K \times R + PE_{LAT} \times P)} \tag{4.4}$$

A similar treatment of MPSoC based LDPC decoder architectures is presented in the next section.

4.3 MPSoC based LDPC Decoder architecture

A LDPC decoder is defined by its parity check matrix H of M rows by N columns. Each column in H is associated with one bit of the codeword or Variable Node (VN), and each row corresponds to a parity check equation or Check Node (CN). LDPC codes are decoded in an iterative way by using the sum-product algorithm or belief propagation algorithm involving CN and VN updates [Gallager, Jan. 1962]. However this two phase decoding has recently given way to the so called layered or shuffled decoding [Hocevar, Oct. 2004], [Mansour and Shanbhag, Aug. 2002] which results in approximately two times faster convergence of the algorithm. The key point of which is the overlap of the traditional VN and CN phase, as a result of which each iteration can be viewed as the concatenation of several layered sub iterations. After the layered sub iteration reliability of the bit in the received codeword is immediately available to the next layer, which will work on this new information. However across each layered sub iterations CN update messages are shuffled through a network . In order to support arbitrary parity matrix NoC solutions are emerging as an interesting solution.

The Figure 4.2 shows how the parity check matrix is subdivided into different layers (supercodes) and certain number of check nodes (1 in the example) are processed independently by different processing elements (PEs) (4 in the example). All the processing elements are similar in their behavior i.e. they are running the same processing algorithm. Each data of the sub block after processing is send across the network. The important thing to note here is that mode of LDPC decoding application is constituted of different communication sub modes (equal to the number of super codes), as the messages are sent across the network at the boundary of two super codes and the pattern of such traffic is different at different boundaries and is governed by parity matrix.

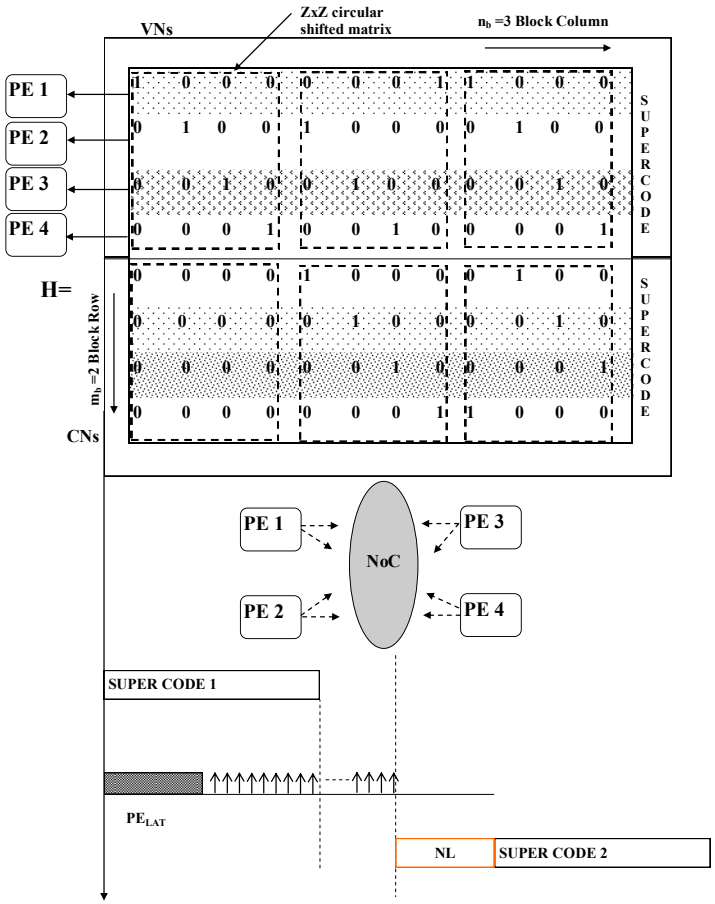

Figure 4.2: MPSoC Architecture LDPC data processing flow.

4.3.1 Throughput Modelling

As shown in Figure 4.2, the parity check matrix H is a class of architecture aware LDPC codes (AA-LDPC) and can be represented as a block matrix H_{BASE} (Base Model Matrix) of size $m_b \times n_b$ as described in *chapter 1*. Each element in H_{BASE} matrix is a $z \times z$ circular shifted identity matrix or zero matrix. The following parameters can be defined for such a matrix:

- Code Length: $N = n_b \times z$;
- Parity Check equations: $M = m_b \times z$;
- k: Check Node Degree.
- PE_{LAT}: Average number of cycles taken by each node processor to calculate the first output data (VN extrinsic value). In a check node centric LDPC decoder architectures this value is dependent on k of the code under consideration.

In presence of P processing elements, the number of c_n executed in each processor is equal to z/P. With the assumption that the interconnection structure latency is negligible, for each check row processing, given clock frequency (F), and processing unit latency (PE_{LAT}), the time employed by the processing unit to obtain $(z/P) \times k$ messages can be estimated as:

$$T_{DECODING} = \frac{It \times m_b \times PE_{LAT}}{F} \tag{4.5}$$

Throughput (D) can be estimated as the number of decoded bits ($N - M$) over the time employed to actually decode them ($T_{DECODING}$):

$$D = \frac{N - M}{T_{DECODING}} = \frac{(N - M) \times F}{It \times m_b \times (PE_{LAT})} \tag{4.6}$$

4.4 Network on Chip (NoC)

Network-on-Chip (NoC) is an emerging paradigm for communications within large VLSI systems implemented on a single silicon chip. In a NoC system, modules, such as processor cores, memories and specialized IP blocks exchange data using a network as a "public transportation" sub-system for the information traffic. A NoC is constructed from multiple point-to-point data links interconnected by switches, such that messages can be relayed from any source module to any destination module over several links, by making routing decisions at the switches. A NoC is similar to a modern

telecommunications network, using digital bit-packet switching over multiplexed links. The wires in the links of the NoC are shared by many signals. A high level of parallelism is achieved, because all links in the NoC can operate simultaneously on different data packets. Therefore, as the complexity of integrated systems keeps growing, a NoC provides enhanced performance (such as throughput) and scalability in comparison with previous communication architectures (e.g., dedicated point-to-point signal wires, shared buses, or segmented buses with bridges). Of course, the algorithms must be designed in such a way that they offer large parallelism and can hence utilize the potential of NoC. The adoption of NoC architecture is driven by several forces: from a physical design viewpoint, in nanometer CMOS technology, interconnects dominate both performance and dynamic power dissipation, as signal propagation in wires across the chip requires multiple clock cycles. NoC links can reduce the complexity of designing wires for predictable speed, power, noise, reliability, etc., thanks to their regular, well controlled structure. From a system design viewpoint, with the advent of multi-core processor systems, a network is a natural architectural choice. A NoC can provide separation between computation and communication, supports modularity and IP reuse via standard interfaces, handles synchronization issues, serves as a platform for system test and hence increases engineering productivity.

4.4.1 Classification of Interconnection Network

Based primarly on network topology, interconnection networks are classified into four major classes: Shared-medium networks, direct networks, indirect networks and hybrid networks [Jantsch and Tenhunen, 2003]. Figure 4.3 reflects this classification along with some examples of network beloging to respective categories.

In shared-medium networks , the transmission medium is shared by all communicating devices. It is the least complex interconnect structure of all, only one device is allowed to use the network at a time. A suitable arbitration mechanism is needed to determine the mastership of the shared-medium network. They are simple to implement however limited bandwidth and non scalability are a botteleneck in high throughput application.

Direct Networks or point to point networks are a highly scalable network architecture. They consists of a set of nodes, each one being directly connected to small subset of other nodes in the network. A common component of these nodes is a router which manages message flow across the nodes that is why these are also known as router based networks. These networks are characterized by topology, routing and switching. Topology defines how the nodes are interconnected. For topologies in which packets may have to traverse some intermediate nodes, a routing algorithm decides the path selected by the message to reach its destination. The switching mechanism determines how the network router resources are allocated for message transmission. All these elements determine the network performance however they are not independent to each other.

Indirect networks are another class of interconnection networks also known as switch based net-

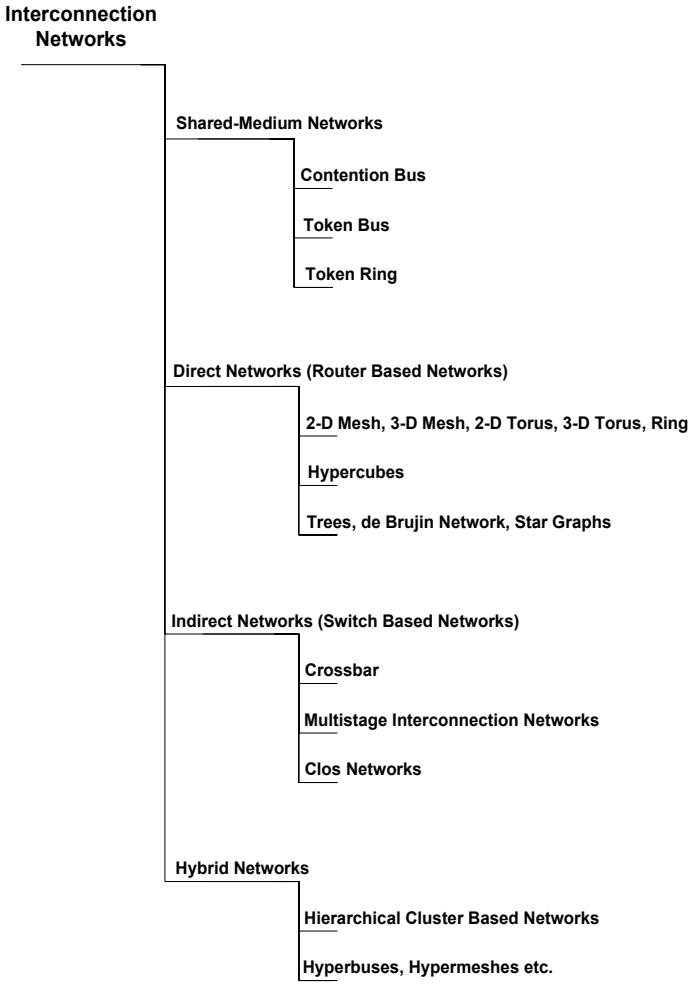

Figure 4.3: Classification of Interconnection Network

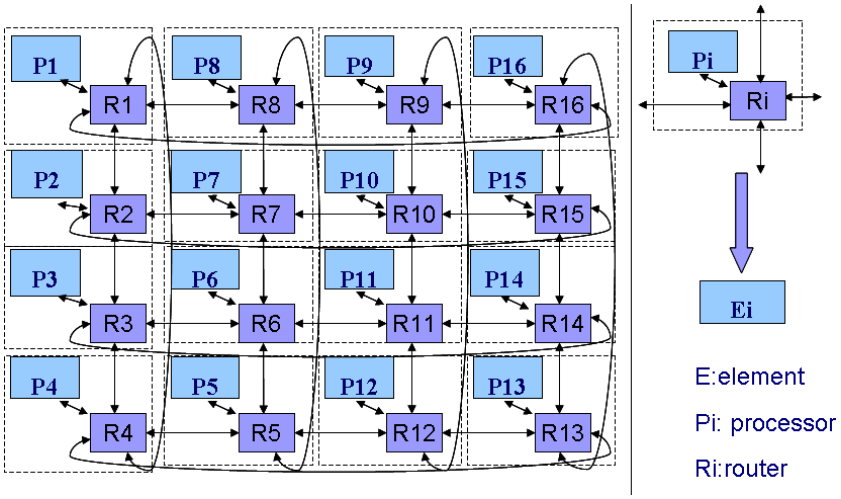

Figure 4.4: Structure of interconnection with 16 processing element and routing elements.

work as instead of providing a direct connection between two nodes, the communication between two nodes has to be fecilitated by a switch. Similar to direct networks, an indirect network is also defined by topology, routing and switching, The topology defines how the switches are interconnected.

Hybrid networks combine mechanism from shared medium networks and direct or indirect networks. The result is a higher bandwidth with respect to a shared medium network and smaller distance between nodes with respect to direct and indirect networks. However for very high performance, hybrid networks are limited by scalability.

4.4.2 2-D torus Network

In this section we will talk in more detail about 2-D torus network which is a direct interconnection type network, as we will use this topology to evaluate impact of NoC on iterative decoders performance, largly due to its highly regular structure. 2-D torus topology is an extension of the 2-D Mesh topology where each node of the network is connected to four other nodes, in the east, west, north and south directions. In figure 4.4 the structure of interconnection with 16 processing elements is depicted.

In our experiment, each router implements $O1$TURN routing algorithm [Seo et al., June 2005], it is selected because of its low design complexity and good average-case through-put. The idea of this routing algorithm is quite simple: in the whole network, each packet is allowed to turn only once. Therefore, if we consider a 2-D Mesh topology, there are only two ways for each packet to reach its destination, one taking first the X axis of the network and then taking the Y axis (X-Y way) or the second way which takes first the Y axis and then the X axis (Y-X way) (figure 4.5), though both ways are equivalent in distance. This policy can be extended to a 2-D torus topology. Theoritically, four ways are possible with this topology, with two (X-Y, Y-X) pairs. Yet we can simplify this by taking only the (X-Y, Y-X) pair which is smaller in distance.

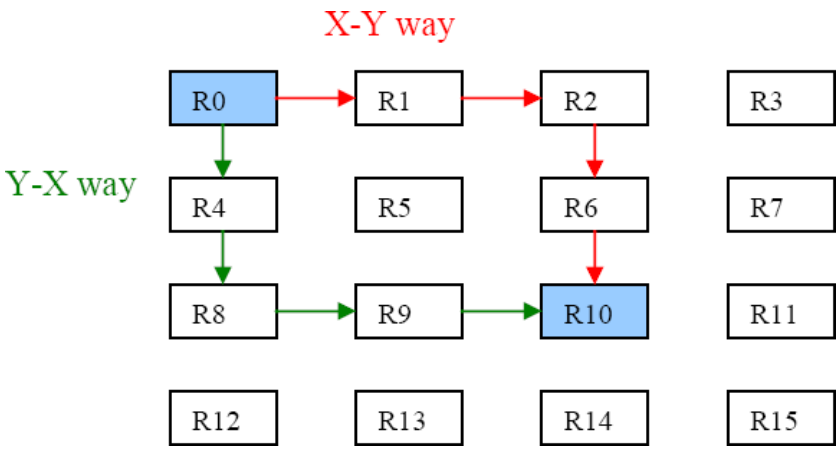

Figure 4.5: XY and YX ways for the 2-D Mesh topology

For better understanding of the $O1$TURN routing algorithm (useful for simulation results presented in later sections), first we work with directions (east, west, north and south) instead of using virtual channels, and then we choose for each (source-destination) pair only one (X-Y, Y-X) pair in the torus topology. When data is sent by a processor into the network the X-Y mode or the Y-X mode is selected randomly. Let us consider an example with the node number 5, when it receives a packet whose X-Y mode is enabled:

- If the destination address is 0, 4, 8 or 12, node 5 will output this packet through its eastern port.
- If the destination address is 2, 3, 6, 7, 10, 11, 14 or 15, node 5 will output this packet through its western port.
- If the destination address is 1, node 5 will output this packet through its northern port.
- If the destination address is 9 or 13, node 5 will output this packet through its southern port.
- If the destination address is 5, the packet will be sent to the memory associated with the node.

These choices are represented by the left part of figure 4.6, while the case of Y-X mode is described by the right half of the diagram.
The resultant routing table for node 5 is presented in table 4.1. The routing table of other nodes can be derived in the similar manner.

4.5 Case Study using 2D-Torus/Mesh NoC

In [Scarpellino et al., August 2008], a processing element capable of performing Turbo and LDPC decoding functionality for WiMAX codes is presented. To evaluate the impact of NoC on MPSoC architectures based iterative decoders, we use such PEs as the building blocks and evaluate the

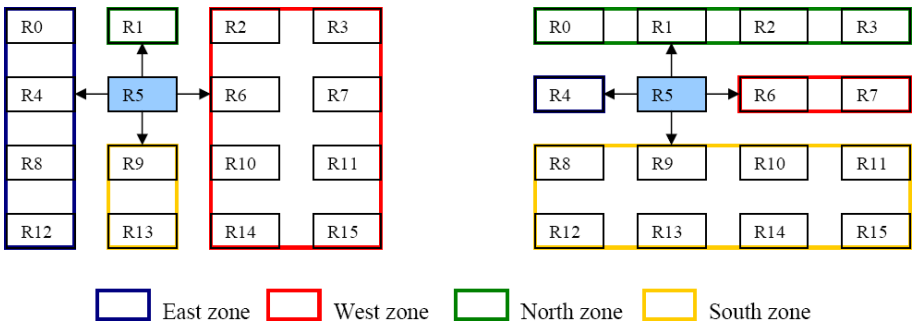

Figure 4.6: Elaboration of the routing tables : Determination of X-Y and Y-X ways.

Table 4.1: Node 5 routing tables.

Destination	X-Y way	Y-X way
0	W	N
1	N	N
2	E	N
3	E	N
4	W	W
5	–	–
6	E	E
7	E	E
8	W	S
9	S	S
10	E	S
11	E	S
12	W	S
13	S	S
14	E	S
15	E	S

behaviour of NoC for its parameters. We believe that such an analysis can be a basis for SoC designer for choosing a network on chip topology for high performance decoders. In our parallel architecture NoC is the medium to carry out exchange of extrinsic-information values and extrinsic VN values between processors. They are important because of their capability to sustain any type of permutation law. However such a flexibility comes with costs, primarly the overhead involved in message delivering through the network and secondly each element connected to the network needs interface. Next section describes some of the characteristics of processing unit architecture to provide clarity on the system model used for our experiments.

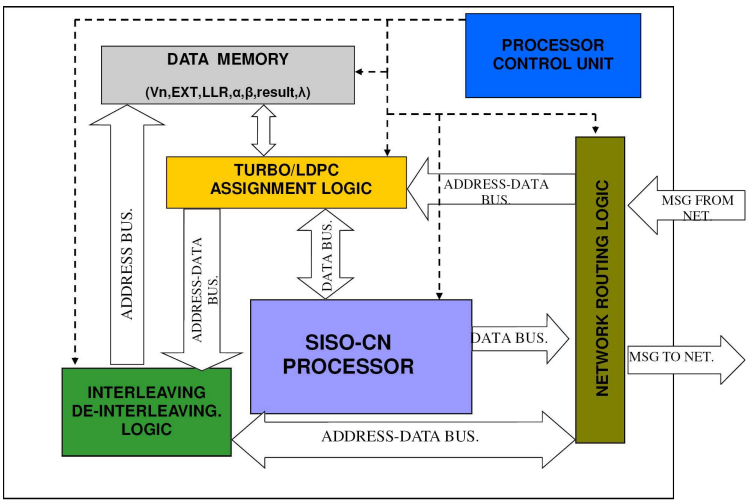

Figure 4.7: Reconfigurable Processing Unit Architecture.

4.5.1 Processing Unit Architectural Overview

Figure 4.7 shows the overall architecture of the reconfigurable processing element. It consists of the SISO-CN Processor, data memories, TURBO/LDPC assignment logic unit, processor control unit and an interleaving and deinterleaving logic connected to a network routing logic. Data is exchanged through the packet based network. Before designing the processing element supporting both turbo and LDPC decoding, general design choices had to be made. In the case of LDPC decoding the parity-check matrix H is seen as a concatenation of c parity-check matrices $H^1,\ldots,H^c$, corresponding to supercodes [Mansour and Shanbhag, Aug. 2002]. This in turn enables us to decompose the interconnection network defined by the overall matrix into smaller networks or interleavers that connect together adjacent supercodes. This step transforms the LDPC decoding problem into a turbo decoding problem, with the supercodes acting as constituent codes, thus facilitating the reuse of a hardware platform meant for turbo decoding in the LDPC case.

In the turbo case, corresponding to each block size K, certain number of processing elements (maximum 16 in our implementation) for decoding are activated. Each block is divided into a number of windows I and the size of each windows is W. Accordingly, the same number of windows are processed by each processing element [Blankenship et al., June 2005a]. Before processing, the configuration of all sub-blocks is done by the processor control unit (PCU). PCU asserts signals according to block size, first or second half iteration, number assigned to each processor, in addition supercode number for LDPC case, thus taking care of different decoding scenarios to function correctly.

Data memories consists of memory blocks for the storage of extrinsic/VN values, channel llr, α-β values, λ values and hard decoding results in case of turbo decoding. The design choices previously explained make sure sharing of the maximum amount of memories between the LDPC and turbo

decoding; this comes at the cost of an increased complexity of interfaces between the processing core (SISO-CN Processor) and the memory blocks.

Turbo/LDPC assignment logic unit is responsible for the selection of inputs based on the selected decoding. In addition it performs reordering of variable node values for the LDPC case before assigning it to the SISO-CN processor. Reordering of VN is necessitated by the fact that VN values coming from the network are not in the proper sequence.

SISO-CN processor architecture is typical of a turbo decoding SISO as explained in [Martina et al., April 2008] and the blocks are reused for the LDPC decoding case. It computes and updates the extrinsic information for each subiteration in turbo case. Suitable modifications were made to incorporate the LDPC decoding case, as trellis has a different structure than the turbo decoding case [Mansour and Shanbhag, Aug. 2002]. In LDPC case it computes and updates the intrinsic information.

Interleaving and deinterleaving logic generates destination addresses for the extrinsic information for turbo case according to selected half iteration and permutation law. While for LDPC case, it generates addresses to read from the data memories, variable node values involved in the supercode under consideration. The destination addresses are sent with the data to network routing logic, which interfaces processing element with the network. This unit according to the destination address is able to understand if the data should be sent through the network to another processor or should remain in the same processor based on the permutation law or the check node allocation map for turbo and LDPC case respectively.

4.5.2 Impact of NoC on Decoder performance

Assuming that an iterative decoder is being implemented as a parallel architecture using the processing elements mentioned in previous sections and the interleaving operation (turbo case) or CN update (LDPC case) message shuffling is mapped to a network on chip. The problem of performance evaluation can be modeled as efficient deployment of a real application over ad/hoc NoC. Now looking at the Figure 4.1 and Figure 4.2 it is evident that to have a higher decoding throughput the expression $\mathrm{PE}_{LAT} + NL$ should be minimized. PE_{LAT} is dependent on the processing hardware, thus for a given R the NoC optimization goal would be in the direction of reducing the NL for achieving the higher throughput.

4.5.2.1 System Simulation using Network on Chip Simulator

In order to evaluate the traffic pattern of the iterative decoder over a NoC we used NoC Simulator, NNSE [LU et al., April 2005]. NNSE allows the user to analyze the performance impact of NoC configuration parameters. It allows not only to configure a network with respect to topology, flow control and routing algorithm etc., also to configure various regular and application specific traffic

patterns. The network can be evaluated with the traffic patterns in terms of latency and throughput. The main component of this tool are as follows:

- **Simulation kernel**: The tool logically consists of a NoC simulation kernel wrapped with a graphical user interface (GUI). The kernel provides a layered network simulation engine. Following ISO's OSI model, the kernel implements five layers, namely, the physical layer, the data link layer, the network layer, the transport layer and the application layer. The transport layer provides transaction-level communication primitives such as read() and write() to enable communication via channels between application processes. Each layer may be configured with a set of parameters that represents its characteristics.

- **Network configuration**: A network is characterized by topology, flow control scheme and routing algorithm etc. Each of them has a large design space on its own. With the tool, all of these characteristics are parameterized. The tool at present realizes only a limited set of configurations, like 2D mesh and 2D torus topologies and wormhole routing and deflection routing algorithms.

- **Traffic configuration**: One important aspect in NoC design is that the chosen network should be customized for applications, i.e. application-specific. In addition to customize the network parameters, the network should be evaluated extensively with various regular and application-specific traffic patterns since the network selection is an architectural decision and thus should be relatively stable. The regular traffic patterns consist of uniform and locality traffic. The uniform traffic is distributed over the network nodes uniformly. With locality traffic, one can specify the locality index which controls the communication probability between nodes at different distances. The application-specific traffic is based on per channel and can be used to configure application-specific, irregular traffic. One can specify explicitly the communication characteristics of each channel. The former is dynamic while the latter is static. With both types of traffic patterns, one can specify message temporal behavior and message sizes.

- **Performance evaluation**: After configuring a network and a traffic pattern, one can evaluate the network with the traffic pattern. The evaluation is based on the kernel simulation results. The main performance measures are latency and throughput. Typical figures include average latency, throughput etc.

For our experiment patterns of the messages over network for both turbo and LDPC decoding were obtained using MATLAB software. The traffic generator of the simulator was adapted to provide the same traffic pattern obtained from MATLAB software and simulation was performed over the $2D$ Torus/Mesh infrastructure provided by the simulator. In the simulations all packets are assumed to have a constant length. All resource contention is handled without bias, such that

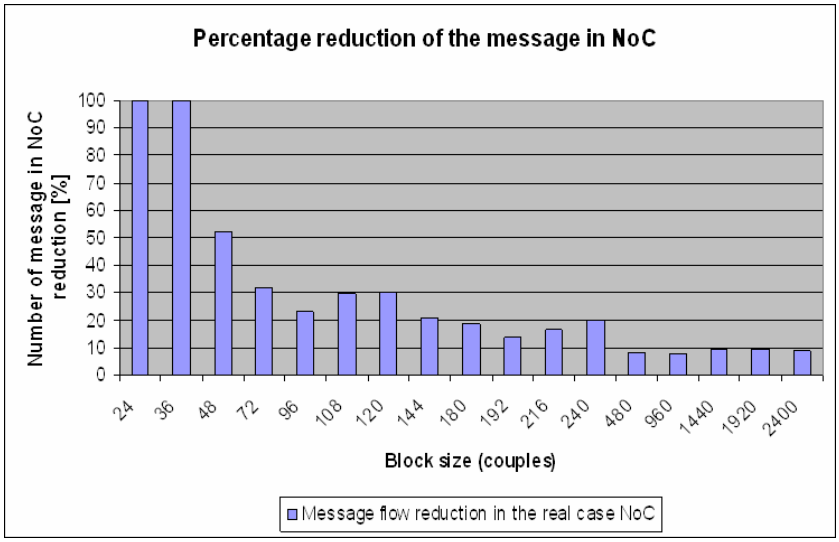

Figure 4.8: Percentage reduction of the number of sent messages in the realistic case with respect the two approximations.

granting of resources to packets is done on a first come, first-serve basis. The tool was modified to provide X-Y routing for the Torus/Mesh topologies.
The tool proposes various traffic pattern options, for example *periodic sending* where all generators send data into the network at the same time periodically or *random sending* where they all send data randomly. We choose the first traffic pattern as it is closer to the functioning of a iterative decoder implemented over multiprocessor architecture. Since the tool is designed for more heterogenic general purpose NoC application it presents some limitations for simulating a iterative decoding traffic pattern, e.g. the simulation scenarios where one processor may want to write in its own associated memory is not taken into account. This has an impact on the results because in that case real number of message sent over the network are less than the entire block size. The percentage reduction of the packets sent over the network in one half iteration are depicted in figure 4.8, as can be seen that for block size 24 and 36 there are no message in the network as number of PE is one while for higher block length number of PEs varies from 2 to 16.

We resolved this issue by using two step approximations:

- First step: when a processor has to write in his own memory, it sends the packet towards a neighbor processor.
- Second step: when a processor has to write data in its own memory, it sends packet towards a random destination.

The first case is less realistic, as it results in congestion in the network between two neighbors PEs. If the traffic pattern is localised, they should exchange many messages and this would lead to an unjustified increase of latency values obtained. In the second approximation this does not happen

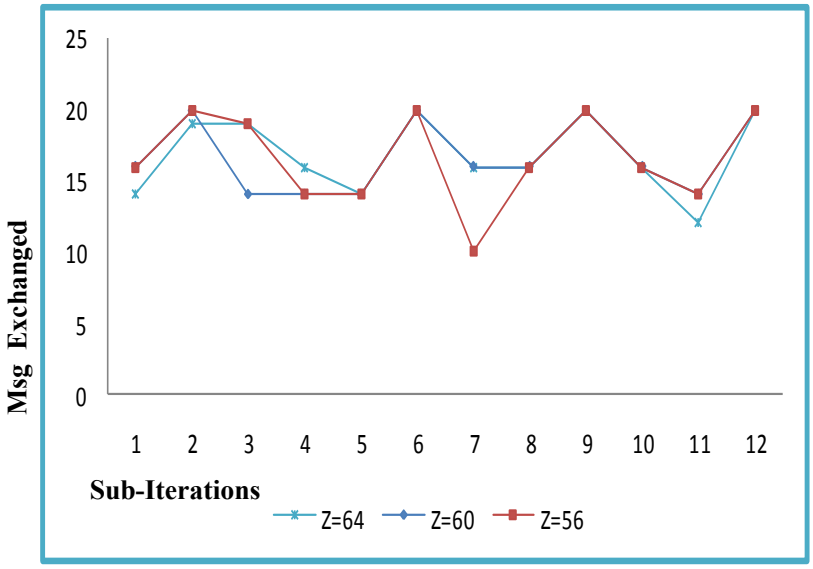

Figure 4.9: Number of messages exchanged on the network during LDPC decoding.

therefore, this approximation represents a more realistic scenario.

One of the important thing to mention here is that in case of LDPC decoding initial mapping of the VN values on the processor memories has great impact on the traffic patterns obtained. During the Matlab^{TM} simulations, we have observed that an average 65 percent of the messages are exchanged implicating a high amount of traffic on the underlying mesh network. Figure 4.9 shows the number of messages exchanged between processors before the *SISO-CN PROCESSOR* processing for a given super-code, from the values thus observed it is recommended to perform a further exploration in the direction of optimal mapping methods.

4.5.2.2 Network latency

Each packet sent over the network suffers latency. We can observe that, reducing the sending period below a certain threshold, the network latency increases. In the simulations, for the turbo decoder case the first half iteration (*interleaving law*) has been considered. For each packet sent over the network, two possible latencies values are recorded during the experiment:

- NL_{MAX}: Maximum Network latency suffered by an information packet.
- NL_{AVG}: Average Network latency suffered by an information packet.

Figure 4.10 shows the maximum latency (NL_{MAX}) values of the network obtained during the simulations, for the traffic of each half iteration by varying *sending period*(R) in case $K = 2400$ and $K = 120$,using the first step approximation proposed previously. Instead in figure 4.11 the similar results for $K = 2400$ is presented however with the second step approximation. As can be seen

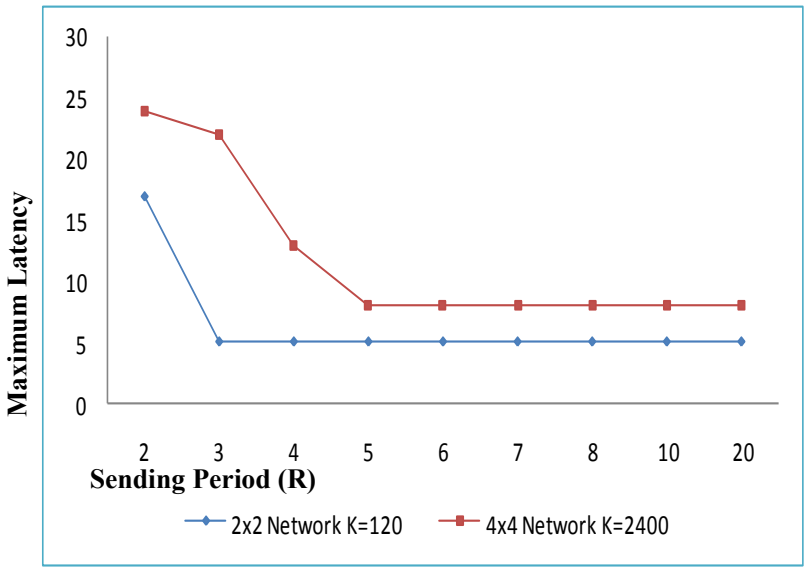

Figure 4.10: NL_{MAX} vs sending period in the first step approximation.

from the figures, for the same *sending period* and the same K there is a reduction in the maximum latency values. Reducing the *sending period*, the latency on the network increases, similar behaviour is observed in the simulations for the LDPC decoder case. Figure 4.12 shows such a variation for one of the sub-iteration during LDPC decoding with Z=64. This implies that no matter how efficient is our PE to decode the data at faster rate (thus lesser R), available network bandwidth would be a limitation on obtainable maximum performance.

Figures 4.13 and 4.14 shows maximum and average latency values of the network in case of LDPC decoder, respectively obtained during the simulations for the traffic of each sub-iteration using Network on Chip simulator. Traffic patterns are different for each sub-iteration and are obtained through Matlab^{TM} simulations of the PE architeture mentioned previously.

4.5.2.3 Throughput

Here it would be interesting to see the impact of NL on throughput equations 4.4 and 4.6. Impact of network latency on overall decoder throughput for different message sending period can be best described by a timing diagram model of message exchange between two processing elements in MPSoC architecture. In Figure 4.15 such a model for two possible scenarios is depicted. First scenario corresponds to sending period (R) greater than or equal to the network latency (NL), for such a scenario time to exchange k messages across network would be $k \times R + NL$, while in other case where R is less than NL time to exchange k messages becomes $k \times NL + R$. Considering that, T_{SISO} in turbo decoding case becomes :

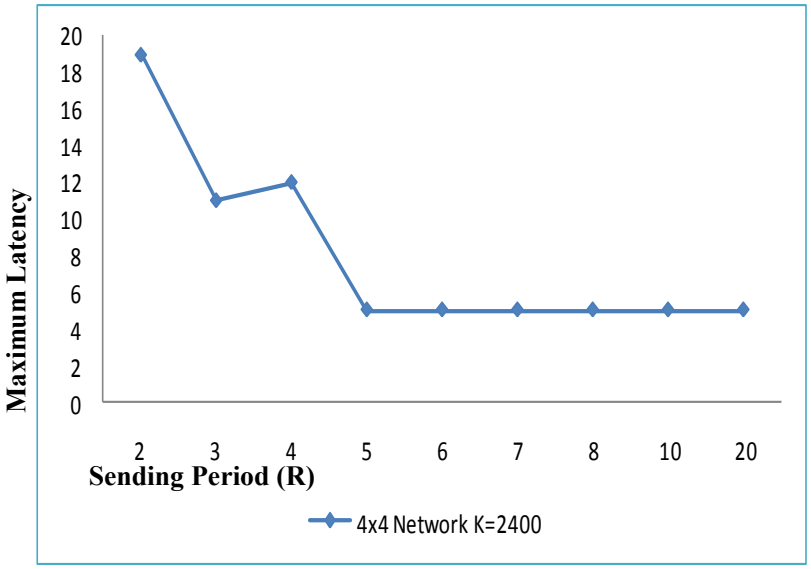

Figure 4.11: NL_{MAX} vs sending period in the second step approximation.

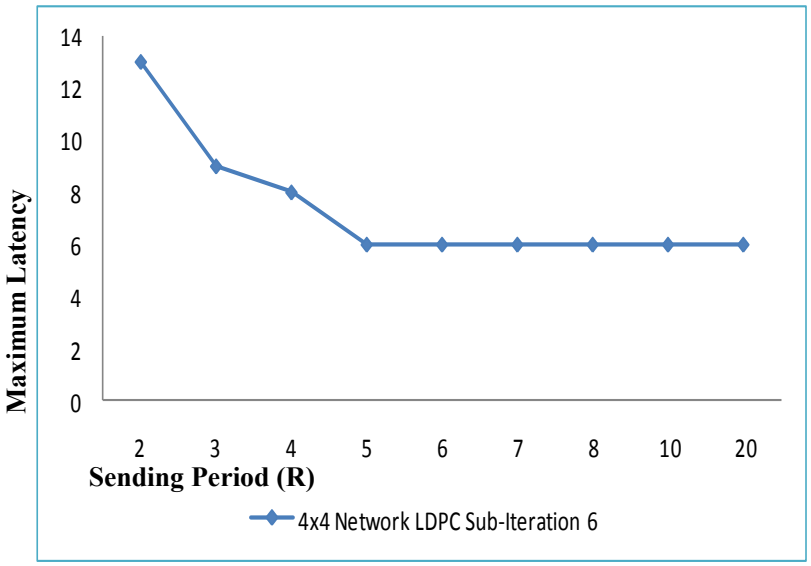

Figure 4.12: NL_{MAX} vs sending period in LDPC decoder at Sub-Iteration 6.

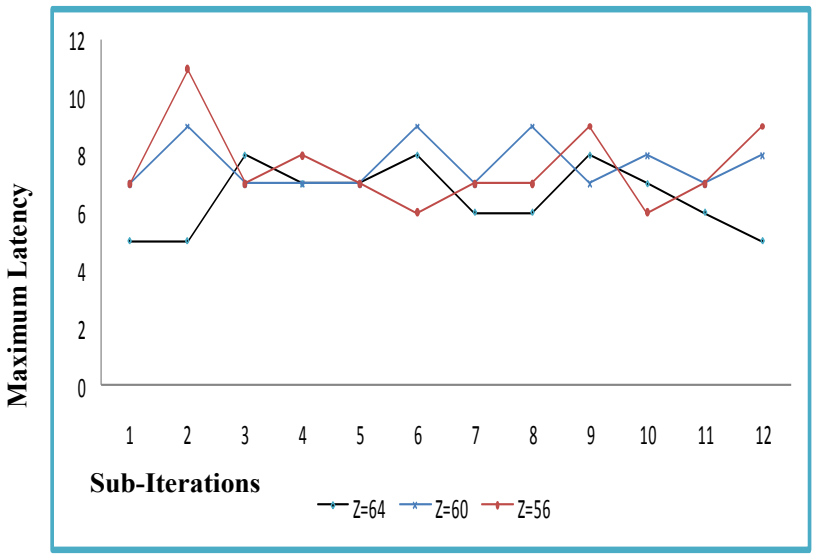

Figure 4.13: Maximum latency values during each sub-iteration in the LDPC decoding.

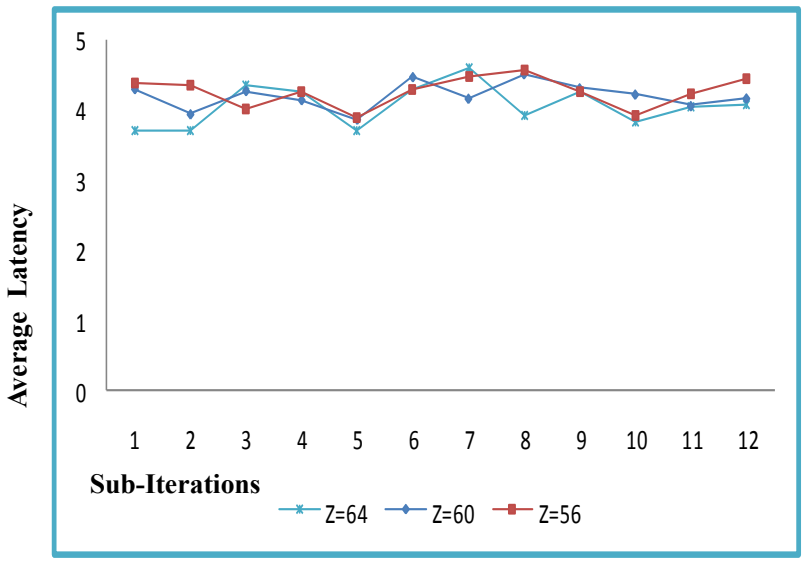

Figure 4.14: Averege latency values during each sub-iteration in the LDPC decoding.

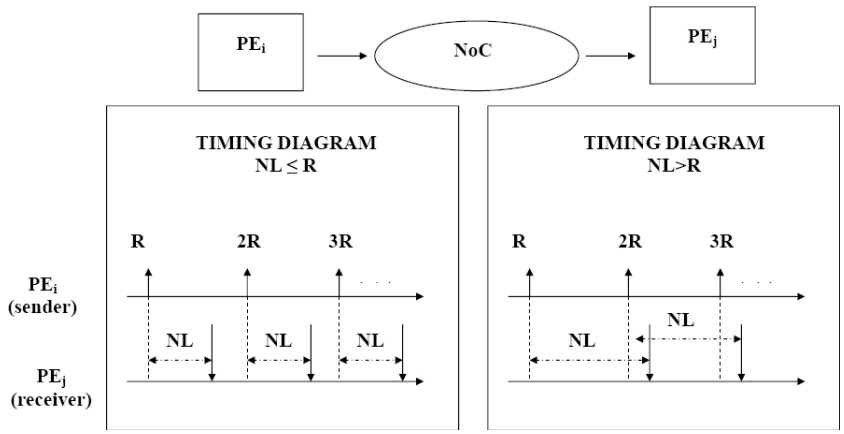

Figure 4.15: Timing Diagram Model for message exchange across network.

$$T_{SISO} = \frac{It}{F}\left(\frac{K}{P} \times \max(R, NL) + PE_{LAT} + \min(R, NL)\right) \tag{4.7}$$

while the corresponding average throughput(based on average latency suffered by packets) and worstcase throughput(based on maximum latency suffered by a packet)are given as:

$$D_{AVG} = \frac{s \times K \times F \times P}{2 \times It\,(K \times \max(R, NL_{AVG}) + PE_{LAT} \times P + \min(R, NL_{AVG}))} \tag{4.8}$$

$$D_{WORST} = \frac{s \times K \times F \times P}{2 \times It\,(K \times \max(R, NL_{MAX}) + PE_{LAT} \times P + \min(R, NL_{MAX}))} \tag{4.9}$$

Based on these formulas, throughput values for different number of PEs and different R have been calculated (network has a strong impact in the total achievable throughput). In Figure 4.16 throughput results are presented for WiMax turbo code of block length 2400 with the network, considering F=200 MHz, number of iterations equal to 8 for different sending periods. D represents the throughput in case of negligible network latency, D_{WORST} and D_{AVG} are the obtainable throughput in the presence of a Torus/Mesh network. We can observe that the accepted traffic by the network depends on the rate at which the each SISO element is injecting data into the network. Ideally accepted traffic should increase linearly with this injection load (lesser sending period) as in the scenario of without network case D. However, due to the limitation of routing resources (switches and interconnect wires), it saturates below a certain value of R resulting in a reduced decoding throughput.

In LDPC case considering N_{MSG} being the number of messages received by each node processor

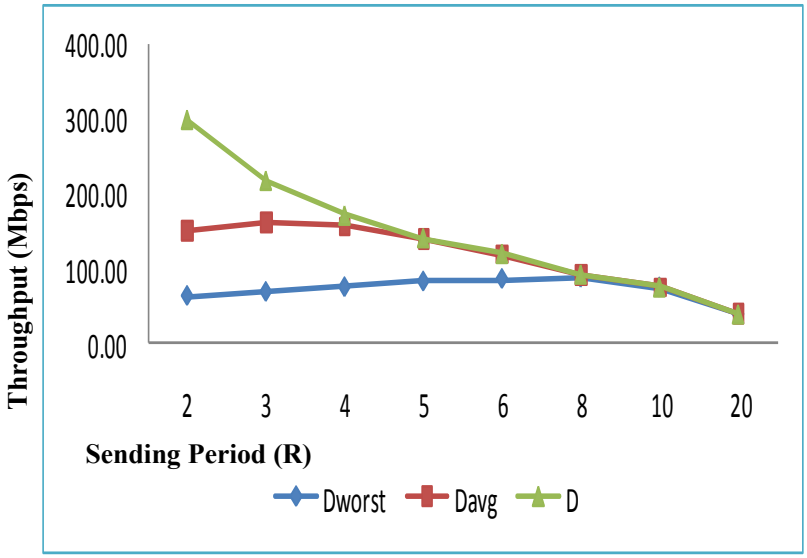

Figure 4.16: Throughput variation with different sending periods.

during a sub-iteration from network, $T_{DECODING}$ becomes:

$$T_{DECODING} = \frac{1}{F}\left(PE_{LAT} + N_{MSG} \times \max(R_{NET}, NL) + \min(R_{NET}, NL)\right) \tag{4.10}$$

Modified throughput expression becomes:

$$D_{AVG} = \frac{(M-N) \times F}{It \times m_b \left(PE_{LAT} + N_{MSG} \times \max(R_{NET}, NL_{AVG}) + \min(R_{NET}, NL_{AVG})\right)} \tag{4.11}$$

$$D_{WORST} = \frac{(M-N) \times F}{It \times m_b \left(PE_{LAT} + N_{MSG} \times \max(R_{NET}, NL_{MAX}) + \min(R_{NET}, NL_{MAX})\right)} \tag{4.12}$$

In the Table 4.2 throughput results are presented for turbo and LDPC case with the network, considering F=200 MHz, number of iterations equal to 8 (10 in LDPC case) and sending period=4 for two block sizes K (three block sizes for rate 1/2 in LDPC case). Throughput values obtained are lower for LDPC case than turbo case as the CN update process is more complex in LDPC decoding than extrinsic calculation in turbo. As a result the PE latency (PE_{LAT}) in LDPC case, associated to each variable node that must be processed is high.

Table 4.2: Throughput summary table in case of Turbo and LDPC.

Block Size	Turbo/LDPC	D [Mbit/sec]	D_{MAX} [Mbit/sec]	D_{AVG} [Mbit/sec]
120	*Turbo*	45.4	16	19.1
2400	*Turbo*	322.6	39	86.5
1344	*LDPC*	32.9	6.76	10, 27
1440	*LDPC*	35.3	7.19	11.06
1536	*LDPC*	37.6	8.63	11.2

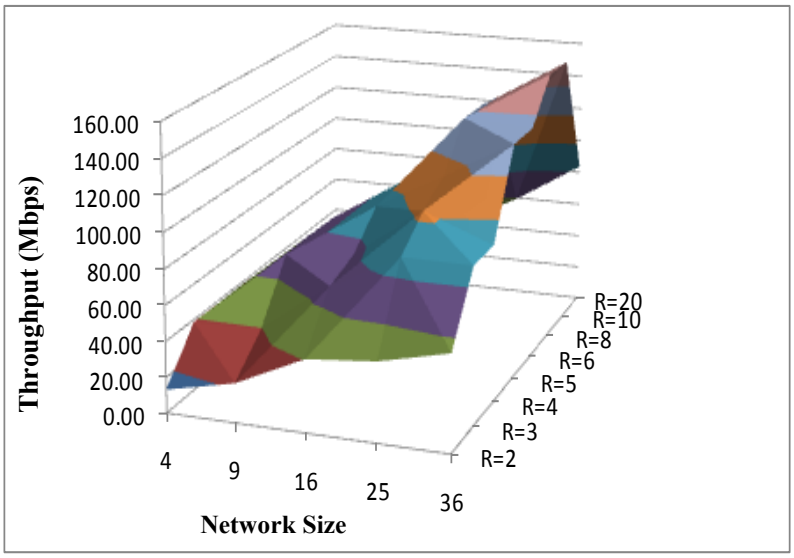

Figure 4.17: Throughput versus Network Size versus Injection Load

4.5.2.4 Network Size

From expressions 4.8 and 4.11 shown in previous section we can observe that increasing the number of processing elements increases the obtainable decoding throughput. However such an increase is not linear. With the network simulations it was observed that the network latency increases with increase in the network size (increase in P). In the Figure 4.17 impact of increase in processing elements at different injection rates on decoder throughput is depicted in three dimensional space. It can be inferred that there exists a optimal choice in terms of number of processing units required capable of generating data at a given rate to achieve a certain decoding throughput. The plot is drawn based on maximum latency offered by the network for turbo code of block length 2400, similar reasoning can be extended for the LDPC codes.

4.6 Conclusion

In this chapter, we have detailed the MPSoC architecture for two kinds of iterative decoder viz. Turbo and LDPC. For high-throughput decoding, we used multi-processor platform (MPSoC) as a natural transition to parallel decoding. As the inter-processor communication becomes the bottleneck for high degrees of parallelization, we presented a case study that analyzes the traffic pattern of these iterative decoding algorithms over a 2-D Torus/Mesh Network on Chip. By a detailed traffic analysis we showed that the overall processing performance is impacted by the networks limited communication bandwidth.

Chapter 5

Multistandard FEC Kernel

Contents

In this chapter, we explore datapath reuse possibilities across some important FEC families like convolutional, turbo and low density parity check (LDPC) codes. At first, design of a reduced complexity trellis network for shuffling the updated state metrics in Viterbi and Turbo decoding kernel is presented. Then in the following sections hardware reuse potential of different existing implementation schemes for check node processing in LDPC decoding is explored over such a FEC kernel architecture. In the last part, we propose a novel parallel implementation approach("Tree-Way") for check node processing, which fits very well with underline FEC Kernel architecture. Implementation results are presented showing that the proposed scheme, provides significant speed-up in terms of required clock cycles without significant increase in combined datapath area compared to existing approaches.

5.1 Introduction

Sharing of datapath and memories across different forward error correction (FEC) decoder implementations are important in flexible wireless communication system design. In recent years many research activities have emerged proposing multistandard ASIPs implementations in order to achieve flexible and high throughput decoder architectures for these codes. In [Alles et al., Sept. 2008] a memory sharing across turbo and LDPC codes in an application specific processor (ASIP) was explored while datapth reuse gains were considered insignificant. In [Naessens et al., Oct. 2008] authors proposed a reconfigurable ASIP supporting different codes across FEC families like convolutional, turbo and LDPC codes where datapath reuse across different FEC families has been used however at a macroscopic level of decoding structure.

As mentioned previously in *chapter 2*, in the present literatures, a comprehensive analysis of datapath reuse providing the designer with the choice to decide on the different possible ways of implementing basic FEC computation units is still lacking. This chapter is an effort in analyzing possible hardware reuse scenarios across different algorithms. The analysis explores the VLSI implementation complexity of this joint datapath supporting different block length, code rates, constraint length and polynomials proposed in emerging wireless standards like WiMAX 802.16e, 3GPP LTE, DVB-S2/T2 etc.

5.2 Multistandard FEC System

As already discussed before, wireless standards incorporate decoders from one or more of the four following code families: Reed-Solomon (RS), convolutional, Turbo, and LDPC. Some preliminary observations on resource sharing for combined decoder designs results from the analysis of the datapath structure and complexity of the corresponding decoder. For example, RS requires very different operation compared to other code families, so it is very difficult to find any sensible combination incorporating hardware resource sharing for this code family. On the other hand, convolutional, Turbo and LDPC decoding involve processing segment containing a parallel processing array performing similar basic computations as detailed in later sections, thus it is interesting to explore datapath reuse possibilities acorss these algorithms. Figure 5.1 shows the block diagram of a generic FEC decoder; the decoding algorithm is partitioned into two segments: control and the processing segment. Even though the figure represents a serial decoder implementation, in our work parallel implementations are addressed as described in *chapter 3*. The need to incorporate viterbi decoder, turbo decoder (single and duo-binary), and LDPC decoder inside a single hardware demands a judicious sharing of logic and memory units. Sharing of memory has been explored extensively in [Alles et al., Sept. 2008] and [Naessens et al., Oct. 2008] where it was established that significant reuse of memory resource is possible across these algorithms, however in this work our focus is on

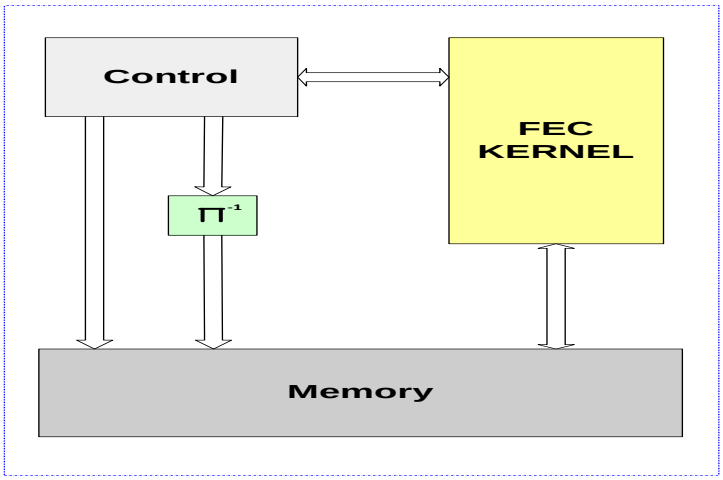

Figure 5.1: Multistandard FEC Decoder.

the datapath logic reuse.

5.3 A FEC Kernel for Turbo and Viterbi decoding

By analyzing the concepts and the architectures of the convolutional and the turbo code decoders, some circuits sharing techniques are applied to merge the main functions into one decoder. As explained in previous chapters in turbo decoding the processing steps are the basic Soft Input Soft Output (SISO) decoders which implement the BCJR algorithm. Decoding of the binary and duo-binary turbo codes is performed using either Log-Map or the Max-Log-Map version of BCJR algorithm. In the Log-MAP algorithm, the function used to compute the forward (A_k), backward (B_k) and log-likelihood ratio (LLR) output metric is given as:

$$f(a,b) = max(a,b) + \ln(1 + e^{-|a-b|}). \tag{5.1}$$

On the other hand Max-Log-MAP algorithm is a simpler sub-optimal version of Log-MAP and could be obtained simply by discarding the correction factor in equation 5.1. Untill now some approaches combining the Turbo and Viterbi decoding have been reported [Huang et al., May 2004], [Kreiselmaier et al., Feb 2004]. The fundamental fact utilised in these work about the datapath sharing between Turbo and Viterbi decoding is that both hard-output Viterbi and Max-Log-Map algorithms work with add-compare-select (ACS) operations. Fig. 5.2 depicts the architecture of a multistandard turbo kernel which is quite similar to the one proposed in [Muller et al., Mar. 2006]. We have followed multiprocessor approach to implement reconfigurable FEC kernel where each processor performs state computation operation. Moreover, within a particular type of channel coding, the code rates and polynomials are also variable, this translates to a need of high flexibilty

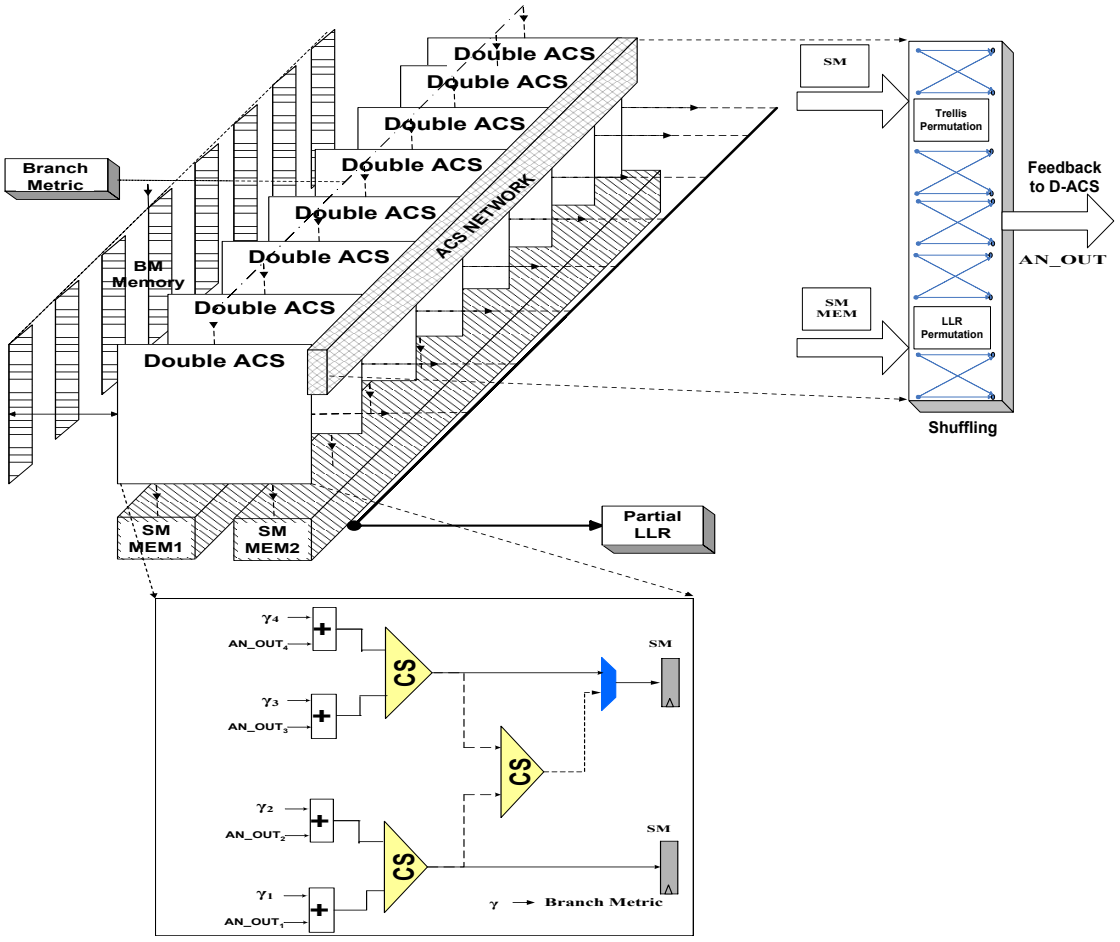

Figure 5.2: Multistandard FEC Kernel for Turbo and Viterbi Decoding.

inside the trellis kernel.

The kernel consists of 8 processing units named Double-ACS (D-ACS) as the basic building block of Max-Log-MAP algorithm implementation. Each of these units contains 4 adders and 3 compare select (CS) elements. The kernel performs all the forward and backward state metric (SM) computation and certain stages of LLR calculation (partial LLR) involving similar add-compare-select operations. The architecture can process binary turbo codes directly or reusing duo-binary trellis, thanks to trellis compaction [Black and Meng, Dec. 1992]. One D-ACS unit is assigned to each state (2 states for binary codes) and the interconnection between these units is established by means of a network (ACS Network) that maps multiple trellis diagrams for different turbo codes. Each D-ACS unit caters to 4 trellis transitions. Hence the maximum trellis transition parallelism is 32, which very well supports the requirement of current standards as shown in Table 5.1 and Table 5.2.

For a more efficient reuse of the computing resources, for lower states code(4,8) forward and backward recursions are performed in parallel. The most critical factor that affects the processing

Table 5.1: FEC Kernel Parameters in different standards for CC coding

Standard	Code Rate	Constraint Length	States	Throughput (Mbps)
UMTS	1/2, 1/3	9	256	0.064
CDMA 2000	1/2, 1/3	9	256	0.038
DVB	1/2	7	64	32
DAB	1/4	7	64	1.1
WiLAN	1/2	7	64	54
GSM	1/2	5	16	0.0096

Table 5.2: FEC Kernel Parameters in different standards for Turbo coding

Standard	Code Type	Constraint Length	States	Throughput (Mbps)
WiMAX 802.16e	DUO-BINARY	4	8	70
DVB-RCS	DUO-BINARY	4	8	31
UMTS	BINARY	4	8	2.3
CDMA 2000	BINARY	4	8	2
3GPP2	BINARY	5	16	-

efficiency is the communication scheme of moving data between D-ACS units. Since the reconfigurable platform supports different communication standards, multiple trellis states with different communication requirements needs to be supported. Furthermore the kernel is reused for Partial LLR computation (to perform certain stages of ACS operations of LLR computation), such a reuse necessisates extra connections on the ACS network element of the kernel.

For Viterbi decoding case, 2 states are mapped to one D-ACS unit. The platform is capable of handling upto 16 states CC as fully parallel architecture, for the states higher than 16 viz. 32-256 large number of states are mapped to one D-ACS following the approach mentioned in [Min, June 1991]. Similar to [Min, June 1991] the de-brujin network connectivity for the 16 states is further used to emulate the metric updating of higher state codes. N states are partitioned over 8 D-ACS with V consecutive states in each of them. The number of steps required in one decoding cycle will be $V/2$ as two states are treated in one step. For minimizing the time of one step, D-ACS should be able to read simultaneously from local memory two state metrics for updating in parallel and then simultaneously write the results into memory. For this purpose, metrics accessed together are in separate physical memories (SM1 and SM2 in figure 5.2). The memory module is organized in two separate banks such that one stores the metrics of local states with even parity and the other, those with odd parity; simultaneous access of two metrics can be achieved. Once the metric is read from each bank, the new metric can be written at that place. The figure 5.3.a shows such a memory organization. The methodology can be explained with an example of V=8: at decoding step j=0 figure 5.3.b shows the switch position as well the control signal. At step j=1 the metrics to be read are 010 and 011 thus simultaneous read from two memory modules is effectuated with CTRL set to 1 forcing the switch to be at cross position, as shown in figure 5.3.c. At the end of step j=1 as

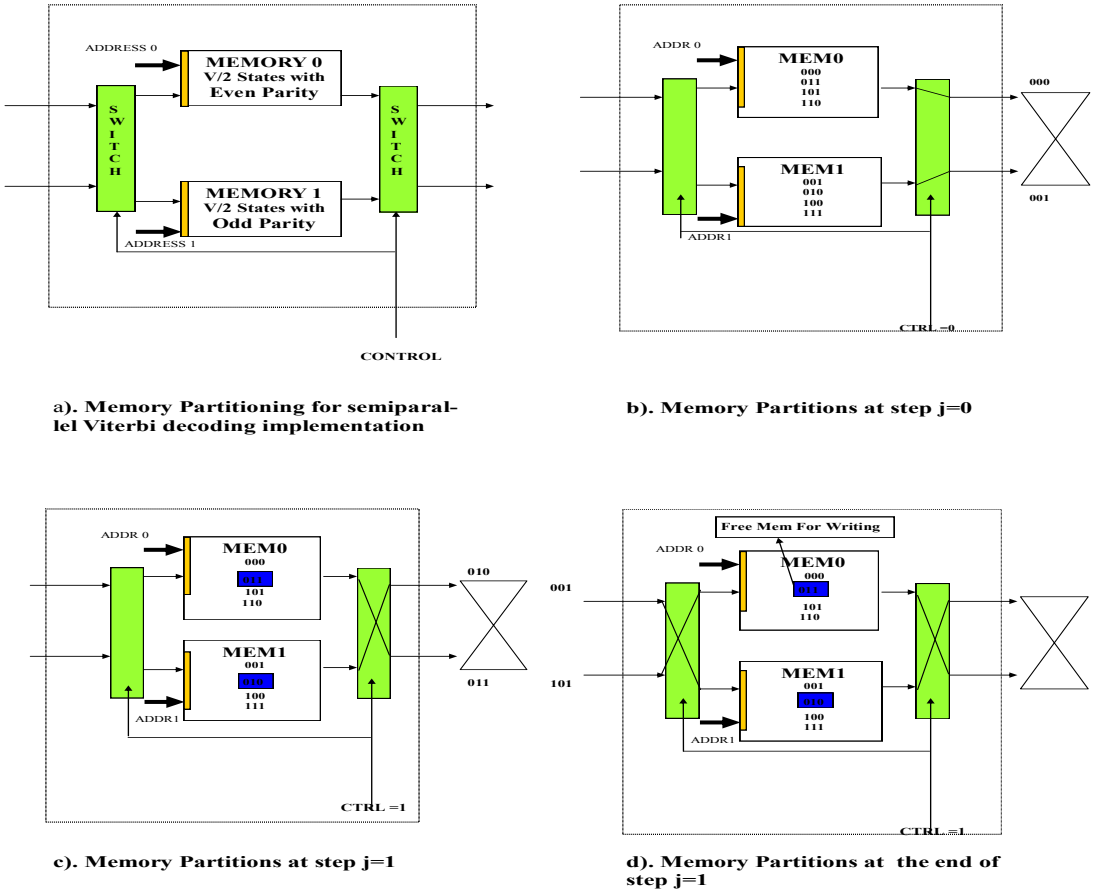

Figure 5.3: D-ACS associated memory orgnization and access for higher state viterbi decoding.

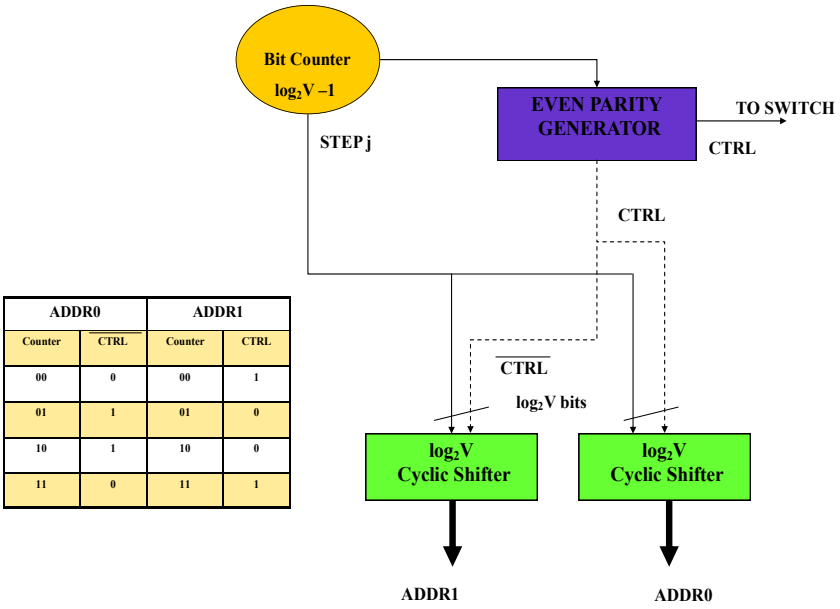

Figure 5.4: Memory Control for semiparallel implementation of Viterbi decoding

shown in figure 5.3.d locations are free for writing in the two memories however values written are not of the same state. Thus the addressing mechanism needs to be such that for the next decoding cycles correct state metrics are read at the beginning of each step. With elaboration it can be seen that each state changes its address in memory every decoding cycle and comes back to its original position after $\log_2 V$ decoding cycles. The figure 5.4 shows the generation of such control mechanism for memory. The sequence of addresses can be derived from the repeated left cyclic shift of step j.

5.3.1 ACS Network Reuse

As mentioned in previous sub-section, interconnection between D-ACS units is established according to trellis diagram of the code. As shown in Table 5.1 and Table 5.2, in mobile and wireless communication systems the constraint length of convolutional codes varies typically between (5-9) implying states to be supported as 16-256. Convolutional codes are used as component codes for turbo codes used in emerging wireless standard. Here the constraint length typically varies between 3-5, implying states to be supported as 4-16. In addition to this, WiMax(802.16e) and DVB standards use duo-binary turbo codes where each state in the trellis has four incoming and four outgoing branches. Since the reconfigurable platform supports different communication standards, multiple trellis states with different communication requirements need to be supported. The simplest solution to support this flexibility would be a fully interconnected network as designed in [Muller et al., Mar. 2006], which however would occupy significant area. For our kernel design interconnection between 16 outputs of kernel which are fed back to the 32 inputs based on trellis

Table 5.3: ACS Interconnection scenarios

Identifier	Interconnection Scenario
$\star$	16 and higher State CC
1	8 state Duo-Binary turbo forward trellis
2	8 state Duo-Binary turbo backward trellis
3	16 state Binary turbo forward trellis
4	16 state Binary turbo Backward trellis
5	8 state Binary turbo forward/backward trellis
6	4 state Binary turbo forward/backward trellis
7	16 state Binary turbo LLR computation
8	8 state Duo-Binary turbo LLR computation
9	16 state Binary turbo LLR computation
+	4 state Binary turbo LLR computation

structure, a fully interconnected network will require (32x15=480) 2-1 multiplexers. For a more judicious use of hardware resoure we propose to map different possibile communication scenarios resulting from the need to support different standards; such scenarios are classified in Table 5.3.

The Kernel is reused for Partial LLR computation (to perform certain stages of ACS operations of LLR computation), such a reuse necessitates extra connections on the ACS network element of the kernel. For the forward recursion network layout is organized as De-brujin graph [Min, June 1991]. Such a layout for state 4,8,16 binary turbo code is implemented, similary the layout derived for backword recursion is also implemented. Further more for duo binary turbo code 8 state forward and backword trellis interconnection layout is also implemented. Figure 5.5 shows the interconnection map for all the trellis recursion for different scenarios under consideration, while Figure 5.6 shows the interconnection map when LLR computations are also taken into account. D-ACS inputs are along rows and outputs are mapped along columns.

As can be seen our manual mapping approach results in common interconnection between different scenarios of the codes being implemented providing a signficant hardware reduction. Taking into account forward and backward trellis computations for all the codes supported in standards under consideration and possibility of reuse of kernel for partial LLR computation, the designed network still occupies 70 % less area in terms of mulitplexers used as compared to fully interconnected network as designed in [Muller et al., Mar. 2006].

5.4 FEC Kernel Reuse for LDPC decoding

As described in *chapter 1* LDPC decoder is defined by its parity check matrix H of M rows by N columns. Each column in H is associated with one bit of the codeword or variable node (VN), and each row corresponds to a parity check equation or check node (CN). A multistandard LDPC decoder requires great amount of flexibility to support varying parameters mentioned in emerging

	out1	out2	out3	out4	out5	out6	out7	out8	out9	out10	out11	out12	out13	out14	out15	out16	MUX(2-1)
in1	*123456																0
in2		4	*1356		2												2
in3					1				24								1
in4							1			4			2				2
in5	24				*356				1								2
in6		4			2		*356				1						3
in7									24				1				1
in8										4			2		1		2
in9	126	5	4						*3								3
in10		6	1	45	2						*3						4
in11					1		6		2		4						3
in12							1	6				4	2				3
in13	26		4			5			1				*3				4
in14		6		4	2			5			1				*3		5
in15							6		2		4		1				3
in16								6				4	2		1		3
in17	1	*3	2		4				5								4
in18			1	*3		4	2			5							4
in19					1						2		45				2
in20							1							45	2		2
in21			2		4	*3			15								3
in22						4	2	*3		5	1						4
in23											2		145				1
in24														45	12		1
in25	1		2				4			*3	5						4
in26			1				2	4				*35					3
in27					1						2				45		2
in28							1								2	45	2
in29			2				4		1		5			*3			4
in30							2	4			1	5				*3	4
in31											2		1		45		2
in32															12	45	1
															Total		84x(2-1)

Figure 5.5: ACS Interconnection Matrix for trellis computation with D-ACS inputs mapped across rows and outputs along columns

	out1	out2	out3	out4	out5	out6	out7	out8	out9	out10	out11	out12	out13	out14	out15	out16	MUX(2-1)
in1	*123456789+																0
in2	78	4	*13569 +		2												3
in3	8		7		19+				24								3
in4	8		7				19+			4			2				4
in5	24		8		*3567				19								3
in6		4	8		27		*356				19						4
in7			8				7		24				19				3
in8			8				7			4			2		19		4
in9	126	59+	4		8				*37								4
in10		6	1	459+	28				7		*3						5
in11					18	9+	6		2		47						4
in12					8		1	69+			7	4	2				5
in13	26		4			5	8		1	9			*37				6
in14		6		4	2		8	5			1	9	7		*3		8
in15							68		2		4		1	9	7		5
in16							8	6				4	2		17	9	5
in17	1	*37	2		4				589+								4
in18		7	1	*3		4	2		89+	5							6
in19				7	1				8		29+		45				4
in20				7			1		8		9+			45	2		5
in21			2		4	*37			15		8		9				5
in22						47	2	*3		5	18		9				4
in23								7			28		145		9		3
in24								7			8			45	129		3
in25	1		2				4			*379+	5		8				5
in26			1				2	4		79+		*35	8				5
in27					1						2	79+	8		45		4
in28							1					79+	8		2	45	4
in29			2				4		*1		5			*379	8		5
in30							2	4			1	5		79	8	*3	6
in31											2		1		458	79	3
in32															128	4579	1
															Total		133x(2-1)

Figure 5.6: ACS Interconnection Matrix for trellis and Partial LLR computation with D-ACS inputs mapped across rows and outputs along columns

Table 5.4: FEC Kernel Parameters in Different Standards for LDPC decoding

Standard	Code Rates	CN degree	VN Degree
WiMax 802.16e	1/2 to 5/6	6 to 20	2 to 6
WiFi 802.11n	1/2 to 5/6	7 to 22	2 to 12
DVB-S2	1/4 to 9/10	4 to 30	2 to 13

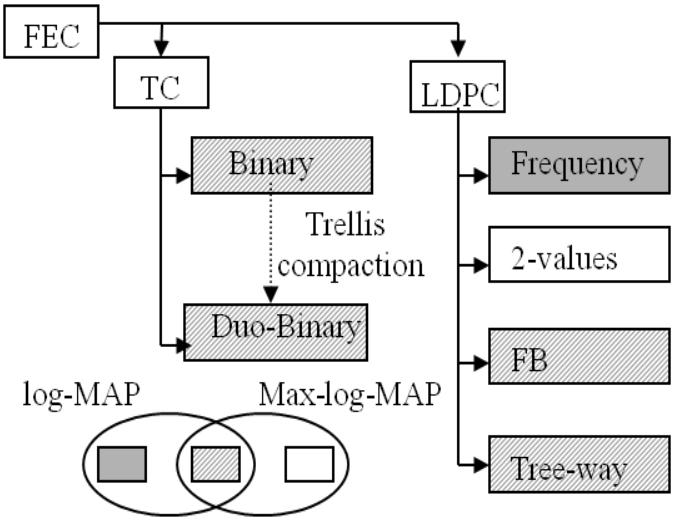

Figure 5.7: State Metric (turbo) and Check Node (LDPC) processing implementations

wireless standards as shwon in Table 5.4. Since we are targeting a check-node-centered computation scheme, VNs are reduced to storage places and all computations are carried out by the CN processing unit [Fossorier and Imai, May 1999]. The LLRs a CN needs to produce for all connected VNs can efficiently be computed by applying the different sub-optimal check node approximations of low complexity as shown in right half of the Figure 5.7.

Frequency domain computation kernel [Hagenauer et al., Mar. 1996] involves addition operation and look up table (LUT) and is far away from turbo kernel's ACS operations, thus it will not be considered in this work. The forward-backward (FB) way and 2 Value approachs are described in following sub sections while the proposed Tree way approach will be dealt with in detail in later sections .

5.4.1 FB Way

The forward backward (FB) way is similar to turbo processing on a 2 state trellis and was first proposed by [Mansour and Shanbhag, Aug. 2002]. The LLRs values received by the CN_j from connected VNs are used as input metrics I_i, with i $= 1 \cdots d_c$(check node degree of CN_j). The LLR (e_i) produced for connected VNs are then computed by taking the appropriate state metric from the forward and backward recursion, α_i and β_i, respectively, as shown in Figure 5.8. When using min-sum (which is similar as Max-Log-Map from implementation perspective) for FB way

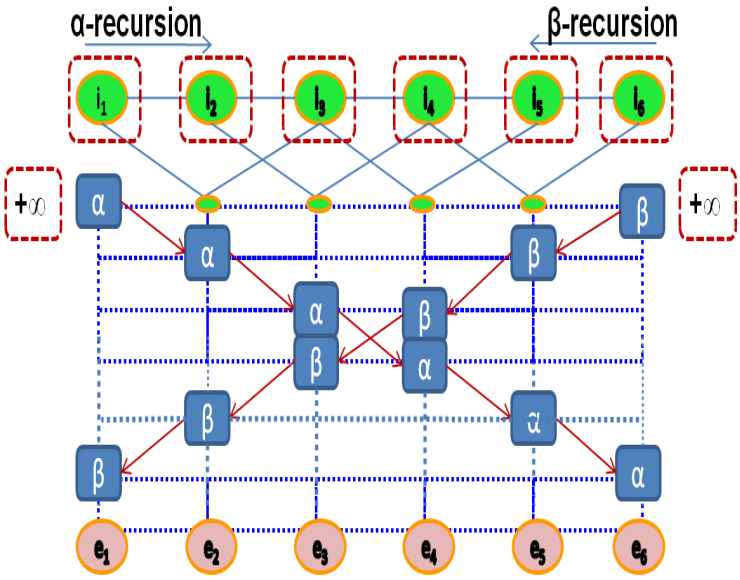

Figure 5.8: Forward Backward LDPC CN processing

[Priewasser et al., Oct. 2008] the basic function used for LLR computation is given as following:

$$f(a,b) = min(|a|,|b|) \cdot sgn(a) \cdot sgn(b). \tag{5.2}$$

Additionally, the boundary conditions need to be set at $\alpha_i = +\infty$ and $\beta_i = +\infty$ for the first element of the forward and the backward recursion, respectively.

While reusing the turbo decoding kernel for LDPC decoding utilising FB way, 8 check node computation can be processed in parallel, each of them mapped onto separate D-ACS unit. Incorporating LDPC functionality in the turbo kernel results in increased complexity of the basic D-ACS unit. For example Fig. 5.9 shows a check node architecture utilising the FB computation approach, where turbo D-ACS unit is reused for α, β and LLR calculation. The inputs AN_OUT corresponds to updated state metric values routed through ACS network in turbo decoding case. It can be seen that multiplexers units, AND gates and one additional compare-select units are introduced in D-ACS while supporting the LDPC functionalities.

5.4.2 2 Value Way

2 values calculation method first proposed in [Fossorier and Imai, May 1999] exploits the fact that in min-sum decoding out of all CN LLRs of a CN only two magnitudes are of interest, since only the minimum (MIN) and the second minimum magnitude (MIN1) with index value of the minimum (MINDEX) are used to produce LLRs for connected VNs. It is the natural way to compress data in memory and normally results in significant memory saving in CN kernel. For example when check node degree is 30 this approach saves roughly a factor of 10 in memory space when we only store

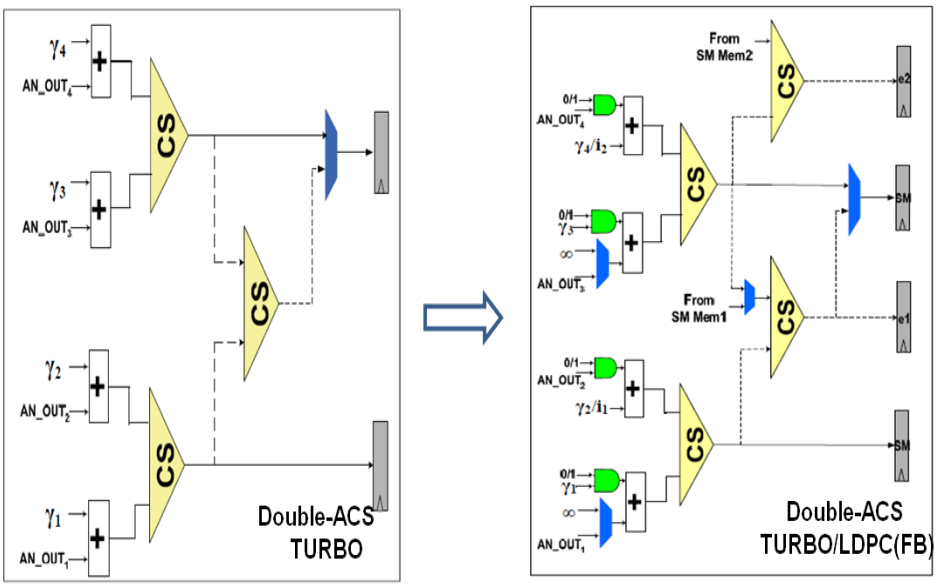

Figure 5.9: LDPC check node processor (FB) reusing the turbo D-ACS unit

those 3 numbers. Figure 5.10 shows the proposed implementation strategy for the CNP employing min-sum approximation. For each check node with degree d_c it performs following operations:

1. Computation of the exclusive or (XOR) of all hard bits output by the connected variable nodes (as determined by the signs of the output log-likelihood ratios), except the j^{th} one (j=1,2,...,d_c). A cheap way in hardware to compute all the d_c xor results is to compute the XOR of all d_c input bits using sign accumulator block. The XOR of all d_c input bits except one can be calculated by taking the XOR of the aforementioned total and then taking the XOR with the input bit (sign of MV2C) under consideration.

2. MIN and MIN1 values of the inputs are computed in serial based on algorithm shown in 5.11 and requires only 2 compare select logic elements and few multiplesxers in the hardware. New check to variable messages (MC2V)are computed using the index of the MIN, sign vector and a compare-select unit.

For further details of the implementation aspects of 2 Value check node architectures, readers are referred to [T. Bhatt and McCain, May 2006].

While reusing the turbo decoding kernel for LDPC decoding utilising 2 value computation approach, 8 check node computations can be processed in parallel, each of them mapped onto separate D-ACS unit. Fig. 5.12 shows the D-ACS units with both turbo/viterbi and LDPC(2-Value) functionalities compared to only turbo/viterbi functionality. Turbo/viterbi D-ACS unit is reused for MIN and MIN1 calculation by the min-finder block. The inputs AN_OUT corresponds to updated state metric values routed through ACS network in case of turbo decoding. The dataflow for implementing the min finder algorithm is highlighted by the the dotted lines. It can be seen that

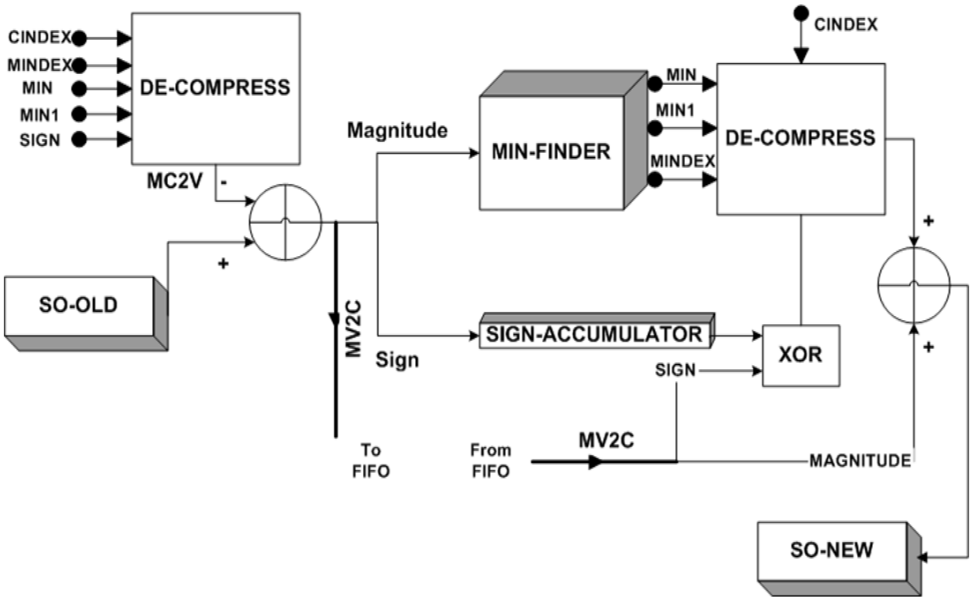

Figure 5.10: 2 MIN LDPC CN processing

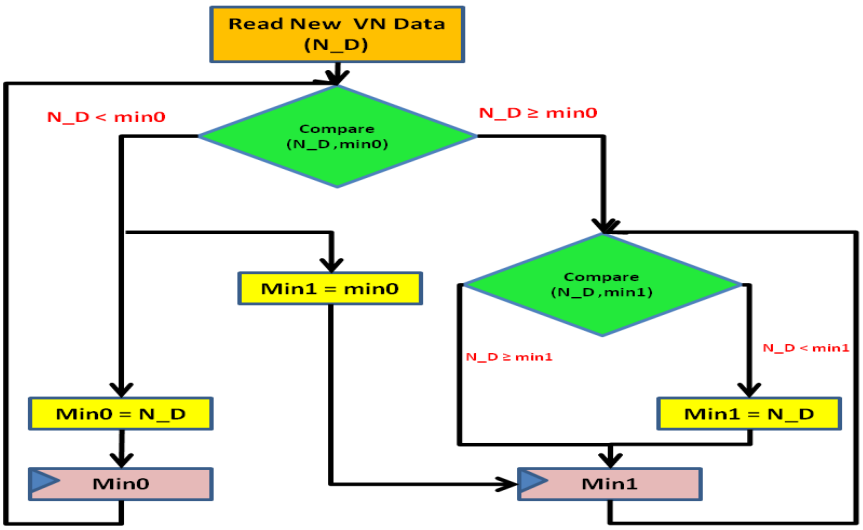

Figure 5.11: 2 MIN LDPC Min Finder Scheme

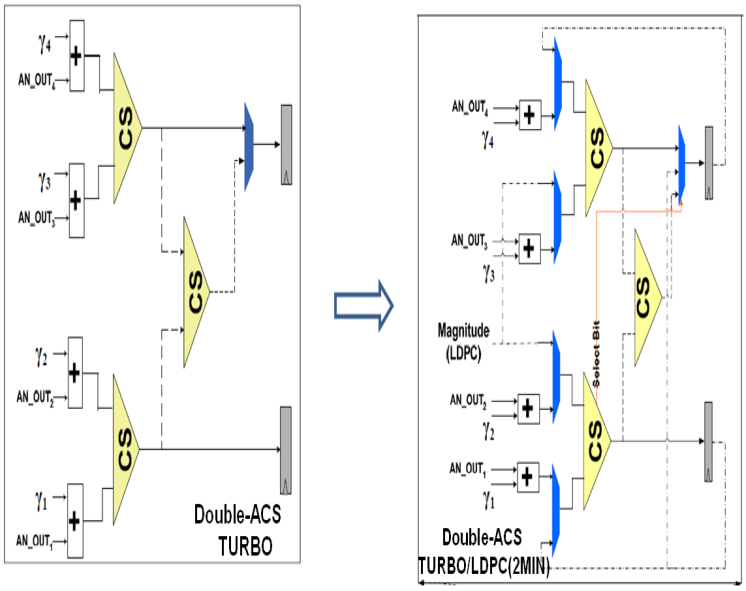

Figure 5.12: LDPC check node processor (2 values computation) reusing the turbo D-ACS unit as min finder block.

extra multiplexers units (Level 2 and 3 MUXs) are introduced in turbo D-ACS while supporting the min finder functionality of LDPC decoding, which doesnt utilises the Level 1 adder units. An effort on the reuse of Level 1 adders for accomplishing the addition operations in check node processor outside the min finder block results in lesser area gain and in some cases an increase in area possibily because of increase in the control logic for handling the different cases. Implementation results are summarized in Table 5.6 presented in next section.

5.5 Tree-Way Implementation Scheme

In this section we present a parallel implementation scheme of check node updates, developed keeping in mind the trellis computation kernel of turbo decoding. Fig 5.13 shows a simple graphical representation of the approach. For check node degree $d_c = 6$, each VN (i_1,i_2, ...i_6) is represented as a leaf node and tree is traversed performing min calculation at branch nodes untill VN extrinsics are derived at the root nodes (e_1, e_2, ...e_6). One of the key contributions of this work is that for a parallel check node architecture we generalize the tree network connectivity and the data flow for any value of d_c and present a fairly simple control mechanism for it. For the sake of architecture uniformity odd d_c values are considered as their even counterpart with extra VN intrinsic value initialized at $+\infty$ (i.e. $d_c^{'}$=d_c if d_c is even; else $d_c^{'}$=d_c+1).

Fig 5.14 shows the different stages of VN extrinsic calculation for proposed "Tree-Way" scheme, other than sign accumulation which is performed by separate *XOR* tree (not shown in figure), VN extrinsic calculation consists of following stages:

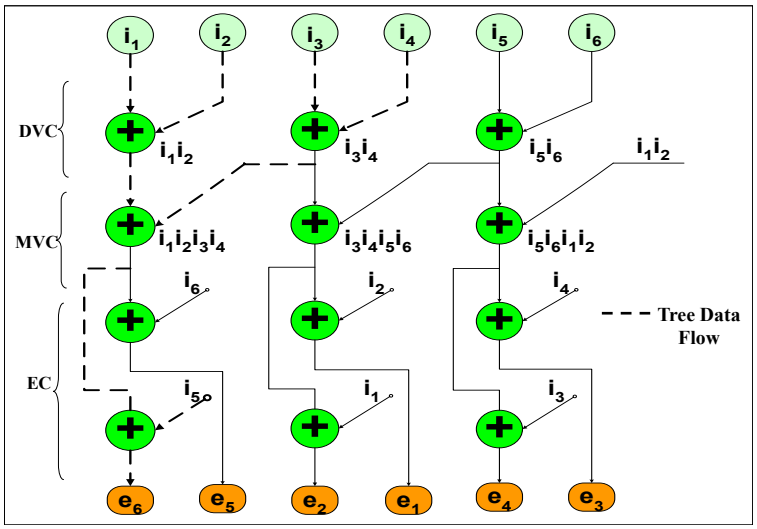

Figure 5.13: Parallel Tree-Way Calculation of VN extrinsics

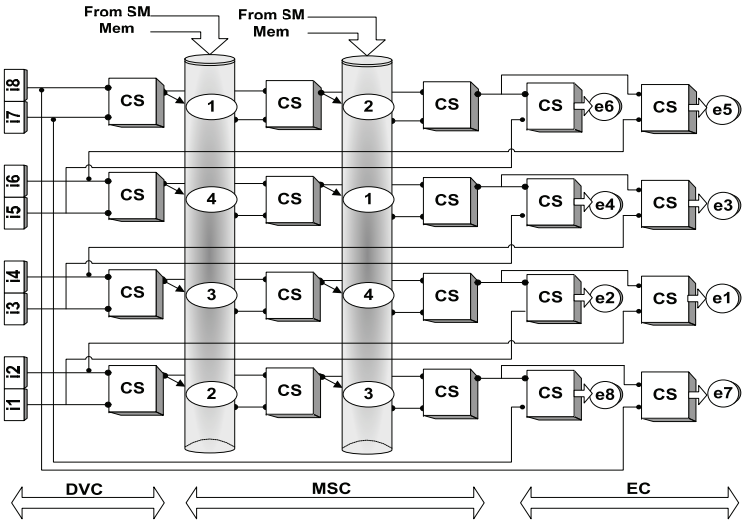

Figure 5.14: Generalization of Tree-Way Scheme

Table 5.5: Clock Cycle requirement and Shift Permutation.

$\mathbf{d}_c$	$\mathbf{N}_{CC}$	**Permutation**	$\mathbf{d}_c$	$\mathbf{N}_{CC}$	**Permutation**
5,6	4	1	19, 20	7	1, 2, 4, 8
7,8	5	1, 2	21, 22	7	1, 2, 4, 8
9,10	5	1, 2	23, 24	8	1, 2, 4, 8, 10
11,12	6	1, 2, 4	25, 26	7	1, 2, 4, 8
13,14	6	1, 2, 4	27, 28	8	1, 2, 4, 8, 12
15,16	7	1, 2, 4, 6	29, 30	8	1, 2, 4, 8, 12
17,18	6	1, 2, 4	31, 32	9	1, 2, 4, 8, 12, 14

5.5.1 Direct VN comparison (DVC) stage

As seen in Fig 5.14 for $d_c^{'}$=8, the intrinsic values (i_1,i_2, ...i_8) are fed parallely to two D-ACS units that are configured as 4 CS units. For all values of d_c there is only one direct comparison stage. The output of DVC and each subsequent stage is passed on to next stage through ACS network (AN_OUT) as well as stored in state metric memory (see Fig 5.2) for use in later stages.

5.5.2 Multiple Shuffled comparison (MSC) stage

Shuffled comparison stage could be compared to the trellis computation stages in BCJR algorithm for turbo decoding. The shuffle network implements a circular shifting permutation, which can be easily mapped on to acs network without significant hardware cost. The rotational shift depends on $d_c^{'}$ and the substage of the shuffled comparison stage. There are multiple shuffled stages depending on $d_c^{'}$ and equal to N_{cc}-*3* where (N_{cc}) is the required number of clock cycles for check node update. For different values of d_c Table 5.5 provides the information on N_{cc} values and shift associated with each suffled stage. These permutations are stored in similar way as trellis configuration in case of turbo decoding for different codes.

As can be seen from Fig 5.14 the input to the shuffle network is either the output from the immediate previous stage or output of a much earlier stage stored in the state metric memories. The connections are shown for output of earlier stages stored in the state metric memories. Some of the empty coulmns signify that for those values of $d_c^{'}$ input to the shuffle network is the output from the immediate previous stage for all the subsequent stages. For a given stage, the address to be accessed in SM memory depends on $d_c^{'}$ value and can be derived using a relatively simple control. Fig 5.15 shows the memory access pattern for $d_c^{'}$ up to 32. Vertical binary level '10' represent DVC stage output, while '100' and '1000' represent first two MSC stage outputs stored in SM memory. The bottom horizontal row shows the binary representation of the value $d_c^{'}$-2. For a given $d_c^{'}$ the address can be derived directly from corresponding binary value in the bottom layer. For example for $d_c^{'}$=16, corresponding binary representation of d_c-2 = 1110 i.e. 1000+100+10, thus address for memory access during the shuffle stage corresponds to value at binary level of 100 and 10.

At this point it is interesting to evaluate the hardware complexity of the MSC shuffled network.

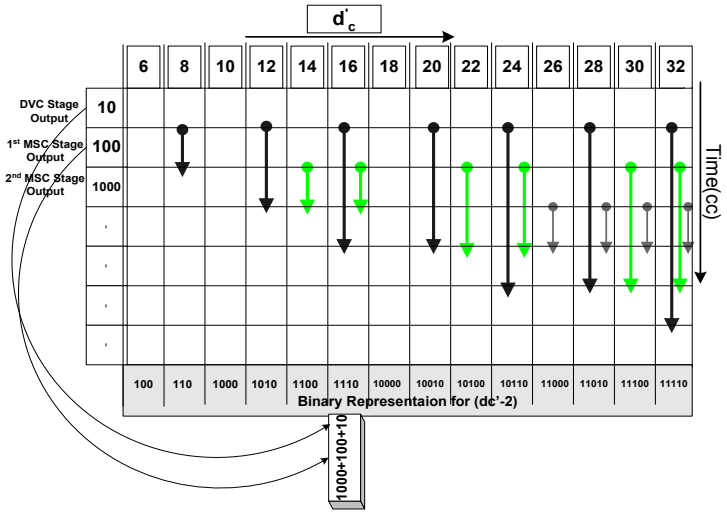

Figure 5.15: Memory Access During Shuffle Stage

Figure 5.16 shows the interconnection map for all the permutation for different check node degrees shown in Table 5.5 over a 32x16 matrix. It can be seen that designed network occupies 70 % less area in terms of multiplexers used as compared to fully interconnected network. From the perspective of reusing the ACS-Network of Turbo/Viterbi kernel for LDPC decoding these connection are mapped over the interconnection matrix for Turbo/viterbi case shown in Table 5.6. The resultant interconnection map is shown in Table 5.17 where 'x' corresponds to extra connection needed for Turbo/Viterbi fucntionality. As can be seen our manual mapping approach results in common interconnection between these two scenarios providing a signficant hardware reduction. The designed network still occupies 50 % less area in terms of mulitplexers used as compared to fully interconnected network as designed in [Muller et al., Mar. 2006] and supports turbo, viterbi and LDPC decoding functionalities.

5.5.3 Extrinsic Calculation(EC) stage

The last two stages for any value of $d_c^{'}$ are extrinsic calculation stage. As seen in Fig 5.14 input to these stages is the output from the last MSC stage and the shifted VN intrinsic values. This shift is circular over $d_c^{'}$ and equal to 1 and 2 for EC stage 1 and 2 respectively.

5.5.4 D-ACS unit implementation and ASIC Synthesis Results

Fig. 5.18 shows the D-ACS units with both turbo/viterbi and LDPC(Tree-way) functionalities compared to only turbo/viterbi functionality. It can be seen that, this flexibility comes at the cost of introduction of multiplexers and logic gate layers in the architecture. Level 1 AND gates and Level 3 MUX are driven by the choice of FEC decoding algorithms (i.e. Turbo-binary/Turbo-duo binary/ LDPC), while level 2 MUXs are supporting inputs from different stages of "Tree-Way"

	out1	out2	out3	out4	out5	out6	out7	out8	out9	out10	out11	out12	out13	out14	out15	out16	MUX(2-1)
in1	■																0
in2		■	■		■		■		■		■		■		■		7
in3		■															0
in4			■	■		■		■		■		■		■		■	7
in5			■														0
in6	■			■	■		■				■				■		5
in7				■													0
in8	■	■			■	■		■				■				■	6
in9					■												0
in10	■	■	■			■	■		■				■				6
in11						■											0
in12	■	■	■	■			■	■		■				■			7
in13							■										0
in14	■	■	■	■	■			■	■		■				■		8
in15								■									0
in16	■	■	■	■	■	■			■	■		■				■	9
in17									■								0
in18	■	■	■	■	■	■	■			■	■		■				9
in19										■							0
in20	■	■	■	■	■	■	■	■			■	■		■			10
in21											■						0
in22	■	■	■	■	■	■	■	■	■			■	■		■		11
in23												■					0
in24	■	■	■	■	■	■	■	■	■	■			■	■		■	12
in25													■				0
in26	■	■	■	■	■	■	■	■	■	■	■			■	■		12
in27														■			0
in28	■	■	■	■		■	■	■		■	■	■			■	■	11
in29															■		0
in30	■	■	■	■			■	■			■	■	■			■	9
in31																■	0
in32	■	■		■				■				■		■			5

Total 134x(2-1)

Figure 5.16: ACS Interconnection Matrix for Tree computation with D-ACS inputs mapped across rows and outputs along columns

	out1	out2	out3	out4	out5	out6	out7	out8	out9	out10	out11	out12	out13	out14	out15	out16	MUX(2-1)
in1																	0
in2	x																8
in3	x		x		x				x								4
in4	x						x						x				10
in5	x				x				x								3
in6		x															6
in7			x				x		x				x				4
in8			x				x			x			x		x		11
in9	x	x	x						x								4
in10				x	x						x						9
in11					x		x				x						3
in12					x						x	x	x				11
in13	x		x			x			x	x			x				6
in14							x					x	x				11
in15							x		x		x		x	x	x		6
in16							x	x					x		x		13
in17	x	x	x		x												4
in18									x								10
in19				x	x				x		x		x				5
in20									x						x		12
in21			x		x	x			x				x				5
in22										x	x						13
in23								x			x		x		x		4
in24											x				x		14
in25	x		x				x			x							4
in26												x	x				14
in27					x						x	x	x		x		5
in28													x				12
in29			x				x		x		x			x			8
in30														x	x		11
in31											x		x		x		3
in32															x	x	7
															Total		237x(2-1)

Figure 5.17: ACS Interconnection Matrix for Kernel computation with D-ACS inputs mapped across rows and outputs along columns

implementation.

Upper half of the Table 5.6 shows the comparison of ASIC synthesis results for different check node architectures mapped onto turbo decoding kernel. At the D-ACS unit level FB approach is the most promising as it shows a datapath area saving of 17.2 % compared to sum of two dedicated architectures for turbo and LDPC decoding. On the other hand lower half of the table shows the comparison of ASIC synthesis results at a higher level of hierarchy of FEC kernel. It can be seen that though LDPC kernel with tree way approach occupies larger area than its FB and 2MIN counterparts, the shared kernel architecture complexity remains almost the same for all three approaches, thanks to the reuse of turbo ACS network in case of "Tree-Way" approach. It should be noted that, FB and 2MIN approaches being inherently serial, take longer time for check node updates, while "Tree-Way" approach parallizes the check node computation and results in lesser clock cycles as shown previously in Table 5.5.

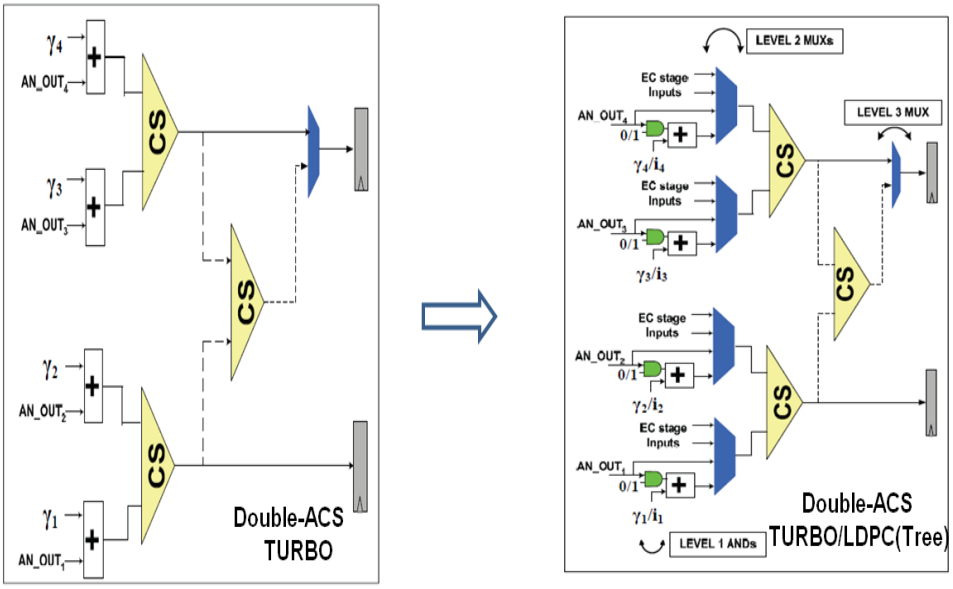

Figure 5.18: D-ACS unit with LDPC (Tree-way) functionalities.

Table 5.6: Synthesis Comparison Results[Logic Gates].(0.13μm at 300MHz)

Cost of FEC D-ACS Unit H/W									
Independent Architectures				Combined Architectures					
TC	LDPC			Sum of Two Architectures			Shared Architecture		
	FB	2MIN	Tree	FB	2MIN	Tree	FB	2MIN	Tree
405	309	396	342	714	801	747	591 (-17.2%)	687 (-14.2%)	625 (-16.0%)
Cost of FEC Kernel H/W									
TC	LDPC			Sum of Two Architectures			Shared Architecture		
	FB	2MIN	Tree	FB	2MIN	Tree	FB	2MIN	Tree
7691	2472	3168	4803	10163	10859	12494	8628 (-15.0%)	9396 (-13.4%)	8901 (-27.2%)

5.6 Conclusion

We presented a VLSI complexity analysis of datapath sharing across three important FEC code families viz. convoluional, turbo and LDPC. A reduced complexity network was designed to realise trellis structures for different Turbo and convolutional codes used in various wireless communication systems. Furthermore, various implementation possibilities of check node architectures in LDPC decoding over a Turbo Max-Log-MAP core were explored. In addition to this, we presented the first parallel implementation of check node computations using Min-sum algorithm for LDPC decoding, which is optimized for maximun reuse of turbo decoding kernel (-28.7 % less area compared to two independent turbo and LDPC kernel) and is highly efficient in terms of clock cycles required for check node computations.

Chapter 6

Performance Evaluation of Iterative Decoding Algorithm

Contents

In this chapter, a new metric for fast and efficient performance comparison of iterative sub-optimal decoding algorithms is proposed. It is based on the estimation of a function between the A-Posteriori Probability (APP) decoded symbol of optimal and suboptimal decoding algorithms. We apply the notion of entropy to evaluate this function. The metric is tested on data sets from the different sub optimal algorithms for the duo binary turbo codes used in WiMax(802.16e) application and a (251,502) Galois Field (2^6) low density parity check (LDPC) code. Experimental results show that the values of the proposed metric correlate well with the BER performance of the suboptimal implementation of the iterative decoding algorithm. The work has been published in IEEE communications letter [Singh et al., July 2009]

6.1 Background

As shown in previous chapters, efficient implementations of iterative decoding algorithm with emphasis on small area, low power consumption and high throughput are of emerging importance. The achievement of such requirements often implies the adoption of sub-optimal choices and simplifications that affect code performance. Due to the large number of options to be tested, efficient methods for performance evaluation are of great interest.

The principle of Bit Error Rate (BER) or Frame Error Rate (FER) estimation with the Monte-Carlo (MC) simulation is well known: first fix the SNR of the simulation to determine the value of the standard deviation of the white gaussian noise in the channel, then generate a modulated codeword, add the white gaussian noise and perform the iterative decoding algorithm to compute the APP of the codeword symbols. Finally, based on APP, take a decision on the decoded symbols. If uncoded and decoded codewords differ, compute the number of errors. This process is iterated a given number of time. If one looks at the set of final APP distributions before decision and the final BER (or FER), a huge amount of information has been discarded. The question arises if it is possible to take into account the information before decision to improve the BER (or FER) estimation?

In [Hoeher et al., Sept. 2000] it was shown that the use of APP distribution offers practical advantage of numerical stability over the conventional MC simulations. In this paper, we propose to use the value of APP before decision in a different application. Our approach is closely related to the "Role Model" realm proposed in [Sayir, sept. 2008]. APP of symbols at the end of a given number of iterations in case of an optimal version of algorithm is considered as the knowledge available to the role model while APP of symbols for the cases of sub optimal versions of algorithm is considered as the knowledge available to ourself. Averaged distance between APP distribution of symbols for above mentioned two cases could be an effective and quick method to determine the performance of the sub optimal version relative to the optimal one. In this work our aim is to find a metric for such a distance calculation, so that metric and performance degradation are well related.

6.2 Classical Distribution Distances

In many science and engineering fields, the similarity between two features is determined by computing the distance between them using a certain distance metric. Many information-theoretic divergence measures between two probability distributions have been introduced and extensively studied [Ali and Silvey, 1996],[Johnson, 1979],[Kullback and Leibler, 1951]. Let a discrete distribution have probability function p_k and a second discrete distribution have probability function q_k. For a divergence measure $d(\mathrm{p}_k, \mathrm{q}_k)$ to be classified as a distance following conditions should be satisfied:

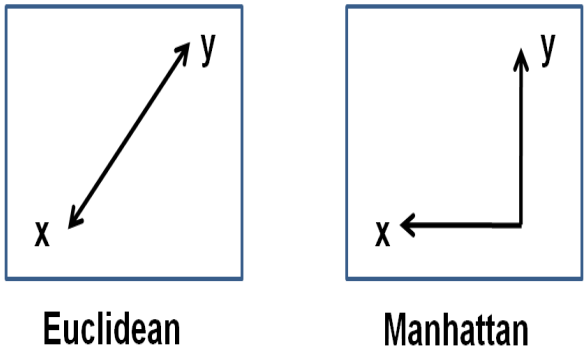

Figure 6.1: Euclidean and Manhattan Distances

1. $d(\mathrm{p}_k, q_k) = 0$ if and only if $p_k = q_k$ (Identity of indiscernibles).
2. $d(\mathrm{p}_k, q_k) = d(\mathrm{q}_k, p_k)$ (Symmetry).
3. $d(\mathrm{p}_k, \mathrm{r}_k) \leq d(p_k, \mathrm{q}_k) + d(\mathrm{q}_k, \mathrm{r}_k)$ (Triangle inequality).

Some of the most commonly used divergence metrics are discussed in the following subsections.

6.2.1 Euclidean distance

In computer vision as well as some other areas, the Euclidean distance or SSD (L_2 - sum of the squared differences) is one of the most widely used metrics. However it is most effective when it is assumed that the data have a Gaussian isotropic distribution. In mathematics, the Euclidean distance or Euclidean metric is the "ordinary" distance between two points that one would measure with a ruler, and is given by the Pythagorean formula as shown in Figure 6.1. For two discrete distribution with probability function p_k and q_k it is defined as:

$$SSD = \sqrt{\sum_k (p_k - q_k)^2} \tag{6.1}$$

6.2.2 Manhattan distance

Another common natural distance measure between probability distributions is the L_1 norm (or the Manhattan distance), defined as,

$$SAD = \sum_k |p_k - q_k| \tag{6.2}$$

as shown in Figure 6.1 it is the sum of the (absolute) differences of the coordinates of two points under consideration. L1 norm is a distance measure satisfying all the metric properties, including triangle inequality.

6.2.3 Kulbiek Luber distance

The KL distance is an information-theoretic distance measure between probability density functions [Kullback and Leibler, 1951]. The relative entropy of p with respect to q, also called the Kullback-Leibler distance, is defined by:

$$KLD = \sum_{k} log_2 \left(\frac{p_k}{q_k} \right) \tag{6.3}$$

Although KLD(p,q) $\neq$ KLD (q,p), so relative entropy is therefore not a true distance metric, it satisfies many important mathematical properties.

6.3 Entropy Inspired Distance

As explained in previous section from a maximum likelihood perspective, it is well known that the SSD is justified when the feature data distribution is Gaussian while the Manhattan distance or SAD is justified when the feature data distribution is Exponential (double or two-sided exponential). Therefore, one can determine which metric to use if the underlying data distribution is known or well estimated. Finding a suitable distance metric becomes a challenging problem when the underlying distribution is unknown and could be neither Gaussian nor Exponential. For this reason the task of finding a significant distance metric between two symbol APP distributions is not trivial. Classical distribution distances do not give any significant correlation with BER (or FER) estimation of MC simulations. For example, use of a Manhattan distance between two APP distributions does not lead to a significant correlation. This can be explained by the fact that, from a decoding point of view, probabilities of a symbol value of 10^{-6} and 10^{-12} are rather different, which is not the case when Manhantan distance is used. These considerations bring us to search a metric that takes into account both absolute difference and ratio of magnitude. At this point, a metric derived from the entropy definition of Shannon [Shannon] was tested with success. The information entropy $H(X)$ of a discrete random variable X that can take on possible values $x_1, \ldots, x_n$ is given as:

$$H(X) = -\sum_{i=0}^{n} p\left(x_i\right) log_2 p\left(x_i\right) \tag{6.4}$$

where $p(x_i) = P_r(X = x_i)$ is the probability mass function of X and entropy relates to the representation of information by quantifying its uncertainty.

In the following section, we first define quality criteria which allows us to measure if there exist a direct quantitative correlation between the proposed entropy inspired distance *EID* metric and the BER degradation of a sub-optimal decoding algorithm. *EID* metric is then defined and is compared with several classical distances in later sections.

6.3.1 Definition of Quality Criteria

For a given SNR, we defined Δ_{BER} as $\Delta_{BER} = log_{10}(BER_{sub}/BER_{opt})$. It corresponds to the BER degradation between the sub-optimal algorithm (BER_{sub}) and the optimal algorithm (BER_{opt}). Our first objective (*Case 1*) is to define a metric *EID* (to be defined later) between optimal and sub-optimal algorithm that respects the relation order between Δ_{BER} and *EID*. This means that if two design choices 1 and 2 result into suboptimal algorithms with performance given by $\Delta_{BER1} < \Delta_{BER2}$, then definition of *EID* will calculate an higher distance for choice 2. If the computational complexity to compute *EID* is low compared to BER estimation through MC simulation, then *EID* can be an efficient tool to perform design choices. Note that the ideal situation (*Case 2*) is to find a monotonic function ζ so that $\Delta_{BER} = \zeta(EID)$ in a interval of interest for making design choices: in that case, no more MC simulation is required to evaluate the BER.

6.3.2 *EID* definition

In a Non-binary iterative decoding algorithm (Turbo or LDPC code) exchanged messages can be represented as LLR vectors. A q element probability vector $P = (p_0, p_1, \ldots, p_{q-1})$ is a vector of real numbers such that $p_i > 0$ for all i and $\sum_{i=0}^{q-1} p_i = 1$. The LLR vector associated to P is $\Lambda = (\lambda_0, \lambda_1, \ldots, \lambda_{q-1})$ with $\lambda_i = log\frac{p_i}{p_0}, i = 0, \ldots, q-1$. Symbol probability as a function of LLR values is expressed as follows:

$$p_i = \frac{e^{\lambda_i}}{e^{\lambda_0} + e^{\lambda_1} + \cdots\cdots + e^{\lambda_{q-1}}} \tag{6.5}$$

In the experimental setup shown in Figure 6.2, the encoded data (a set of M codewords of length K, i.e. a total of $N = M \times K$ q-ary symbols) is modulated (Binary Phase Shift Keying in our example) and sent over a noisy channel (Additive White Gaussian Noise in our example). At receiver after demodulation, intrinsic probabilities of the received symbols are computed (Database T_{in}). The M received codewords of T_{in} are then fed to the optimal and to the suboptimal decoders. The N output of the optimal decoder are used to generate database T_{opt}. The n^{th} element of T_{opt} is an APP vector of size q denoted as $P^n = (p_i^n)_{i=0,1,\cdots q-1}$. Similarly, the output of the sub-optimal algorithm is used to generate database T_{sub}. T_{sub} is composed of APP vector $\tilde{P}^n = (\tilde{p}_i^n)_{i=0,1,\cdots q-1}$. Extending the entropy equation 6.4 we define distance *EID* in the form :

$$EID(T_{opt}, T_{sub}) = \frac{\sum_{n=0}^{N-1} \sum_{i=0}^{q-1} (|p_i^n - \tilde{p}_i^n|)\,(log_2 |p_i^n - \tilde{p}_i^n|)}{\sum_{n=0}^{N-1} -H(P^n)} \tag{6.6}$$

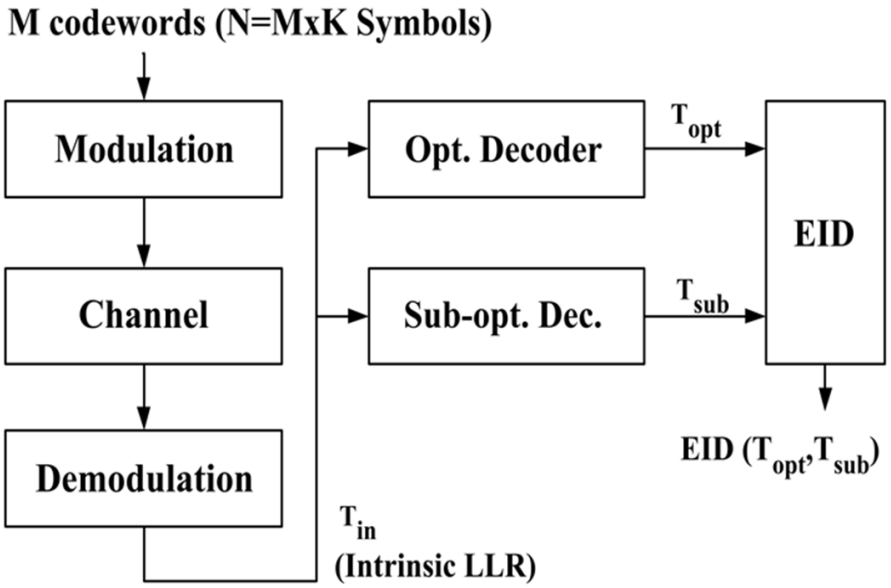

Figure 6.2: Model for Distance Evaluation System.

6.4 Experimental Results

Previously mentioned quality criteria *Case 1* and *Case 2* are subsequently verified for the proposed metric in the following experiments. A duo binary turbo code used in WiMax(802.16e) application (block length K=960 and rate=1/3) and a (251,502) Galois Field (2^6) LDPC code are used.

6.4.1 WiMax Turbo Optimal Quantization of Channel Input

Fixed point arithmetic and quantization result in additional noise in the turbo decoding system. As the rounding off noise is fixed for a given structure, increasing the signal level to quantizer could result in better performance. However it cannot be increased too much because it may cause overflow as the dynamic range of quantizer is exceeded. Thus a optimal scaling factor α for received symbol needs to be found which results in the best error performance of the decoder [Wu and Woerner, May 1999]. The system model is shown in Figure 6.3. In order to validate the performance of the *EID* metric we evaluate Δ_{BER} varying the scaling factor α. Similar experiment is performed with conventional metrics used for distance evaluation like SSD (Sum of the Squared Differences or Euclidean distance), SAD (Sum of the Absolute Differences or Manhattan distance), KLD (Kullback Leibler distance or Relative entropy). To numerically obtain the Δ BER we use channel input representation with large number of bits, thus making it a near floating point representation. BER values obtained for this floating point representation of the algorithm is used as the reference value (BER_{opt}).

Figure 6.4 shows the variation of distance calculated using the conventional metrics and the proposed metric with scaling factor α. Databases T_{opt} and T_{sub} are created with $M = 100$ frames

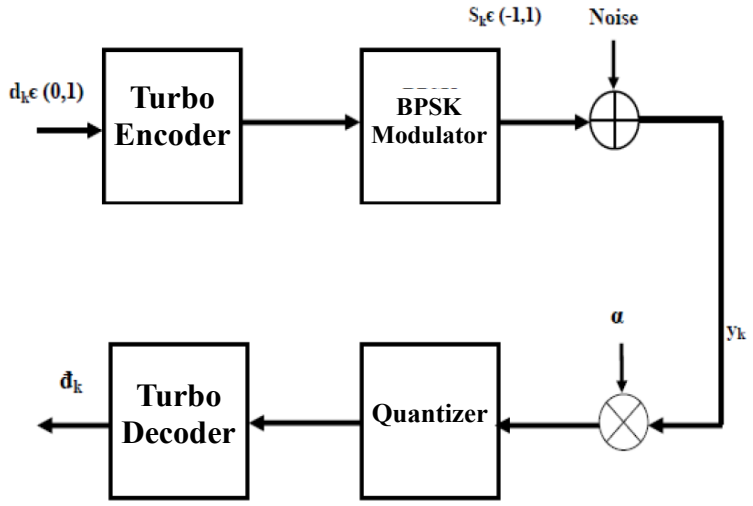

Figure 6.3: System Model for Optimal Quantization of Channel Input

successfully decoded at the end of 7 iterations. Different distances are scaled with different factors to make sure all curves fit in the same plot. Significant performance degradation is expected at both lower and higher values of α and optimal performance should be obtained somewhere in the middle. It can be observed that conventional metrics fail to capture the expected performance degradation variation with scaling factor α.

Figure 6.5 illustrates the variation of couple (Δ_{BER}, EID) for different scaling factors α. The value of α varies from 0.6 to 2.4 with step 0.2. The correlation curves are plotted for different *Eb/N0* and different code rates. The BER values are of the order of 10^{-3} and 10^{-4} for the *Eb/N0* values of 0.77dB and 0.87dB respectively. It can be seen that the couple (Δ_{BER}, EID) gives the same optimal value of scaling factor α at 1.6 for code rate $R = 1/3$ and at 1.2 for $R = 1/2$, thus validating the *Case 1* mentioned previously.

6.4.2 WiMax Turbo Extrinsic BitWidth Optimization

In serial, deterministic interleaver based or network on chip (NOC) based implementation of turbo decoders, size of the extrinsic memory, complexity of the interleaver and the communication resources of the network on chip greatly increases with the bit width of the extrinisic information. In [Singh et al., Sept 2008] it was shown that least significant bit (LSB) drop-append combined with most significant bit (MSB) clipping can be an useful method for countering these effects. We utilise this bit width optimization method to establish the correlation between BER performance and proposed distance metric. The suboptimal database corresponds to symbol probabilities in LSB drop-append and MSB clipped version of the algorithm, while algorithm with 8 bit fixed point representation for the extrinsics is assumed to be optimal.

The correlation plots between Δ_{BER} and distance metric for suboptimal algorithms (shown by

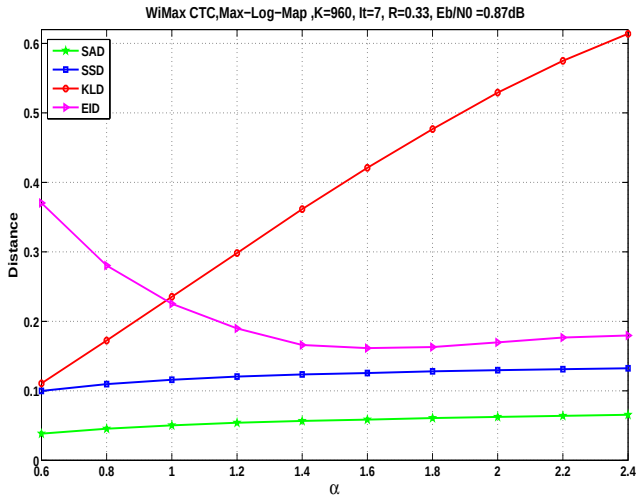

Figure 6.4: Variation of Different Distance Metrics with α.

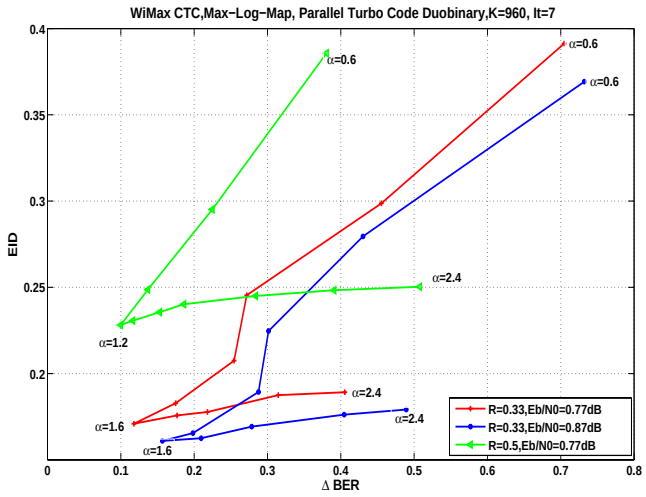

Figure 6.5: Correlation curve for Δ_{BER} and *EID* variation with α.

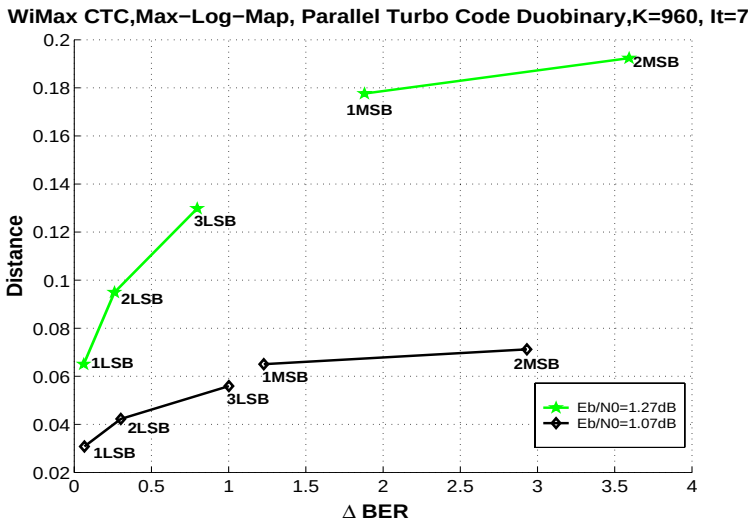

Figure 6.6: Correlation between Δ_{BER} and Distance for Suboptimal algorithms LSB drop-append and MSB clip

the dots in the curves) corresponding to 1, 2 and 3 LSBs drop append and 1, 2 MSB clip respectively is presented in Figure 6.6. The two curves correspond to different *Eb/N0*. The number of bits N used for distance metric simulations are lesser by a magnitude order of 100 compared to the Monte Carlo simulations to obtain the BER values. We can observe that for a given *Eb/N0* the correlation order is always respected between the bit width optimized sub optimal algorithms, that is, Δ_{BER} and distance both increase with increase in sub optimality. The correlation order also holds true across different *Eb/N0*.

6.4.3 LDPC GF(2^6) Case

The experiments were performed over a rate 1/2 LDPC code (251,502) in $GF(64)$. The optimal algorithm is the well known belief propagation algorithm over a non-binary LDPC code [Declercq and Fossorier, April 2007]. The sub-optimal algorithm is the Extended Min-Sum (EMS) proposed by [Voicila et al., June 2007]. In EMS, only the highest n_m probabilities of the message are consider in the decoding process, remaining (64-n_m) lower probabilities are simply discarded. Since n_m is significantly lower than 64, both memory and computational complexity are saved thanks to this approximation. Using the optimal algorithm, we have generated T_{opt} with $M = 100$ frames successfully decoded with 20 iterations. Then, several databases T_{sub} have been created with the EMS algorithm for values of n_m ranging from 6 to 32. The FER values are of the order of 2×10^{-4} and 2×10^{-6} for the SNR values of 1.4dB and 1.6dB respectively for the optimal algorithm. In Figure 6.7 correlation between Δ_{FER} and *EID* for these suboptimal algorithms (shown by the dots in the curves) is depicted. Approximately linear slope of the curves in a wide FER range validates the second quality criteria *(Case2)* mentioned previously for the proposed *EID* metric.

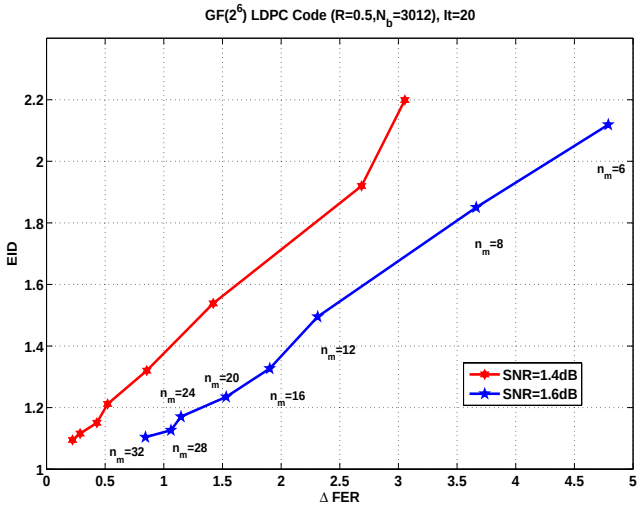

Figure 6.7: Correlation between Δ_{FER} and *EID* for optimal and sub-optimal algorithms for different message length n_m.

6.5 Conclusion

We presented a novel error performance assessment metric for sub optimal iterative decoding algorithms. It takes into account APPs measured at the end of certain iteration to estimate how far is the APP distribution of the symbol in case of suboptimal version of algorithm from the optimal version. We extended the concept of entropy to evaluate this distance. Experimental results show that the values of the proposed metric correlate well with corresponding BER performance analysis of the sub optimal iterative algorithms, giving a significant improvement in terms of simulation speed. The work provides us a practical tool to quickly compare and assess the performance of suboptimal iterative decoding algorithms.

We know that other tools of the information theory can be used for our project (like mutual information, EXIT chart and so on) but we didn't find yet a useful way of using it for our problem. This question is still open.

Chapter 7

Conclusion and Future Perspectives

The aim of this thesis was to analyze the methods and practical architectures for high performance flexible channel decoders. In particular, we focused on architectures which are able to easily adapt to different current and future standard requirements. By developing the understanding of flexible design paradigm we established the fact that we required flexible architectures in order to tame design cost, specially Non-Recurrent Engineering Cost and at the same time to support with a single circuit, most of the different operative mode required by the current and upcoming telecommunication standards.

However before focussing on new solutions, we conducted a study on digital communication and in particular on three important families of channel codes, the convolutional, turbo and LDPC codes. We have studied their structures and algorithms for their decoding. Study and analysis of various state of art implementation of turbo code decoders led to proposition of a methodology for extrinsic message size reduction. Cost, area and energy consumption of the turbo decoder implementation scales with the bit-width of extrinsic information. The presented results showed how the optimization potential through communication centric paradigm can be fully exploited without serious degradation of the bit-error performance.

Further analysis of utilised degree of parallism of iterative decoding algorithm and their implementation on MPSoC plateform led us to the problem of collisions in memory access and exploration of NoC paradigm was performed to fully understand this issue. As the inter-processor communication becomes the bottleneck for high degrees of parallelization, we presented a case study that analyzes the traffic pattern over a 2-D Torus/Mesh Network on Chip. By a detailed traffic analysis we not only showed that the overall processing throughput is impacted by the network's limited communication bandwidth, in addition choice of parallelism degree and design of processing unit also plays an important role.

After the analyses of interconnection network, we turned our attention to a flexible processing element design supporting different codes. We investigated various possibilities of hardware reuse across datapath. We presented a VLSI complexity analysis of datapath sharing across two FEC

code families viz. turbo and LDPC. Various implementation possibilities of check node architectures in LDPC decoding over a Turbo Max-Log-MAP core were explored. In addition to this, this work presented a parallel implementation of check node computations using Min-sum algorithm for LDPC decoding, which is optimized for maximun reuse of turbo decoding kernel (-28.7 % less area compared to two independent turbo and LDPC kernel) and is found to be efficient in terms of clock cycles required for check node computations for maximum operating frequency of 300 MHz.

As can be inferred, the main focus of this work was to investigate architectures with an inherent high flexibility. However, given the multitude of sub-optimal algorithms to be evaluated for their error performance to arrive at the right choice for low power, low cost implementation, a need of quicker error performance methods than the conventional Monte Carlo method was felt. In this context, we presented a novel error performance assessment metric for sub optimal iterative decoding algorithms. It takes into account APPs measured at the end of certain iteration to estimate how far is the APP distribution of the symbol in case of suboptimal version of algorithm from the optimal version. We extended the concept of entropy to evaluate this distance. Experimental results show that the values of the proposed metric correlate well with corresponding BER performance analysis of the sub optimal iterative algorithms, giving a significant improvement in terms of simulation speed. The work provides us a practical tool to quickly compare and assess the performance of suboptimal iterative decoding algorithms.

Given the multifaceted nature of our work, lot of scope for future enhancement of proposed contributions are possible. Some of the key future work which we have foreseen include but are not limited to; Decreasing the extrinsic message bandwidth over the network for increased performance of a MPSoC based iterative decoder. Especially for duo-binary turbo codes a non-uniform quantization of extrinsic message using clustering algorithm approach is envisioned. Further more message bandwidth reduction methodology needs to be evaluated for MPSoC based LDPC decoders. On the development of flexible FEC platform further work is required for efficient memory sharing across different channel decoding algorithm supported. For faster and efficient error performance assessment of iterative decoding algorithm we know that other tools of the information theory can be used for our project (like mutual information, EXIT chart and so on) but we didn't find yet a useful way of using it for our problem. This question is still remains open.

Bibliography

IEEE 802.16e. Ieee standard for local and metropolitan area networks-part 16, air interface for fixed broadband wireless access system,ieee std 802.16. 2004.

S. M. Ali and S. D. Silvey. A general class of coefficients of divergence of one distribution from another. *J . Roy. Statist. Soc., Ser. B*, 28:131–142, 1996.

M. Alles, T. Vogt, and N. Wehn. Flexichap: A reconfigurable asip for convolutional, turbo, and ldpc code decoding. *Turbo Codes and Related Topics, 2008 5th International Symposium on*, pages 84–89, Sept. 2008.

N. Axvig, D. Dreher, K. Morrison, E. Psota, L.C. Peŕez, and J.L. Walker. Average min-sum decoding of ldpc codes. *in Proc. of the International Symposium on Turbo Codes and Related Topics, Laussanne, Switzerland*, Sep. 2008.

L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT 20:284–287, Mar. 1974.

S. Benedetto, R. Garello, G. Montorsi, C. Berrou, C. Douillard, A. Ginesi, L. Giugno, and M. Luise. Mhoms: high-speed acm for satellite applications. *IEEE Trans. Wireless Commun.*, 12(2):66–67, April 2005.

S. Benedetto, G. Masera, G. Olmo, G. Guidotti, P. Pellegrino, and S. Pupolin. Primo: Reconfigurable platform for wideband wireless communications. *in Proc. 7th Int. Symp. on Wireless Personal Multmedia Communications (WPMC 2004)*, Sept. 2004.

C. Berrou and M. Jezequel. Non binary convolutional codes for turbo coding. *Elect. Letters*, 35(1): 39–40, Jan. 1999.

C. Berrou, R. Pyndiah, P. Adde, R. Bidan, and C. Douillard. An overview of turbo codes and their applications. *ECWT 05 - Proceedings of the 2nd European Conference on Wireless Technology, Paris, France*, pages 1–9, 2005.

C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. *in Proc. ICC 93*, page 10641070, May 1993.

M. A. Bickerstaff, D. Garrett, T. Prokop, and B.and Zhou G.and Davis L. M.: Thomas, C.and Widdup. A unified turbo/viterbi channel decoder for 3gpp mobile wireless in 0.18-mm cmos. *IEEE Journal of Solid-State Circuits,*, pages 1555–1564, Nov. 2002.

P. Black and T. Meng. A 140-mb/s, 32-state, radix-4 viterbi decoder. *IEEE Commun. Lett.*, 27 (12):1877–1885, Dec. 1992.

T. Blankenship, B. Classon, and V. Deai. High-throughput turbo decoding techniques for 4g. *Int. Conf. 3G Wireless and Beyond,San Francisco, CA*, pages 137–142, June 2005a.

T. Blankenship, B. Classon, and V. Deai. High-throughput turbo decoding techniques for 4g. *in Proc. Int. Conf. 3G Wireless and Beyond. San Francisco, CA*, page 137142, June 2005b.

E. Boutillon, C. Douillard, and G. Montorsi. Iterative decoding of concatenated convolutional codes-implementation issues. *Transactions of the IEEE*, 95(6):1201–1227, June 2007.

T. Brack, M. Alles, F. Kienle, and N. Wehn. A synthesizable ip core for wimax 802.16e ldpc code decoding. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC06), Helsinki, Finland*, pages 1–5, Sept. 2006.

M. A. Castellon, I. J. Fair, and D. G. Elliott. Fixed-point turbo decoder implementation suitable for embedded applications. *IEEE CCECE/CCGEI, Saskatoon*, pages 1065–1068, May 2005.

J. R. Cavallaro and M. Vaya. Viturbo: A recongurable architecture for viterbi and turbo decoding. *Proceedings of ICASSP '03*, pages 497–500, April 2003.

J. Chen and M. P. C. Fossorier. Near-optimum universal belief propagation based decoding of low-density parity check codes. *IEEE Trans. Commun.*, 50(3):406–414, Mar. 2002.

Coware. Available: http://www.coware.com.

D. Declercq and M. Fossorier. Decoding algorithms for nonbinary ldpc codes over gf(q). *IEEE Trans. on Commun.*, 55(4):633–643, April 2007.

M. Fossorier, M.P.C.and Mihaljevic and I. Imai. Reduced complexity iterative decoding of low-density parity-check codes based on belief propagation. *IEEE Transactions on Communications*, 47:673–680, May 1999.

R. G. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, pages 21–28, Jan. 1962.

A. Giulietti, L. van der Perre, and A. Strum. Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements. *Elec. Lett.*, 38(5):232–234, Feb. 2002.

F. Guilloud, E. Boutillon, and J. Danger. λ-min decoding algorithm of regular and irregular ldpc codes. *in Proc. of the 3rd International Symposium on Turbo Codes and Related Topics, Brest, France*, pages 451–454, Sep. 2003.

J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on Information Theory*, 42(2):429–445, Mar. 1996.

D. Hocevar. A reduced complexity decoder architecture via layered decoding of ldpc codes. *Signal Processing Systems, SIPS 2004. IEEE Workshop on, (Austin, USA)*, pages 107–112, Oct. 2004.

P. Hoeher, I. Land, and U. Sorger. Log-likelihood values and monte carlo simulation - some fundamental results. *Int. Symp. on Turbo Codes and Rel. Topics*, pages 43–46, Sept. 2000.

M. Hosemann, R. Habendorf, and G. P. Fettweis. Hardware-software codesign of a 14.4mbit - 64 state - viterbi decoder for an application- specific digital signal processor. *in Proc. IEEE Workshop on Signal Processing Systems 2003 (SIPS03)*, 2003.

K.-L. Hsiung, S.-J. Kim, and S. Boyd. Tractable approximate robust geometric programming. *Springer Optimization and Engineering J.*, 9(2):95–118, June 2008.

K. Huang, F. Li, P. Shen, and A. Wu. Vlsi design of dual-mode viterbi/turbo decoder for 3gpp. *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, 2: 773–776, May 2004.

Y. Isukapalli and S.S. Rao. Exploiting the nature of extrinsic information in iterative decoding. *Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, 2:1793– 1797, Nov. 2003.

ITRS. International technology roadmap for semiconductors 2003. *Semiconductor Industry Association, S. Jos[U+FFFD]SA*, 2003.

A. Jantsch and H. Tenhunen. Network on chip. *Hingham, MA, USA : Kluwer Academic Publishers*, 2003.

G. Jeong and D. Hsia. Optimal Quantization for Soft-Decision Turbo Decoder. *Proceedings of the IEEE Veh. Tech. Conf. (VTC'99)*, 3:1620–1624, Sept 1999.

R. W. Johnson. Axiomatic characterization of the directed divergences and their linear combinations. *IEEE Trans. Inform. Theory*, IT-25(6):709–716, 1979.

Z. Juntan and M.P.C. Fossorier. Shuffled iterative decoding. *IEEE Transactions on Communications*, 52(2):209–213, Feb. 2005.

G. Kreiselmaier, T. Vogt, and N. Wehn. Combined turbo and convolutional decoder architecture for umts wireless applications. *in Proc. DATE*, pages 192–197, Feb 2004.

G. Kreiselmaier, T. Vogt, and N. Wehn. Combined turbo and convolutional decoder architecture for umts wireless applications. *Proceedings of DATE*, pages 192–197, Feb. 2004.

F.R. Kschischang and B.J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *Journal on Selected Areas in Communications*, 16:219–230, 1998.

S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics 22*, pages 79–86, 1951.

D.-S. Lee and I.C. Park. Low-power log-map decoding based on reduced metric memory access. *Circuit and Systems, IEEE Transactions on*, 53(6):1244– 1253, June 2006.

Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. Soda: A low-power architecture for software radio. *in Proc. 33rd International Symposium on Computer Architecture (ISCA06*, pages 89–101, 2006.

Z. LU, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch. Nnse: Nostrum network-on-chip simulation environment. *in Proc. Of the Swedish System-on-Chip Conference (SSoCC 05)*, April 2005.

D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, Mar. 1999.

D. J. C. MacKay and R.M. Neal. Good codes based on very sparse matrices. *5th IMA Conference on Cryprography and Coding. Berlin, Germany: Springer.*, 1995.

F. J. MacWilliams and N. J. A. Sloane. The theory of error-correcting codes. *New York: Elsevier*, 1978.

M. Mamidipaka, K. Khouri, N. Dutt, and M. Abadir. Analytical models for leakage power estimation of memory array structures. *CODES + ISSS04 IEEE International Conference*, pages 146–151, Sep. 2004.

M. Mansour and N. Shanbhag. Low-power vlsi decoder architectures for ldpc codes. *Low Power Electronics and Design, 2002. ISLPED02. Proceedings of the 2002 International Symposium on, (Monterey,USA)*, pages 284–289, Aug. 2002.

M. Martina, M. Nicola, and G. Masera. A flexible umts-wimax turbo decoder architecture. *IEEE Trans. on Circuits and Systems II: Express Briefs*, 55(4):369–373, April 2008.

G. Masera and et al. Newcom dr4.1- report on reserach gaps and action plan, newcom - network of excellence for wireless communications. *Tech. Rep. DR4.1,*, Oct. 2004.

G. Masera, F. Quaglio, and F. Vacca. Implementation of a flexible ldpc decoder. *IEEE Transactions on Circuits and Systems II*, 54(6):542–546, June 2007.

G. Masera, G. Piccinini, M. Roch, and M. Zamboni. Vlsi architectures for turbo codes. *IEEE Trans. VLSI Syst*, 7:369–379, Sept. 1999.

K. (Editor) Masera, G.and Sripimanwat. Turbo code applications: a journey from a paper to realization,. *chapter 14 - VLSI for turbo codes, Springer*, (ISBN: 1-4020-3686-8):450, 2005.

A. Matache, S. Dolinar, and F. Pollara. Stopping rules for turbo decoders,. *JPL TMO Progress Report*, 42-142:122, Aug. 2000.

B. Min. Architecture vlsi pour le decodeur de viterbi. *Thesis:Telecom Paris*, June 1991.

G. Montorsi and S. Benedetto. Design of fixed-point iterative decoders for concatenated codes with interleavers. *IEEE Journal on Selected Areas in Communications*, 19(5):871–882, May 2001.

H. Moussa, O. Muller, A. Baghdadi, and M. Jezequel. Butterfly and benes based on chip communication networks for multiprocessor turbo decoding. *in Design, Automation and Test in Europe Conference and Exhibition,*, page 654659, 2007.

H. Moussa, A. Baghdadi, and M. Jezequel. Binary de bruijn interconnection network for a flexible ldpc/turbo decoder. *in proceedings of the IEEE International Symposium on Circuits and Systems*, May 2008.

O. Muller, A. Baghdadi, and M. Jézéquel. Asip-based multiprocessor soc design for simple and double binary turbo decoding. *in Proc. Design, Automation and Test in Europe (DATE '06), Munich, Germany*, pages 1330–1335, Mar. 2006.

F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. Van der Perre, and F. Catthoor. A unified instruction set programmable architecture for multi-standard advanced forward error correction. *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 31–36, Oct. 2008.

C. Neeb, M. J. Thul, and N. Wehn. Network-on-chip-centric approach to interleaving in high throughput channel decoders. *IEEE International Symposium on Circuits and Systems (ISCAS), Kobe, Japan*, pages 1766–1769, May 2005.

C. Pan, N. Bagherzadeh, A. Kamalizad, and A. Koohi. Design and analysis of a programmable single-chip architecture for dvb-t base- band receiver. *in Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 468–473, 2003.

S. M. Park, J. Kwak, and K. Lee. Extrinsic information memory reduced architecture for non-binary turbo decoder implementation. *IEEE Vehicular Technol. Conf. Spring, Singapore,*, May 2008.

A. Polydoros. Algorithmic aspects of radio flexibility. *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on*, pages 1–5, Sept. 2008.

R. Priewasser, M. Huemer, and B Bougard. Trade-off analysis of decoding algorithms and architectures for multi-standard ldpc decoder. *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 88–93, Oct. 2008.

T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke. Design of capacity approching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, Feb. 2001.

P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and sub-optimal map decoding algorithmsoperating in the log domain. *Communications, 1995. ICC '95 Seattle, 1995 IEEE International Conference on*, pages 1009–1013, Jun 1995.

S. Roman. Coding and information theory. *New York: Springer-Verla*, 1992.

J. Sayir. What makes a good role model. *submitted to IEEE trans. on Inf. Theory*, page Available at http://arxiv.org/abs/0809.1300v1, sept. 2008.

M. Scarpellino, A. Singh, E. Boutillon, and G. Masera. Reconfigurable architectures for ldpc and turbo decoding: A noc case study. *International Symposium on Spread Spectrum Techniques and Applications (ISSSTA'08), Bologna, Italy*, August 2008.

C. Schurgers, F. Catthoor, and M. Engels. Memory optimisation of map turbo decoder algorithms. *IEEE Trans. on VLSI system*, 9(2):305– 312, April 2001.

D. Seo, A. Ali, W. Lim, and N. Rafique. Near-optimal worst-case throughput routing for two-dimensional mesh networks. *Proceedings of 32nd International Symposium on Computer Architecture*, pages 432 – 443, June 2005.

C. E. Shannon. The mathematical theory of communication. *University of Illinois Press, year = 1949,.*

C. E. Shannon. A mathematical theory of communication part i. *Bell System Technical Journal*, 27:379–423, July 1948.

C. E. Shannon. A mathematical theory of communication part ii. *Bell System Technical Journal*, 27:623–656, Oct. 1948.

S. Shao, L. Shu, and M. Fossorier. Two simple stopping criteria for turbo decoding. *IEEE Trans. Commun.*, 47:11171999, 1999.

A. Singh, E. Boutillon, and G. Masera. On datapath reuse in multi-standard turbo/ldpc kernel. *Submitted to IEEE workshop on Signal Processing (SIPS)*, 2010.

A. Singh, A. Al-Ghouwayel, G. Masera, and E. Boutillon. A New Performance Evaluation Metric for Sub-Optimal Iterative Decoders. *IEEE Communications Letters*, 13(7):20–27, July 2009.

A. Singh, E. Boutillon, and G. Masera. Bit-Width Optimization of Extrinsic Information in Turbo Decoder. *IEEE International Symposium on Turbo Codes and Related Topics*, Sept 2008.

Y. Sun and J. R. Cavallaro. Unified decoder architecture for ldpc/turbo codes. *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 13–18, Oct. 2008.

V. Stolpman T. Bhatt, V. Sundaramurthy and D. McCain. Pipelined block-serial decoder architecture for structured ldpc codes. *in Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on, (Toulouse, France)*, May 2006.

R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, May 1981.

A. Tarable, S. Benedetto, and G. Montorsi. Mapping interleaver laws to parallel turbo and ldpc decoders architectures. *IEEE Trans. Inform.Theory,*, 50(9):2002–2009, Sept. 2004.

C. Thomas, M. A. Bickerstaff, L. M. Davis, T. Prokop, B. Widdup, G. Zhou, D. Garrett, and C. Nichol. Integrated circuits for channel coding in 3g cellular mobile wireless systems. *IEEE Communications Magazine,*, pages 150–159, April 2003.

M. J. Thul, F. Gilbert, and N. Wehn. Optimized concurrent interleaving architecture for high-throughput turbodecoding. *in IEEE International Conference on Electronics, Circuits and Systems*, pages 1099–1102, 2002.

F. Vacca, H. Moussa, A. Baghdadi, and G. Masera. Flexible architectures for ldpc decoders based on network on chip paradigm. *in Euromicro Conference on Digital System Design*, 2009.

J. Vogt and A. Finger. Improving the max-log-map turbo decoder. *Electronics Letters*, 36(23): 1937–1939, Nov 2000.

T. Vogt and N. Wehn. A reconfigurable application specific instruction set processor for convolutional and turbo decoding in a sdr environment. *in Proc. Design, Automation and Test in Europe (DATE 08), Munich, Germany*, Mar. 2008.

A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard. Low-complexity, low memory ems algorithm for non-binary ldpc codes. *Proceedings of ICC2007*, pages 671–676, June 2007.

Z. Wang, Z. Chi, and K.K. Parhi. Area-efficient high-speed decoding schemes for turbo decoders. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 10(6):237–242, Dec 2002.

N. H. E. Weste and K. Eshraghian. Principles of cmos digital design: a system perspective - second edition. *Addison Wesley*, 1992.

L. Woodard, J.P. Hanzo. Comparative study of turbo decoding techniques: an overview. *Vehicular Technology, IEEE Transactions on*, 49(6):2208–2233, Nov 2000.

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 Trans. VLSI Syst*, 13:439–447, April 2005.

Y. Wu and B.D. Woerner. The influence of quantization and fixed point arithmetic upon the ber performance of turbo codes. *IEEE VTC, Houston, USA*, pages 1683–1687, May 1999.

J. Zhang and M. Fossorier. Shuffled belief propagation decoding. *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*, Nov. 2002.

T. Zhang and K. K. Parhi. Joint code and decoder design for implementation-oriented (3;k)-regular ldpc codes. *in Proc. Asilomar Conference on Signals, Systems and Computers*, 2:1232–1236, Nov. 2001,.