Development of System for Teaching Turbo Code Forward Error Correction Techniques

Shuai Shi

JUNE 2007

TITLE: Development of System for Teaching Turbo Code Forward Error Correction Techniques

AUTHOR: Shuai Shi

Dissertation submitted in compliance with the requirements for the Master's Degree in Technology in the Department of Electronic Engineering at Durban University of Technology.

DATE OF SUBMISSION: June 2007

I hereby declare that this dissertation represents my own work both in conception and execution.

_____

Shuai Shi, DURBAN, 2007-06-01

Approved for final submission

_____

C M McLeod, Supervisor

Summary

The objective was to develop a turbo code demonstration system for educational use. The aim was to build a system that would execute rapidly and produce a graphical display exemplifying the power of turbo codes and showing the effects of parameter variation.

The initial approach used an encoder, channel model and decoder all implemented on an FPGA development board controlled by a graphical user interface on a PC. This first-order concept proved to have limited utility due to the extensive resource requirements of the chosen algorithm (SOVA).

In order to explore the issues of making turbo-code simulation easily accessible to the student, a well-known Matlab command-line turbo code simulator was translated to C and provided with a graphical user interface. This allows the code to be used without Matlab licensing and produces graphics that meet the needs of the educator. The time taken to produce the graphics was however excessive.

The limitations encountered in the FPGA design and the speed limitations of the PC simulation precluded meaningful testing with student subjects.

The dissertation concludes by describing an FPGA design approach that would accommodate the complex soft-output decoding algorithms whilst providing satisfactory speed of execution.

Acknowledgments

I would like to express gratitude to my supervisors Dr Meredith McLeod and Prof Hong Jun Xu for their generous support and guidance. Kind assistance with the rough edges of my written English is also acknowledged.

I wish to acknowledge the major influence of Dr Yu Fei Wu's well-known turbo-code simulation software which provided a strong point of reference for the current work.

The generous funding of this project through the channels provided by DUT is greatly appreciated.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

APP	*A Posteriori* Probability
AWGN	Additive White Gaussian Noise
BCJR	Bahl, Cocke, Jelinek and Raviv Algorithm
BER	Bit Error Rate
CLB	Configuration Logic Block
FEC	Forward Error Correction
FIFO	First-Input, First-Output
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
LLR	Log-Likelihood Ratio
LUT	Look-Up Table
MAP	Maximum *a posteriori*
ML	Maximum Likelihood
RSC	Recursive Systematic Convolutional [code]
SNR	Signal-to-Noise Ratio
SOVA	Soft Output Viterbi Algorithm
VA	Viterbi Algorithm
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

Chapter 1. Introduction

1.1 Objective

It has been more than a decade since turbo code was invented. Turbo code has been applied widely in industry [36] [41] [44], and given priority attention by researchers [6] [7] [21] [37] [43] [32] .

Electronic Communication 4 (ECOM4) is a subject for final-year bachelor of technology (BTech) students in electrical engineering at the Durban University of Technology. It is proposed to include this topic in ECOM4. Turbo decoding algorithms will not be presented in detail because they are beyond the scope of this course, but turbo code principles and the critical parameters which affect the performance of turbo code will be introduced.

It was decided that for educational purposes a turbo code demonstration system providing facilities for adjusting coding parameters and signal-to-noise ratio should be built to supplement classroom instruction.

1.2 Constraints

1.2.1 The Existing Methodology

The system should fit in with the current approach used in the communications laboratory of DUT where physical circuits with short-range transmitter, antenna

and receiver are commonly used to introduce systems that are then studied in greater detail in simulation using such facilities as Rohde & Schwarz's WinIQSIM [24] and advanced digital system (ADS) [12]. This suggests an approach that includes hardware functionality as well as simulation.

1.2.2 Model Scenario

It was initially decided to concentrate on simulating a phase-modulated point-to-point (PTP) radio transmitter-receiver link. The encoder and decoder would be modelled using a field programmable gate array (FPGA). A desktop computer would be used to provide a data source, model the channel, transmitter and receiver, and evaluate performance. It was decided that the project would concentrate on the channel coding aspects and source coding would not form part of this study.

In the course of developing the hardware (FPGA) system, design constraints were encountered which limited the functionality of the originally proposed system. This led to the inclusion of a purely software solution. It was decided that any software required to support this educational system should ideally be free of copyright restrictions

1.3 The Approach

1.3.1 Educational

As described in [42] mastery learning based on the simple precepts of outcome based education (OBE) [5] is used. This involves the formulation of a number of observable skills (objectives), grouped under a general outcome, that are structured and communicated to the student. When applied to practical work in the laboratory 100% performance is required: all specified actions must be carried out completely and satisfactorily. This requires the actions and criteria to be accurately described and included in the laboratory manual thus allowing rapid assessment of the performance as *achieved* or *not yet achieved* as defined by [23]. This is an effective framework for conveying elementary performance requirements to a student.

A graphical user interface (GUI) is seen as the most convenient way of presenting the student with all the available features of the system.

1.3.2 Technical

Two methods were applied to this task: an FPGA implementation including noise simulation on the chip, and a total software approach supported on a Windows PC. The PC approach is closely based on [30] which also provided the basic structure for the FPGA design.

Control of the two simulators was then integrated into a suitable GUI to allow simulator configuration, parameter setting and performance reporting.

1.3.3 Criteria

- Suitable for Classroom Use

The completed system must be suitable for direct adoption into the course Electronic Communication 4 at the Department of Electronic Engineering at the Durban University of Technology.

- Conveys Evidence of Turbo Code Effectiveness

The system will provide a clear demonstration of the capability of turbo code structures.

- Permits Experimentation

The system will allow experimental adjustment of critical parameters.

1.4 Outline of Dissertation

Chapters 2 and 3 describe the selection of suitable theoretical encoder and decoder structures for the project. Chapter 4 describes the practical aspects of the implementation. The resulting systems are evaluated in Chapter 5 with conclusions drawn in Chapter 6.

Chapter 2. Choice of Encoder

2.1 Encoder Basics

This section presents a theoretical model largely influenced by [30] and drawing upon [9] [10] [27] and [16]. This will provide a point of reference for the discussion that follows.

2.1.1 Recursive Systematic Convolutional Encoder

The turbo encoder concatenates two recursive systematic convolutional (RSC) codes, hence analysis of the encoder should begin with an analysis of an RSC code. The RSC encoder is therefore a key component of the turbo encoder which is described in [10]. Every turbo encoder needs at least two of these.

The RSC encoder is a variation of the non-systematic convolutional (NSC) encoder which is represented in Figure 2.1.

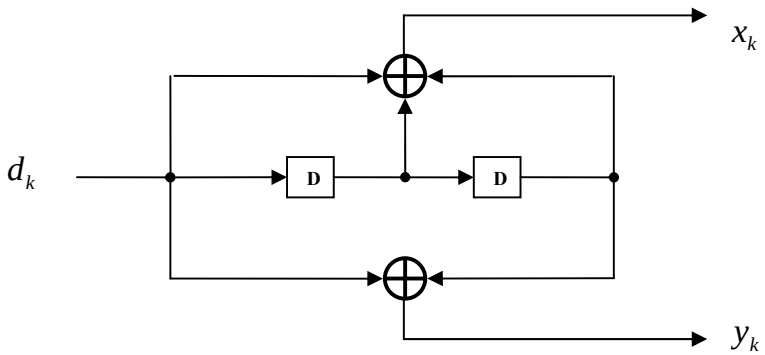


Figure 2.1: Non-systematic convolutional encoder adapted from [27]

Figure 2.1 shows an NSC encoder having a code rate of $1/2$, constraint length of $K = 3$ and memory size $M = K - 1 = 2$. Assume that the input of NSC encoder at time k is d_k and the corresponding output is the codeword (x_k, y_k). Also define g_{1i} and g_{2i} as two encoder generators of NSC. The mathematics can be expressed as follows [10]:

$$x_k = \sum_{i=0}^{K-1} g_{1i} d_{k-i} \quad \text{modulo - 2} \tag{2.1}$$

$$y_k = \sum_{i=0}^{K-1} g_{2i} d_{k-i} \quad \text{modulo - 2} \tag{2.2}$$

Figure 2.2 shows that the RSC encoder can be simply obtained by adding a feedback loop to the NSC encoder and setting the outputs x_k equal to d_k.

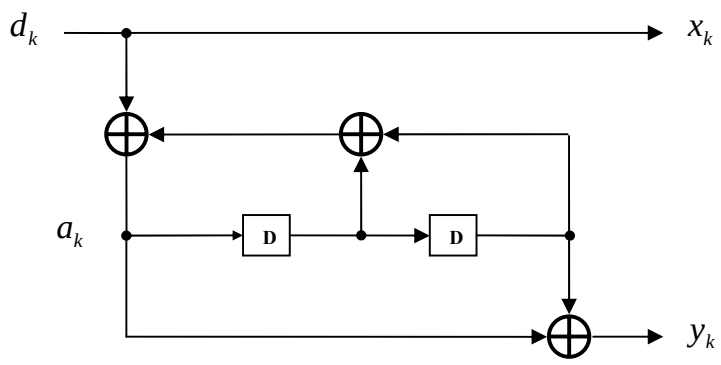


Figure 2.2: Recursive systematic convolutional encoder adapted from [31][27].

In the light of the structure, the shift register input is no longer the bit d_k as in the NSC encoder. Substitute by modulo-2 adding of d_k and the recursive value from memory, using a_k to express this sum. The a_k value can be calculated using the equation below.

$$a_k = d_k + \sum_{i=1}^{K-1} g_{1i} a_{k-i} \qquad \text{modulo - 2} \qquad (2.3)$$

The value of y_k can therefore be obtained by substituting a_k for d_k in Equation (2.2).

The state diagram shows the state knowledge of an RSC encoder. The state information of the coder is stored in the m-bit shift register. Figure 2.3 shows the state diagram of the encoder in Figure 2.2.

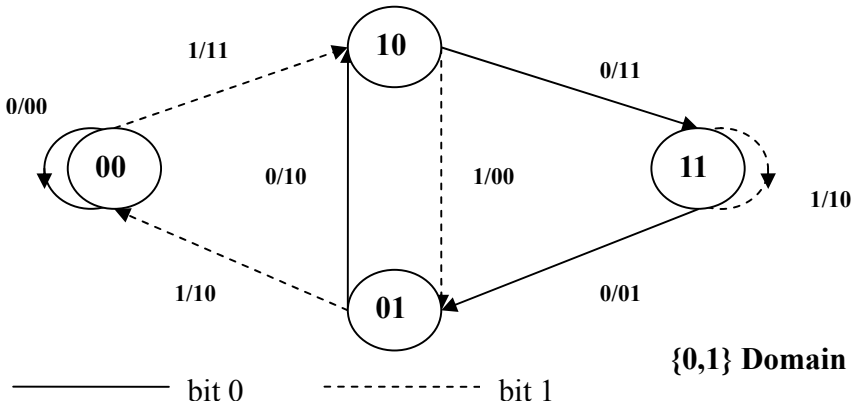


Figure 2.3: State diagram of the RSC component adapted from [9]

The RSC can be represented in other ways as well. A trellis diagram is convenient.

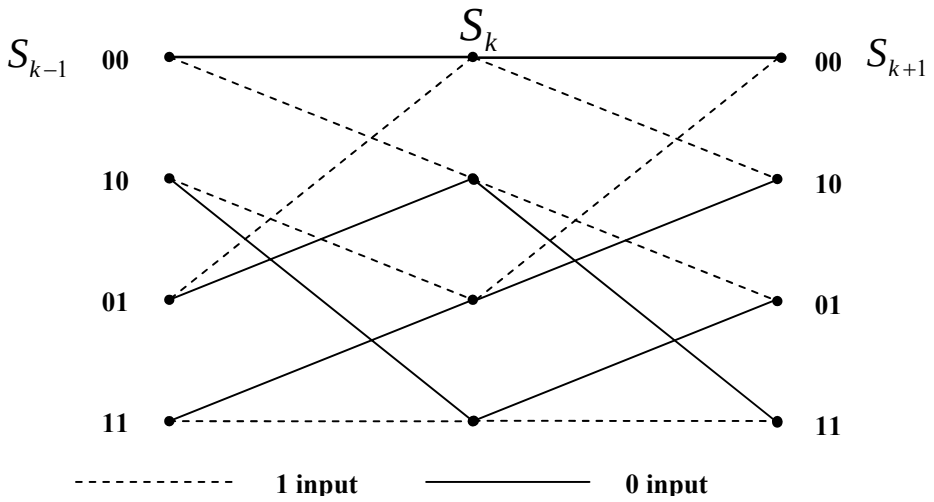


Figure 2.4: Trellis structure for the RSC code adapted from [16].

Figure 2.4 shows the trellis diagram of the $K = 3$ RSC code which is one of the component codes of the encoder. For this $K = 3$ code there are four encoder states and for each encoder state two state transitions are possible which depend on the value of the input bit. One of these transitions is associated with the input bit of 1 shown as a broken line, while the other transition corresponds to the input bit of 0 shown as a continuous line. It can be seen from Figure 2.4 that if the previous state S_{k-1} and the present state S_k are given, then the value of the input bit d_k which caused the transition between these two states, can also be obtained.

2.1.2 Encoder Structure

Figure 2.5 shows an example of a simple encoder structure. In the figure, the turbo encoder uses two RSC encoders in parallel, each of constraint length 3. The generator function {7,5} is an octal representation of the taps that exist on the RSC encoder. The information bits are scrambled by an interleaver before entering the second encoder. When the switch C is used to select the original data bit and the two parity bits the code rate is 1/3. In the other position, the switch C omits every second parity bit which makes the overall code rate 1/2. This omission is known as "puncturing" and is described in section 2.1.7. The puncturing structure is shown in Figure 2.9.

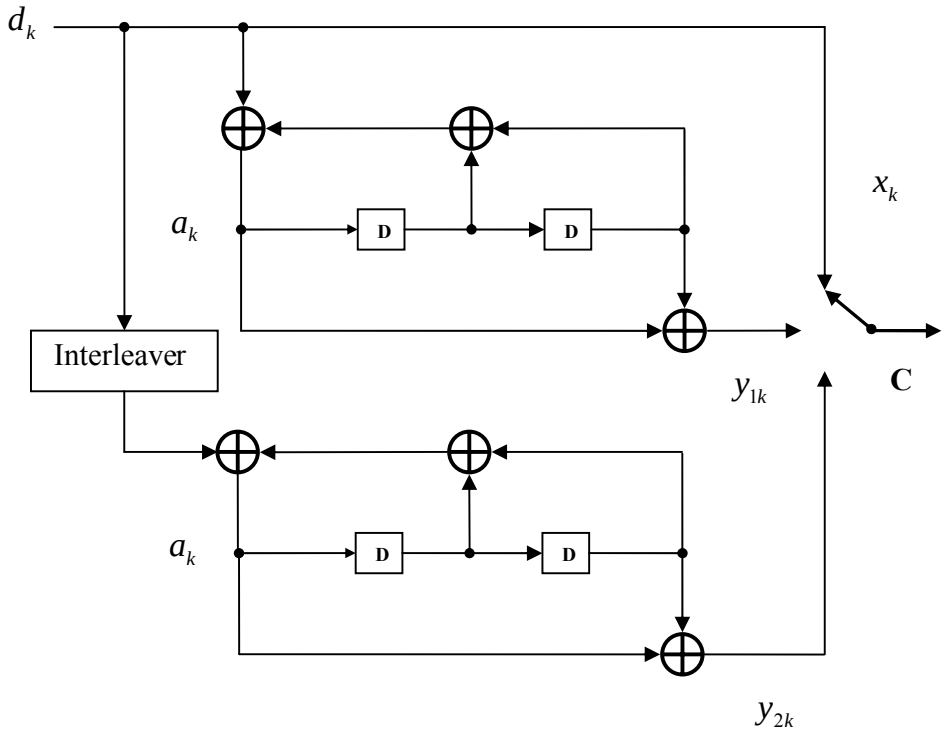


Figure 2.5: Turbo encoder adapted from [31] [39].

2.1.3 Trellis Termination Method.

Before the data sequence is applied to the encoder, the state of the encoder is initialized to zeros. After processing a frame there is no guarantee that the state is still zeros [7]. Commonly, for a conventional convolutional encoder, after the input information sequence has been processed, m zero bits are inserted into the encoder to ensure that the final state is zero [11]. Unfortunately this does not suit the RSC encoder for two reasons discussed below.

Firstly, the additional termination bits for the RSC encoder rely on the state of the encoder which makes it impossible to predict them. Figure 2.6 shows a simple strategy to solve the first problem which has been proposed by [31] [32] [7].

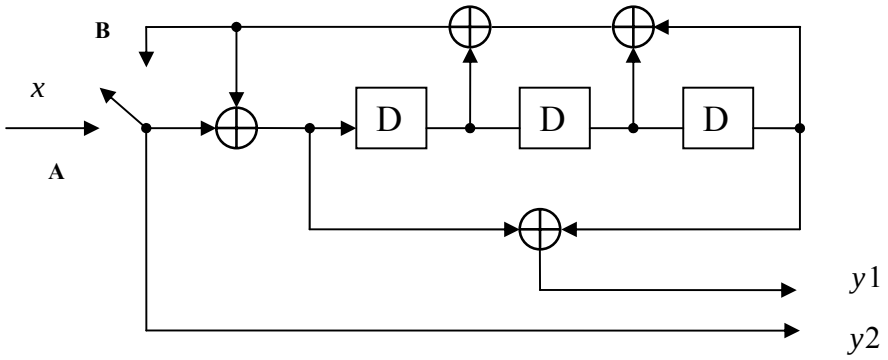


Figure 2.6: Trellis termination method adapted from [32]

When the switch is in position A, the input data sequence comes through and is processed by the encoder. When the switch is in position B, the termination bits are shifted out from the shift register for terminating the trellis.

The second problem arises with a turbo code encoder using more than one RSC encoder operating in parallel through an interleaver. Even if termination bits are found that are suitable for one encoder, they may not drive the other(s) to the all zero state [46]. The solution to this problem will be discussed in section 4.1.5.

2.1.4 The Need for an Interleaver

The distance between two codewords is equal to the weight of a codeword which is the modulo-2 sum of those two codewords [9]. For the purpose of designing practical turbo codes, the effective free distance of the codewords has to be maximal, see [9] page 492. One should avoid getting two simultaneous low-weight codewords from the two encoders [25].

An interleaver is a good solution for increasing the free distance [31]. Figure 2.7 shows an interleaver producing high-weight codewords.

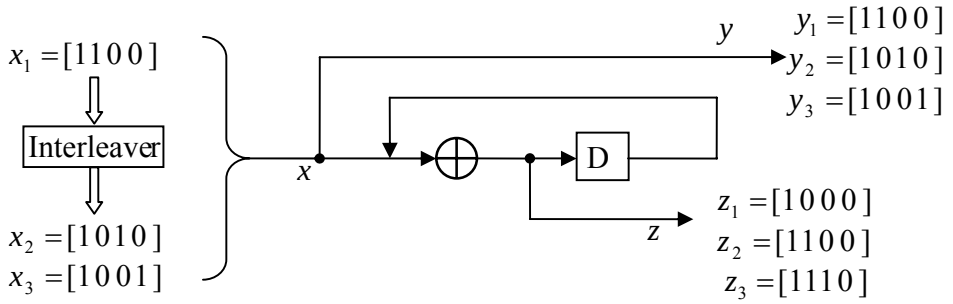


Figure 2.7: A simplified encoder adapted from [11].

The input x_1 is the original sequence, x_2 and x_3 are different permuted sequences sourced from x_1. Table 2.1 shows the resulting codewords and weights.

Table 2.1: Input and output sequences for encoder in Figure 2.7

	Input x_i	Output y_i	Output z_i	Codeword Weight
$i=1$	**1 1 0 0**	**1 1 0 0**	**1 0 0 0**	**3**
$i=2$	**1 0 1 0**	**1 0 1 0**	**1 1 0 0**	**4**
$i=3$	**1 0 0 1**	**1 0 0 1**	**1 1 1 0**	**5**

As can be seen from Table 2.1, the codeword weight is increased by utilising an interleaver to permute the x_1 sequence. The interleaver directly affects the distance properties of the code by avoiding low-weight codeword generation. By this means the performance of a turbo code can improve significantly.

2.1.5 Interleaver Overview

There are different kinds of interleavers applied to turbo codes. These are block, Berrou-Glavieux, pseudo-random and semi-random interleavers.

Block Interleaver

The block interleaver is called a *uniform interleaver* in [11] [10] [26]**.** It is the most commonly used interleaver because it has a simple structure compared to other types of interleavers. The block interleaver is defined by a matrix with x rows and y columns. Data is written row-wise (or column-wise) into the matrix and then read column-wise (or row-wise) out of the matrix.

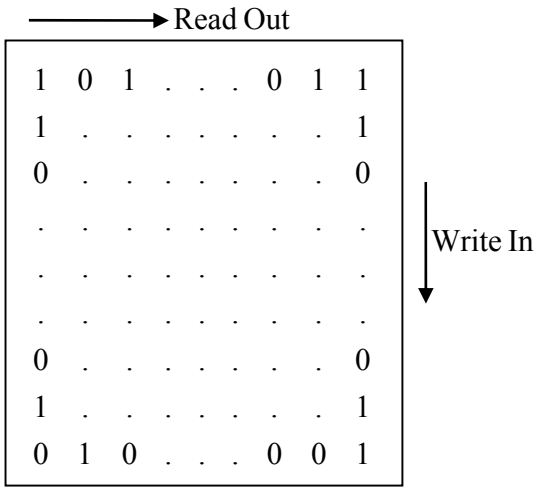


Figure 2.8: Block interleaver diagram

Berrou-Glavieux Interleavers

Berrou and Glavieux chose to use a non-uniform interleaver for their turbo code [27] [10] [26] in order to improve the weight of the codeword. For the writing process, i and j are indices of a matrix which is defined by the number of rows $R = 2^m$ (m is an integer) and the number of columns $C = 2^n$ (n is an integer). For the reading process π_i and π_j are defined as indices of the matrix.

$p(l)\cdots(l = 1 \sim 8)$ are prime numbers.

$$l = (i + j)\ mod\ 8 \tag{2.4}$$

$$\pi_i = (R/2 + 1)\cdot(i + j)\ mod\ R \tag{2.5}$$

$$\pi_j = [(p(l)\cdot(j + 1) - 1]\ mod\ C \tag{2.6}$$

Pseudo-Random Interleaver

The pseudo-random interleaver uses a fixed random index. The input data is sequenced using the fixed random index order to produce the output [34] [26] [11]. Table 2.2 is an illustration of a pseudo-random interleaver with size=5.

Table 2.2: Example of pseudo-random interleaver

Input index	Input data	Output index	Output data
1	1	5	1
2	0	3	1
3	1	1	1
4	0	4	0
5	1	2	0

Semi-Random Interleaver

The semi-random interleaver is a variation of the pseudo-random interleaver [34]. For each selected input index i, the output index π_i is chosen by comparing the input index i to the previously output index π_{i-1}. If $\left| i - \pi_{i-1} \right| \geq \sqrt{l/2}$ (l is length of data) then the current output index $\pi_i = i$ otherwise another index i is selected and the comparison is repeated. This process is repeated until all output indices are obtained [26] [11]. Table 2.3 illustrates a semi-random interleaver ($l = 8$).

Table 2.3: Example of semi-random interleaver

Input index i	Input data	Output index π_i	Output data
1	1	1	1
2	0	3	1
3	1	5	1
4	0	7	1
5	1	4	0
6	0	8	0
7	1	2	0
8	0	6	0

2.1.6 Interleaver Issues

The superior performance of turbo codes is achieved when the length of the interleaver is very large. For large size interleavers, most random interleavers perform well [13].

For silicon applications, the interleaving complexity must be decreased because several interleavers and de-interleavers have to be employed in a real time turbo-decoder [10]. For this purpose small interleavers should be selected for hardware implementation. However, the large interleaver size guarantees good turbo code performance. Smaller interleaver size results in poorer performance [14].

Using short interleavers, the performance of the turbo code with a random interleaver degrades rapidly [14]. For short interleavers, selection of the interleaver is critical for proper performance of the turbo code. For silicon applications especially, a short interleaver size which can achieve acceptable performance is needed [35].

2.1.7 Output Puncturing

The function of puncturing is the deletion of some bits from the codeword on the basis of a puncturing matrix which includes zeros and ones [40]. The zero represents an omitted bit. Puncturing is usually used to increase the rate of a given encoder [33].

In Figure 2.5, without puncturing, switch C is used to select the original data bit and the two parity bits to produce a code rate of 1/3. However, the puncture omits every second parity bit which makes the overall code rate 1/2.

From Figure 2.5 it can be seen that the information bit is output from x_k of RSC1, while the two parity outputs come from y_{1k} and y_{2k} of RSC1 and RSC2. These three outputs are multiplexed to form the codeword (x_k, y_{1k}, y_{2k}). Hence the overall code rate of the turbo encoder is determined at this point.

Figure 2.9 demonstrates how a rate 1/3 turbo code can be converted into a rate 1/2 [34] turbo code by using a puncturing matrix and Figure 2.10 shows how de-puncturing converts it back before decoding.

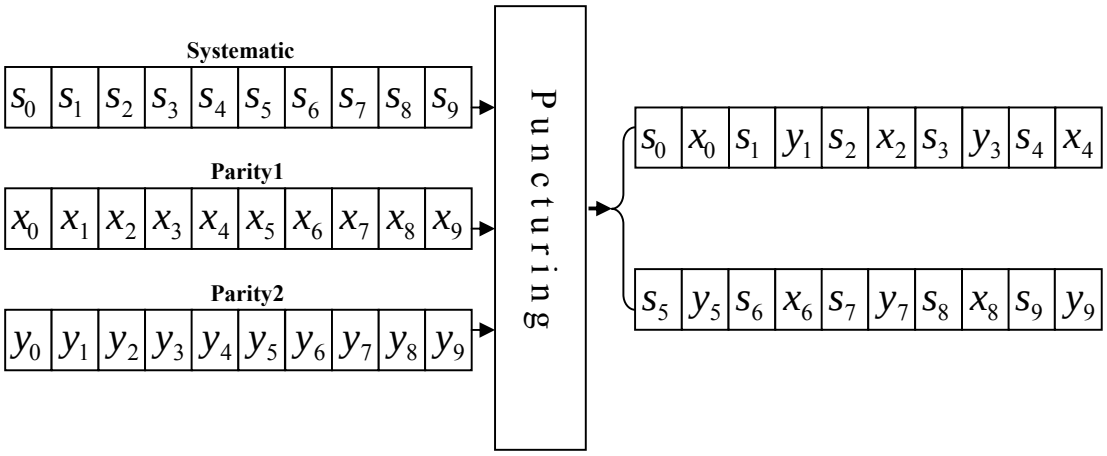


Figure 2.9: Puncturing diagram

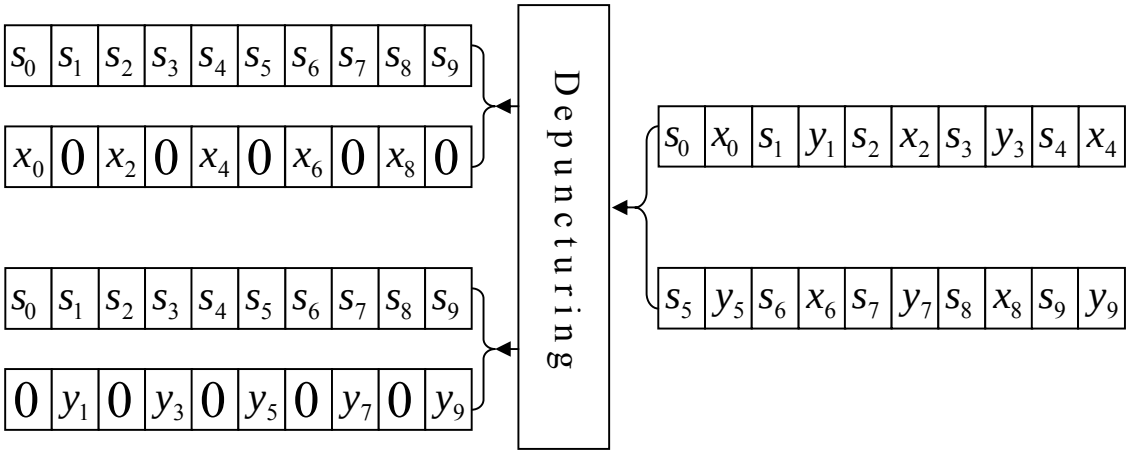


Figure 2.10: De-puncturing diagram

The same encoder may be used for different coding rates by using an appropriate puncturing scheme. Such a scheme would allow one silicon product to be used in different applications [10].

2.2 Encoder for FPGA Simulator

For the FPGA simulator, simplicity was the watchword in order to minimize resource utilization. For this reason the required minimum of two parallel RSC encoders was used. The constraint length was chosen as 3 for the same reason. The final configuration chosen was as depicted in Figure 2.5 with the same generator function {7,5}.

For the reasons given in section 2.1.6 it was decided to use a block interleaver.

2.3 Encoder for PC Simulation

The basis of the PC simulation in this project was a translation of the Matlab simulator of [30]. The code provided by Wu implements the encoder of Figure 2.5 but allows the user to adjust frame size and select puncturing on/off.

The Wu code uses a pseudo-random interleaver which is easily supported in the PC simulation.

Chapter 3. Choice of Decoder

"Iterative decoding of two-dimensional systematic convolutional codes has been termed turbo decoding" Hagenauer [20].

The objective of this chapter is to show the method of iterative decoding in a general framework in section 3.1.1 and present two soft-in-soft-out (SISO) algorithms in section 3.1.3 and section 3.1.4.

3.1 Decoder Basics

3.1.1 Structure

The general scheme of a basic decoder is shown in Figure 3.1. Two component decoders are linked by two interleavers and two de-interleavers. Each component decoder has three inputs: the systematic information, the parity information, and the information from the other component decoder. This information from the other decoder is referred to as *a priori* information. Both component decoders have to process both the inputs from the channel as well as *a priori* information from each other.

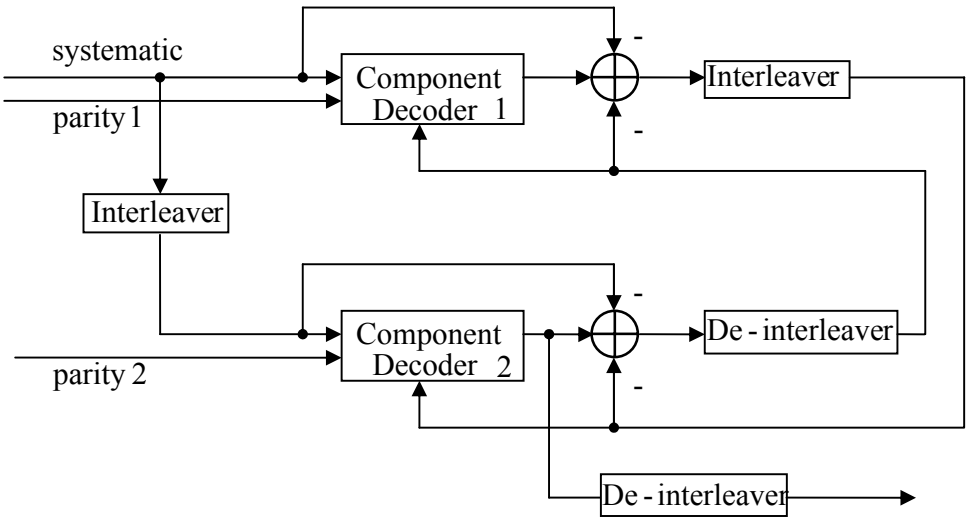


Figure 3.1: Turbo decoder schematic adapted from Hanzo *et al* page 110 [16]

The two most commonly used algorithms for the component decoders are the maximum *a posteriori* (MAP) algorithm and the soft output Viterbi algorithm (SOVA) which are described in sections 3.1.3 and 3.1.4 respectively.

3.1.2 Decoding Algorithm

Before presenting the turbo code algorithm, the foundation concept of log-likelihood ratio (LLR) should be discussed. This concept is described in [9] [8] upon which the following review is based.

Bayes' Theorem in Communication Channel

Bayes' theorem is a definition of the relationship between the conditional and joint probability of random variables. With hypothesis variables A and B the theorem can be expressed as Equations (3.1) and (3.2).

$$P(A|B)P(B) = P(B|A)P(A) = P(A,B) \tag{3.1}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.2}$$

To use this theorem for a communications channel, we need the form of the theorem as expressed in Equations (3.3) and (3.4). From Equation (3.2):

$$P(d=i|x) = \frac{p(x|d=i)P(d=i)}{p(x)} \qquad (i=1,\ldots,\mathrm{M}) \tag{3.3}$$

$$p(x) = \sum_{i=1}^{\mathrm{M}} p(x|d=i)P(d=i) \tag{3.4}$$

Where $d=i$ represents data d belonging to the ith signal class from a set of M classes and $P(d=i|x)$ called the *a posteriori* probability and $p(d=i)$ called the *a priori* probability. Also $p(x|d=i)$ represents the probability density function (pdf). The pdf of the received signal x is $p(x)$.

Binary Likelihood Ratio

Commonly the binary logical elements 1 and 0 transmitted on an AWGN channel can be represented as electronic signal $+1$ and -1. This is the classical

two-signal class case. Here $P(d = i \mid x)$ is the *a posteriori* probability, and $d = \pm 1$ represents data d belonging to the 2 signal classes from a pair of $(-1,+1)$ classes.

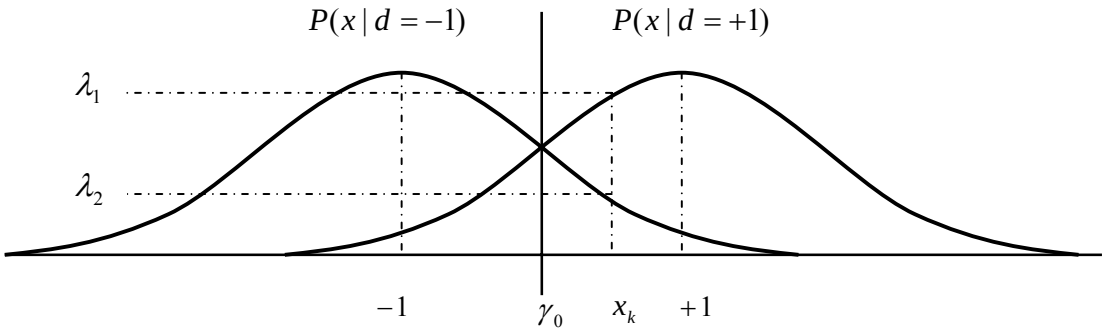


Figure 3.2: Likelihood functions adapted from [9]

$p(x \mid d = +1)$ is the pdf of random variable x when $d = +1$ is transmitted,

$p(x \mid d = -1)$ is the pdf of random variable x when $d = -1$ is transmitted.

When arbitrary value x_k is detected, two likelihood values λ_1 and λ_2 are gained. Maximum likelihood is obtained from comparison of the two likelihood values λ_1 and λ_2. If $\lambda_1 > \lambda_2$, $d = +1$ is transmitted to the channel, otherwise $d = -1$ is transmitted to the channel.

$$p(d = +1 \mid x) \underset{H_2}{\overset{H_1}{\gtrless}} p(d = -1 \mid x) \tag{3.5}$$

H_1 denoted $d = +1$ and H_2 denoted $d = -1$ obey Bayes' theorem. Equation (3.5) can be changed as follows:

$$p(x \mid d = +1)p(d = +1) \underset{H_2}{\overset{H_1}{\gtrless}} p(x \mid d = -1)p(d = -1) \tag{3.6}$$

and

$$\frac{p(x \mid d=+1)\,p(d=+1)}{p(x \mid d=-1)\,p(d=-1)} \underset{H_2}{\overset{H_1}{\gtrless}} 1 \tag{3.7}$$

Log-Likelihood Ratio

Now, taking the logarithm on right sides of Equation (3.7)

$$L(d \mid x) = \log\left[\frac{p(x \mid d=+1)P(d=+1)}{p(x \mid d=-1)P(d=-1)}\right] \tag{3.8}$$

By property of logarithm, Equation (3.9) is obtained.

$$L(d \mid x) = \log\left[\frac{p(x \mid d=+1)}{p(x \mid d=-1)}\right] + \log\left[\frac{P(d=+1)}{P(d=-1)}\right] \tag{3.9}$$

or

$$L(d \mid x) = L(x \mid d) + L(d) \tag{3.10}$$

The *a posteriori* probability of the channel output x condition on $d = \pm 1$ is $L(d \mid x)$. $L(d)$ is the *a priori* LLR of the data bit d. Equation (3.11) can be obtained by simplifying Equation (3.10).

$$L'(\hat{d}) = L_c(x) + L(d) \tag{3.11}$$

Here, the notation $L_c(x)$ denotes the received sequence from the channel. This received sequence $L_c(x)$ includes the transmitted systematic bits and the parity bits from the encoder. Now, turbo decoding will import some extra knowledge from the decoding process in order to make a more refined decision. This extra knowledge is called *extrinsic LLR.*

$$L(\hat{d}) = L'(\hat{d}) + L_e(\hat{d}) \tag{3.12}$$

Taking Equation (3.11) in Equation (3.12)

$$L(\hat{d}) = L_c(x) + L(d) + L_e(\hat{d}) \tag{3.13}$$

Equation (3.13) shows that the *a posteriori* information $L(\hat{d})$ of a systematic decoder includes three LLR elements: received channel measurement, *a priori* knowledge of the data d_k, and an extrinsic LLR $L_e(\hat{d})$ coming from the decoder process. This soft output $L(\hat{d})$ is the reliability value that supports the soft decision for decoding. The sign and magnitude of $L(\hat{d})$ denote the hard decision and the confidence of this decision respectively. If the sign of *extrinsic* value $L_e(\hat{d})$ is the same as the sign of $L_c(x) + L(d)$, the confidence of decision will be improved and vice versa.

Three terms have been mentioned frequently above. Definitions adapted from [16] are collected below for reference.

- *A priori* information is the information related to a bit d_k at the receiver before commencement of the decoding process.
- *Extrinsic* information is generated in the decoding process based on the received sequence and the *a priori* information.
- *A posteriori* information is the information that the decoder generates by considering all available sources of information correlated with d_k.

3.1.3 Maximum *a posteriori* Algorithm

The maximum *a posteriori* (MAP) algorithm was proposed by Bahl, Cocke, Jelinek and Raviv [52] in 1974. Some authors also call the MAP algorithm the *BCJR algorithm* after the originators. The MAP decoding algorithm has received wide attention in the literature. The following explanation has been adapted from [9] [10] [16] [25] [26] and [27].

Assume that at time k, an RSC encoder is in state S_k. If the encoder has v memory cells, the encoder state S_k at time k is represented by Equation (3.14).

$$S_k = \sum_{i=0}^{v-1} 2^i s_i \tag{3.14}$$

Suppose that the information bit sequence $\{d_k\}$ contains N independent bits d_k which take values 0 and 1 with equal probability. Assume the encoder initial state S_0 and the final state S_N are both equal to zero. In practice this can be achieved by using the last v bits to drive the encoder to state zero.

The outputs of the encoder are the systematic bits d_k and the parity bits Y_k. The encoder outputs at time k are converted into ± 1 by the Equation (3.15).

$$\left.\begin{aligned} a_k &= 2d_k - 1 \\ b_k &= 2Y_k - 1 \end{aligned}\right\} \tag{3.15}$$

These outputs are applied to a suitable modulator and passed through an AWGN channel. At the receiver, if the log likelihood ratio $L(d_k)$ is based on the unconditional probabilities $P(d_k = \pm 1)$, the LLR $L(d_k)$ can be defined as:

$$L(d_k) = \log \frac{P(d_k = +1)}{P(d_k = -1)} \tag{3.16}$$

Assume the LLR $L(d_k)$ is known, the probability of $d_k = +1$ and $d_k = -1$ can be calculated from Equation (3.18) to Equation (3.19).

Take the exponent on both side of Equation (3.16).

$$e^{L(d_k)} = \frac{P(d_k = +1)}{P(d_k = -1)} = \frac{P(d_k = +1)}{1 - P(d_k = +1)} \tag{3.17}$$

Equation (3.17) can be written as:

$$P(d_k = +1) = \frac{e^{L(d_k)}}{1 + e^{L(d_k)}} = \frac{1}{1 + e^{-L(d_k)}} \tag{3.18}$$

Similarly:

$$P(d_k = -1) = \frac{1}{1 + e^{L(d_k)}} = \frac{e^{-L(d_k)}}{1 + e^{-L(d_k)}} \tag{3.19}$$

The transmitted symbols（a_k, b_k) in Equation (3.15) are represented as C_k at time k. The transmitted sequence is given below.

$$C_1^N = (c_1, \cdots, c_k, \cdots, c_N) \tag{3.20}$$

Because the AWGN is a discrete Gaussian memoryless channel, the output sequence is defined as:

$$R_1^N = (R_1, \cdots, R_k, \cdots, R_N) \tag{3.21}$$

The APP of a decoded bit d_k can be derived from the joint probability definition.

$$\lambda_k^i(m) = P(d_k = i, S_k = m \mid R_1^N) \tag{3.22}$$

Here, $i = 0, 1$; $m = 0, 1, \cdots, 2^{v-1}$. The APP of a decoded data bit d_k is equal to

$$P(d_k = i \mid R_1^N) = \sum_m \lambda_k^i(m) \tag{3.23}$$

According to Equations (3.16) and (3.23) the $L(d_k)$ associated with a decoded bit d_k can be written as:

$$L(d_k) = \log \frac{\sum_m \lambda_k^1(m)}{\sum_m \lambda_k^0(m)} \tag{3.24}$$

The $L(d_k)$ is the soft output of the MAP decoder. This can be used as an input to a second decoder in a parallel concatenated scheme. Finally the decoder can make a hard decision by comparing $L(d_k)$ with zero. If $L(d_k) \geq 0$, the decoded bit is 1, while if $L(d_k) < 0$, the decoded bit is 0.

According to Equation (3.25).

$$P(A \mid B) = \frac{P(A, B)}{P(B)} \tag{3.25}$$

We can write Equation (3.22) as:

$$\lambda_k^i(m) = P(d_k = i, S_k = m \mid R_1^N) = \frac{P(d_k = i, S_k = m, R_1^N)}{P(R_1^N)} \tag{3.26}$$

Equation (3.28) can be obtained using Equation (3.27).

$$P(A) = \sum_B P(A, B) \tag{3.27}$$

$$\lambda_k^i(m) = \frac{P(d_k = i, S_k = m, R_1^N)}{P(R_1^N)} = \frac{\sum_{m'} P(d_k = i, S_k = m, S_{k-1} = m', R_1^N)}{P(R_1^N)} \quad (3.28)$$

The received sequence can be split into three parts. The first part only contains the past observation before time k, second part contains the present observation at time k and third part is the future observation after time k, ie, $R_1^N = (R_1^{k-1}, R_k, R_{k+1}^N)$. Taking $R_1^N = (R_1^{k-1}, R_k, R_{k+1}^N)$ and Equation (3.28) into Equation (3.24), the LLR $L(d_k)$ can be written as:

$$L(d_k) = \log \frac{\sum_m \sum_{m'} P(d_k = 1, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k=1}^N)}{\sum_m \sum_{m'} P(d_k = 0, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k=1}^N)} \quad (3.29)$$

Applying Equation (3.25):

$$\begin{aligned} & P(d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k+1}^N) \\ = \; & P(R_{k+1}^N \mid d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) \cdot P(d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) \end{aligned} \quad (3.30)$$

If the occurrence of A does not depend on the occurrence of B, or events A and B are statistically independent, the conditional probability $P(A \mid B)$ and the joint probability $P(A, B)$ can be written respectively as:

$$P(A \mid B) = P(A) \quad (3.31)$$

$$P(A, B) = P(A) \cdot P(B) \quad (3.32)$$

Re-examining Equation (3.30) in this light it is seen that the received sequence R_{k+1}^N will depend only on the present state $S_k = m$ and not on the previous state $S_{k-1} = m'$. This also applies to the present and previous received channel sequences R_1^{k-1} and R_k. So Equation (3.30) can be simplified as follows:

$$
\begin{aligned}
& P(d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k+1}^N) \\
= \; & P(R_{k+1}^N \mid S_k = m) \cdot P(d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k)
\end{aligned} \tag{3.33}
$$

Rewriting Equation (3.25):

$$
P(A, B) = P(B \mid A) \cdot P(A) \tag{3.34}
$$

Applying Equations (3.31) and (3.34) to Equation (3.33):

$$
\begin{aligned}
& P(d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k+1}^N) \\
= \; & P(R_{k+1}^N \mid S_k = m) \cdot P(d_k = i, S_k = m, R_k \mid S_{k-1} = m', R_1^{k-1}) \\
& \cdot P(S_{k-1} = m', R_1^{k-1}) \\
= \; & P(R_{k+1}^N \mid S_k = m) \cdot P(d_k = i, S_k = m, R_k \mid S_{k-1} = m') \\
& \cdot P(S_{k-1} = m', R_1^{k-1})
\end{aligned} \tag{3.35}
$$

Substituting Equation (3.35) into Equation (3.29)

$$
\begin{aligned}
& L(d_k) \\
= \; & \log \frac{\sum_m \sum_{m'} P(d_k = 1, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k=1}^N)}{\sum_m \sum_{m'} P(d_k = 0, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k=1}^N)} \\
= \; & \log \frac{\sum_m \sum_{m'} P(R_{k+1}^N \mid S_k = m) \cdot P(d_k = 1, S_k = m, R_k \mid S_{k-1} = m') \cdot P(S_{k-1} = m', R_1^{k-1})}{\sum_m \sum_{m'} P(R_{k+1}^N \mid S_k = m) \cdot P(d_k = 0, S_k = m, R_k \mid S_{k-1} = m') \cdot P(S_{k-1} = m', R_1^{k-1})}
\end{aligned} \tag{3.36}
$$

In order to compute Equation (3.36), the probability functions $\alpha_{k-1}(m')$, $\beta_k(m)$, $\gamma_i(R_k, m', m)$ are defined as follows:

$$
\alpha_{k-1}(m') = P(S_{k-1} = m', R_1^{k-1}) \tag{3.37}
$$

$$
\beta_k(m) = P(R_{k+1}^N \mid S_k = m) \tag{3.38}
$$

$$
\gamma_i(R_k, m', m) = P(d_k = i, S_k = m, R_k \mid S_{k-1} = m') \tag{3.39}
$$

The $\gamma_i(R_k, m', m)$ can be expressed as:

$$\gamma_k(m',m) = \sum_{i=0}^{1} \gamma_i(R_k, m', m) \tag{3.40}$$

Substituting Equations (3.37) (3.38) and (3.39) into Equation (3.35) which is split into a product of 3 terms.

$$\begin{aligned} &P(d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k+1}^N) \\ &= \beta_k(m) \cdot \gamma_i(R_k, m', m) \cdot \alpha_{k-1}(m') \end{aligned} \tag{3.41}$$

By substituting Equation (3.41) into Equation (3.29).

$$L(d_k) = \log \frac{\sum_m \sum_{m'} \beta_k(m) \cdot \gamma_{+1}(R_k, m', m) \cdot \alpha_{k-1}(m')}{\sum_m \sum_{m'} \beta_k(m) \cdot \gamma_{-1}(R_k, m', m) \cdot \alpha_{k-1}(m')} \tag{3.42}$$

Below is a description of how to calculate the values of $\alpha_{k-1}(m')$, $\beta_k(m)$ and $\gamma_k(m', m)$.

Forward Recursive Calculation of the $\alpha_k(m)$ Values

According to the definition of $\alpha_{k-1}(m')$ we can write $\alpha_k(m)$ as:

$$\begin{aligned} \alpha_k(m) &= P(S_k = m, R_1^k) \\ &= P(S_k = m, R_1^{k-1}, R_k) \\ &= \sum_{m'} P(S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) \end{aligned} \tag{3.43}$$

In Equation (3.43) the probability $P(S_k = m, R_1^k)$ is split into the sum of joint probabilities $P(S_k = m, S_{k-1} = m', R_1^k)$ over all possible previous states $S_{k-1} = m'$.

Assuming that the channels are memoryless, Equation (3.43) can be obtained using Bayes' rule:

$$\begin{aligned}\alpha_k(m) &= \sum_{m'} P(S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) \\ &= \sum_{m'} P(S_k = m, R_k \mid S_{k-1} = m', R_1^{k-1}) \cdot P(S_{k-1} = m', R_1^{k-1}) \\ &= \sum_{m'} P(S_k = m, R_k \mid S_{k-1} = m') \cdot P(S_{k-1} = m', R_1^{k-1}) \\ &= \sum_{m'} \gamma_k(m', m) \cdot \alpha_{k-1}(m') \end{aligned} \tag{3.44}$$

Once the $\gamma_k(m', m)$ values are given, the $\alpha_k(m)$ values can be calculated.

Backward Recursive Calculation of the $\beta_k(m)$ Values

From the definition of $\beta_k(m)$, the values of $\beta_{k-1}(m')$ can be calculated from the Equation below:

$$\begin{aligned}\beta_{k-1}(m') &= P(R_k^N \mid S_{k-1} = m') \\ &= \sum P(R_k^N, S_k = m \mid S_{k-1} = m') \\ &= \sum_{m} P(R_k, R_{k+1}^N, S_k = m \mid S_{k-1} = m') \end{aligned} \tag{3.45}$$

Using Bayes' theorem in Equation (3.46)

$$\begin{aligned}P(A, B, C \mid D) &= \frac{P(A, B, C, D)}{P(D)} \\ &= \frac{P(A \mid B, C, D) \cdot P(B, C, D)}{P(D)} \\ &= P(A \mid B, C, D) P(B, C \mid D) \end{aligned} \tag{3.46}$$

Equation (3.45) can be written as

$$\begin{aligned}\beta_{k-1}(m') &= \sum P(R_k, R_{k+1}^N, S_k = m \mid S_{k-1} = m') \\ &= \sum_{m} P(R_{k+1}^N \mid R_k, S_k = m, S_{k-1} = m') \cdot P(R_k, S_k = m \mid S_{k-1} = m') \\ &= \sum_{m} P(R_{k+1}^N \mid S_k = m) \cdot P(R_k, S_k = m \mid S_{k-1} = m') \\ &= \sum_{m} \beta_k(m) \cdot \gamma_k(m', m) \end{aligned} \tag{3.47}$$

Once the values $\gamma_k(m', m)$ are given, the values $\beta_{k-1}(m')$ can be calculated from the values $\beta_k(m)$ using Equation (3.47).

Calculation of the $\gamma_k(m',m)$ Values

The $\gamma_k(m',m)$ values in equation (3.40) can be calculated using Equation (3.46).

$$\begin{aligned}&\gamma_k(m',m)\\&= P(S_k = m, R_k \mid S_{k-1} = m')\\&= P(R_k \mid S_k = m, S_{k-1} = m') \cdot P(S_k = m \mid S_{k-1} = m')\end{aligned} \tag{3.48}$$

The state transits from state $S_{k-1} = m'$ to state $S_k = m$ based on the input bit d_k.

$P(d_k)$ is the *a priori* probability of the input bit. Equation (3.49) can be obtained from Equation (3.18) and Equation (3.19) according to [6].

$$P(d_k = \pm 1) = \left(\frac{e^{-L(d_k)/2}}{1 + e^{-L(d_k)}} \right) \cdot e^{\pm L(d_k)/2} \tag{3.49}$$

Here, C_1 is defined as:

$$C_1 = \left(\frac{e^{-L(d_k)/2}}{1 + e^{-L(d_k)}} \right) \tag{3.50}$$

Equation (3.49) shows, that the probability of d_k depends only on the LLR $L(d_k)$, regardless of whether d_k equal to -1 or $+1$.

The term $P(R_k \mid S_k = m, S_{k-1} = m')$ in Equation (3.48) is equivalent to $P(R_k \mid x_k)$ because x_k is the transmitted codeword associated with the state transition.

Assuming the channel is memoryless, the $P(R_k \mid x_k)$ can be written as:

$$P(R_k \mid S_k = m, S_{k-1} = m') = P(R_k \mid x_k) = \prod_{i=1}^{n} P(R_{ki} \mid x_{ki}) \tag{3.51}$$

where x_{ki} is the individual bit of transmitted codeword x_k. R_{ki} is the symbol in received codeword R_k. Assuming that the transmitted bits x_{ki} have been processed by BPSK over a Gaussian channel, with transmitted symbols $+1$ or -1, $P(R_{ki} \mid x_{ki})$ can be written as:

$$P(R_{ki} \mid x_{ki}) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{E_b}{2\sigma^2} \left(R_{ki} - a x_{ki}\right)^2 \right) \tag{3.52}$$

Here, E_b is the transmitted energy per bit, σ^2 is the noise variance and a is the fading amplitude of the AWGN channel. Assuming the channel is non-fading then a=1.

Substituting Equation (3.52) into Equation (3.51):

$$\begin{aligned}
&P(R_k \mid x_k) \\
&= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{E_b}{2\sigma^2} \left(R_{ki} - a x_{ki}\right)^2 \right) \\
&= \frac{1}{(\sqrt{2\pi}\sigma)^n} \exp\left(-\frac{E_b}{2\sigma^2} \sum_{i=1}^{n} \left(R_{ki} - a x_{ki}\right)^2 \right) \\
&= \frac{1}{(\sqrt{2\pi}\sigma)^n} \exp\left(-\frac{E_b}{2\sigma^2} \sum_{i=1}^{n} \left(R_{ki}^2 - 2a x_{ki} R_{ki} + a^2 x_{ki}^2\right) \right) \\
&= \frac{1}{(\sqrt{2\pi}\sigma)^n} \exp\left(-\frac{E_b}{2\sigma^2} \sum_{i=1}^{n} \left(R_{ki}^2\right) \right) \cdot \exp\left(-\frac{E_b}{2\sigma^2} \sum_{i=1}^{n} \left(a^2 x_{ki}^2\right) \right) \\
&\quad \cdot \exp\left(\frac{E_b}{2\sigma^2} \sum_{i=1}^{n} \left(2a x_{ki} R_{ki}\right) \right)
\end{aligned} \tag{3.53}$$

Here C_2 is defined as:

$$C_2 = \frac{1}{(\sqrt{2\pi}\sigma)^n} \exp\left(-\frac{E_b}{2\sigma^2} \sum_{i=1}^{n} \left(R_{ki}^2\right) \right) \tag{3.54}$$

Equation (3.54) shows the C_2 value is only tied to the channel SNR and the magnitude of the received sequence R_k.

Here C_3 is defined as:

$$C_3 = \exp\left(-\frac{E_b}{2\sigma^2}\sum_{i=1}^{n}\left(a^2 x_{ki}^2\right)\right)$$
$$= \exp\left(-\frac{E_b}{2\sigma^2}a^2 n\right) \tag{3.55}$$

Equation (3.55) indicates that the C_3 value relies on the channel SNR and the fading amplitude. Hence $\gamma_k(m', m)$ can be written as:

$$\begin{aligned}\gamma_k(m', m) &= P(S_k = m, R_k \mid S_{k-1} = m') \\ &= P(R_k \mid S_k = m, S_{k-1} = m') \cdot P(S_k = m \mid S_{k-1} = m') \\ &= \frac{1}{(\sqrt{2\pi}\sigma)^n}\exp\left(-\frac{E_b}{2\sigma^2}\sum_{i=1}^{n}\left(R_{ki}^2\right)\right)\cdot\exp\left(-\frac{E_b}{2\sigma^2}\sum_{i=1}^{n}\left(a^2 x_{ki}^2\right)\right) \\ &\quad\cdot\exp\left(\frac{E_b}{2\sigma^2}\sum_{i=1}^{n}\left(2a x_{ki} R_{ki}\right)\right)\cdot\left(\frac{e^{-L(d_k)/2}}{1+e^{-L(d_k)}}\right)\cdot e^{\pm L(d_k)/2} \\ &= C_1 \cdot C_2 \cdot C_3 \cdot e^{\pm L(d_k)/2}\cdot\exp\left(\frac{E_b}{2\sigma^2}\sum_{i=1}^{n}\left(2a x_{ki} R_{ki}\right)\right) \\ &= C \cdot e^{\pm L(d_k)/2}\cdot\exp\left(\frac{E_b}{2\sigma^2}\sum_{i=1}^{n}\left(2a x_{ki} R_{ki}\right)\right) \\ &= C \cdot e^{\pm L(d_k)/2}\cdot\exp\left(\frac{L_c}{2}\sum_{i=1}^{n}\left(x_{ki} R_{ki}\right)\right)\end{aligned} \tag{3.56}$$

Where $C = C_1 \cdot C_2 \cdot C_3$

Actually C in Equation (3.56) can be considered as a constant because the sign of bit d_k and the transmitted codeword x_k do not impact the value of C. C can be deleted from Equation (3.56).

3.1.4 Soft Output Viterbi Algorithm

The Viterbi [decoding] algorithm (VA) was developed by Viterbi [15] in 1967. Since that time, it has been recognized as a preeminent solution for convolutional decoding. There are two limitations in the VA for iterative decoding [11], [16] and [17].

The first one is that the inner VA tends to produce bursts of errors correlated with the outer VA. This drawback is normally present for all concatenated code frames. The common solution is to use an interleaver between the inner and outer VA.

The second shortcoming is that the inner VA can't produce soft decisions (reliability values) thus preventing the outer VA from benefitting from iteration. To avoid the second drawback, the inner VA needs to output reliability information that can improve the performance.

When concatenated coding is considered using convolutional code as component encoder, the performance of the global decoder cannot be improved unless the inner decoder can provide soft decisions [18]. Unfortunately the VA is unable to provide a soft value to benefit the outer decoder. For practical purposes, the direct solution is to modify it so that it can produce a soft value. The modified Viterbi Algorithm named soft output Viterbi algorithm (SOVA) developed by

Hagenauer and Hoeher [17] in commonly used. The following is summarized from some source about SOVA [18] [17] [16] [19] [11].

Figure 3.2 shows the inputs and outputs of the SOVA decoder.

Figure 3.3: SOVA module

Here, the symbol $L(u)$ is specified as the *a priori* sequence that is produced and obtained from the preceding SOVA decoder and $L_c R$ is the received sequence from the channel.

In the first iteration there is no preceding SOVA decoder so $L(u)$ is initialized to an all zero sequence. The SOVA decoder produces output u' which is the estimated information sequence and $L(u')$ which is the associated log-likelihood ratio sequence. The calculation of $L(u')$ values is described below.

By using Bayes's theorem, the *a posteriori* probability can be written as:

$$P(s_k \mid R_1^k) = \frac{P(s_k, R_1^k)}{P(R_1^k)} \tag{3.57}$$

Here, s_k is the state sequence at stage k in the trellis. The sequence R_1^k is the received sequence from the AWGN channel. Since the probability of the received sequence R_1^k is constant through all paths and does not depend on state s_k, it can

be discarded. For this reason the branch metric should be defined so that maximizing the metric will maximize $P(s_k, R_1^k)$.

The probability $P(S_k = m, R_1^k)$ can be split into the sum of joint probabilities $P(S_k = m, S_{k-1} = m', R_1^k)$ over all possible previous states $S_{k-1} = m'$. So the term $P(s_k, R_1^k)$ in Equation (3.57) can be written as:

$$\begin{aligned} P(s_k, R_1^k) &= P(S_k = m, R_1^k) \\ &= P(S_k = m, R_1^{k-1}, R_k) \\ &= P(S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) \end{aligned} \tag{3.58}$$

Assuming that the channels are memoryless, Equation (3.59) can be obtained using Bayes' rule:

$$\begin{aligned} P(S_k = m, S_{k-1} = m', R_1^{k-1}, R_k) &= P(S_k = m, R_k \mid S_{k-1} = m', R_1^{k-1}) \cdot P(S_{k-1} = m', R_1^{k-1}) \\ &= P(S_k = m, R_k \mid S_{k-1} = m') \cdot P(S_{k-1} = m', R_1^{k-1}) \end{aligned} \tag{3.59}$$

Thus, $P(S_k = m, R_1^k)$ can be written as:

$$P(s_k, R_1^k) = P(s_{k-1}, R_1^{k-1}) \cdot P(S_k = m, R_k \mid S_{k-1} = m') \tag{3.60}$$

The defined metric for the path $path_k^m$ is M_k^m, and M_k^m can be obtained by applying the logarithm to Equation (3.60)

$$\begin{aligned} M_k^m &= \ln(P(s_k, R_1^k)) \\ &= \ln(P(s_{k-1}, R_1^{k-1}) \cdot P(S_k = m, R_k \mid S_{k-1} = m')) \\ &= \ln(P(s_{k-1}, R_1^{k-1})) + \ln(P(S_k = m, R_k \mid S_{k-1} = m')) \\ &= M_{k-1}^{m'} + \ln(P(S_k = m, R_k \mid S_{k-1} = m')) \end{aligned} \tag{3.61}$$

Now, apply Equation (3.62) to second term of last line of Equation (3.61)

$$P(\{A,B\}|C) = P(A|\{B,C\}) \cdot P(B|C) \tag{3.62}$$

$$P(S_k = m, R_k \mid S_{k-1} = m') = P(R_k \mid S_k = m, S_{k-1} = m') \cdot P(S_k = m \mid S_{k-1} = m') \tag{3.63}$$

The state transits from state $S_{k-1} = m'$ to state $S_k = m$ based on the input bit d_k.

$P(d_k)$ is the *a priori* probability of the input bit.

The equation (3.63) is equal to the the values of $\gamma_k(m', m)$, so taking the logarithm on both sides of equation (3.56) produces:

$$\begin{aligned} &\ln(p(S_k = m, R_k \mid \mathrm{S}_{k-1} = m')) \\ &= \ln\left(C \cdot e^{\pm L(d_k)/2} \cdot \exp\left(\frac{L_c}{2}\sum_{i-1}^{n}(x_{ki}R_{ki})\right)\right) \\ &= \hat{\mathrm{C}} + \frac{1}{2}.(\pm L(d_k)) + \frac{L_c}{2}\sum_{i=1}^{n}(x_{ki}R_{ki}) \end{aligned} \tag{3.64}$$

And as the term $\hat{C}$ is constant, it can be omitted and we can rewrite Equation (3.61) as:

$$M_k^m = M_{k-1}^{m'} + \frac{1}{2}.(\pm L(d_k)) + \frac{L_c}{2}\sum_{i=1}^{n}(x_{ki}R_{ki}) \tag{3.65}$$

So the metric is updated. The term $\pm L(d_k)$ is *a priori* information.

Any point in a binary trellis can only be reached along one of two path segments. Define these two paths as $path_k$ and $\hat{p}ath_k$ respectively. Its metrics are $M(path_k)$ and $M(\hat{p}ath_k)$. These two merged paths can be calculated using Equation (3.65). If $path_k$ is selected as the survivor because its metric is higher. So the path $\hat{p}ath_k$ with lower metric will be discarded.

We can define the metric difference as:

$$\Delta_k = M(path_k) - M(\hat{p}ath_k) \geq 0 \quad (3.66).$$

When $path_k$ is selected as the survivor path and $\hat{p}ath_k$ as discarded path, the probability of correct decision is defined as:

$$P\ (\text{correct}) = \frac{P(path_k)}{P(path_k) + P(\hat{p}ath_k)} \quad (3.67)$$

Rearrange Equation (3.66) as:

$$e^{\Delta_k} = M - M' \quad (3.68)$$

Taking Equation (3.68) into equation (3.67):

$$\begin{aligned} P\ (\text{correct}) &= \frac{e^{M}}{e^{M} + e^{M'}} \\ &= \frac{e^{M-M'}}{e^{M-M'} + 1} \\ &= \frac{e^{\Delta_k}}{1 + e^{\Delta_k}} \end{aligned} \quad (3.69)$$

So the soft value of this binary path decision can be written as:

$$\begin{aligned} L\ (\text{correct decision at } S_k = m) &= \frac{P\ (\text{correct decision at } S_k = m)}{1 - P\ (\text{correct decision at } S_k = m)} \\ &= \frac{\dfrac{e^{\Delta_k}}{1 + e^{\Delta_k}}}{1 - \dfrac{e^{\Delta_k}}{1 + e^{\Delta_k}}} = \frac{\dfrac{e^{\Delta_k}}{1 + e^{\Delta_k}}}{\dfrac{1}{1 + e^{\Delta_k}}} \\ &= e^{\Delta_k} \end{aligned} \quad (3.70)$$

After the maximum likelihood path has been determined, the reliability of each bit should be given. The SOVA shows all the surviving paths at a stage k from some point which is δ stage transitions before k in the trellis. The value of δ should be large enough so that all S survivor paths can be merged with sufficiently high probability. Usually the δ is set to be five times the constraint

length of the convolutional code. If the value of δ is higher, the soft decisions would be more accurate, however calculation complexity would be increased.

State transitions depend on different $d_k = -1$ or $d_k = +1$, so the algorithm can select the paths which merged with the ML path named discard path. Thus, when calculating the LLR of the bit d_k, the SOVA must take into account the probability of discard paths that merged with the ML path from stage k to stage $k+\delta$ in the trellis to find the incorrectly discarded path. This is done by considering the values of the metric difference $\Delta_k^{k+\delta}$ for all states S along the ML path from trellis stage k to stage $k+\delta$. It is shown by Hagenauer in [20] that this LLR can be approximated by:

$$L(d_k \mid R) \approx d_k \min_{\substack{i=k\cdots k+\delta \\ d_k \neq d_k'}} \Delta_i \tag{3.71}$$

Here, d_k is the value of the bit given by the ML path, and d_k' is the value of this bit for the path which merged with the ML path and was discarded. Thus Equation (3.71) is only evaluated for those paths merging with the ML path which would have given a different value with the bit d_k. The paths which merge with the ML path, but would have given the same value for d_k as the ML path, obviously do not affect the reliability of the decision of d_k.

3.2 Decoder for FPGA Simulator

The chosen structure is as shown in Figure 3.1. The only design decision is the choice of decoding algorithm.

MAP and SOVA algorithms could both be implemented on the FPGA development system but SOVA was selected for the following reasons.

Firstly, it is the least complex of all soft-in soft-out decoding algorithms. In the literature [21] Robertson said that SOVA is about half as complex as the MAP algorithm. Secondly, MAP uses logarithms in the decoding process. This was seen as adding unacceptable complexity to an FPGA implementation.

For the sake of simplicity only the SOVA algorithm was selected for FPGA implementation in this project.

3.3 Decoder for PC Simulation

The Wu Matlab simulator [30] also uses the scheme shown in Figure 3.1, but offers a choice of MAP or SOVA.

Although MAP has disadvantages from the calculation aspect, it outperforms the SOVA by 0.6 dB when bit-error probability is high [16] [21]. Considering the

educational purpose of this project, the choice of MAP or SOVA was retained in the translated code.

Chapter 4. Implementation

Chapters 2 and 3 introduced the encoder and decoder which were used in this implementation. This chapter presents an overview of the implementation. Section 4.1 deals with the FPGA-based encoder, decoder, noise model and on-fabric UART. Section 4.2 describes the PC simulation, and section 4.3 covers GUI development.

4.1 FPGA Implementation

The platform chosen was a Virtex-II Pro FPGA chip on a Xilinx ML-310 embedded system development board.

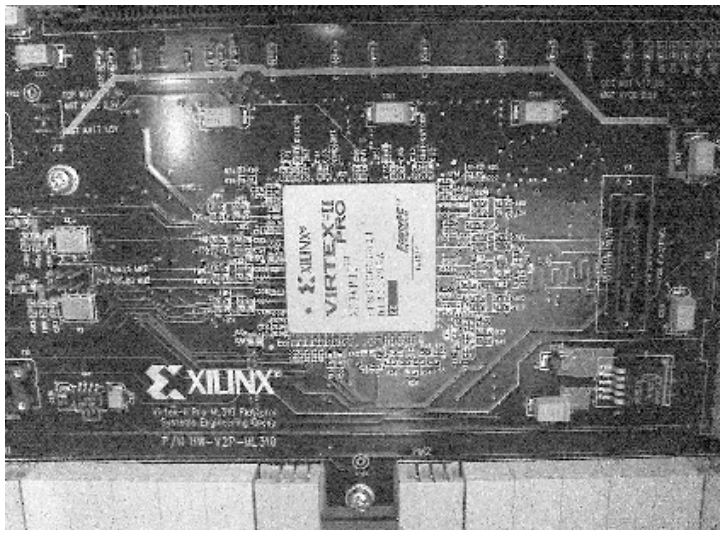


Figure 4.1: Virtex-II Pro FPGA chip

The ML310 is connected to several peripherals as shown in Figure 4.2. The marked blocks that are shown in Figure 4.2 were used in this project.

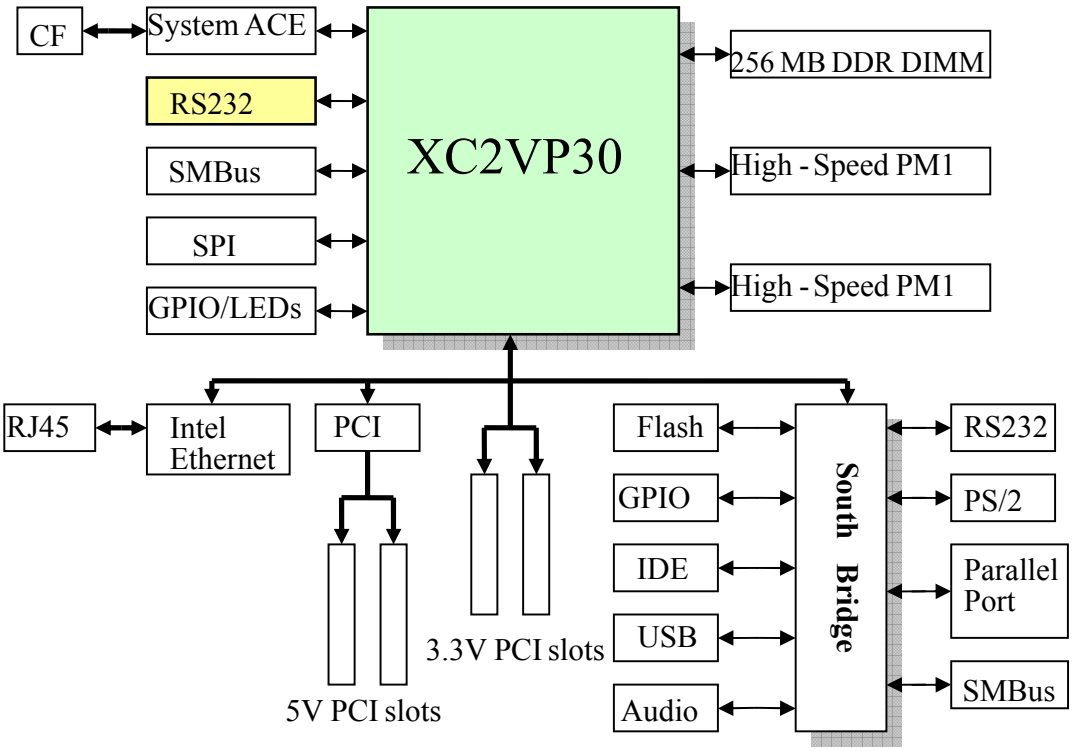


Figure 4.2: ML310 platform diagram adapted from [1]

XC2VP30

The basic element for logic implementation in this FPGA is the configurable logic block (CLB). CLB is fundamental element of the FPGA which forms combinatorial and synchronous logic from user code. The CLB is organized as an array and each CLB has a switch matrix for access into the general routing matrix as shown in Figure 4.3 [4].

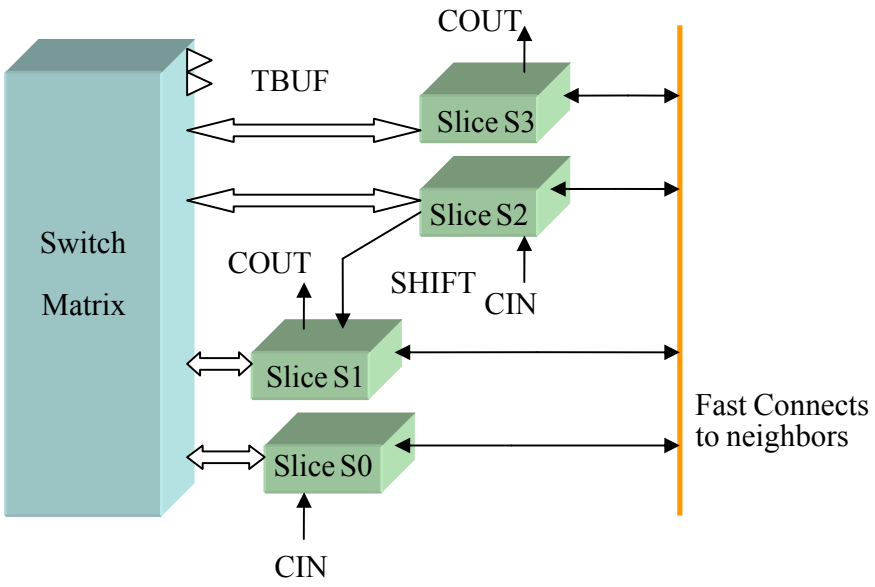


Figure 4.3: CLB in XC2VP30 adapted from [4]

One CLB comprises four slices in 2 columns with two slices in each column. Each pair of slices carries logic chains coming through their column.

Each slice is composed of several parts, which are two 4-input function generators, carry logic, arithmetic logic gates, wide function multiplexers and two storage elements. This is shown in Figure 4.4.

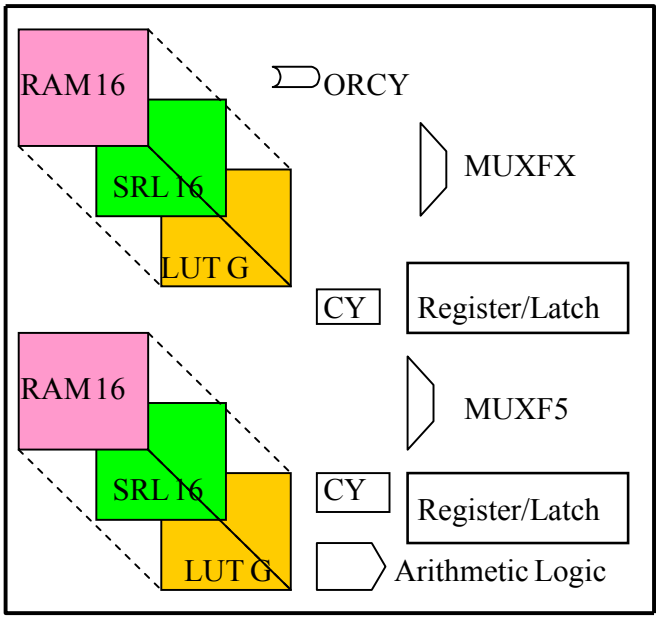


Figure 4.4: XC2VP30 slice configuration adapted from [4]

4.1.1 FPGA Design Approach

The implementation of a turbo code system on FPGA is a complex process because the designer must create a hardware circuit on the FPGA as well as PC software to communicate with it [22] [46].

The approach to the FPGA circuit development is shown in Figure 4.5.

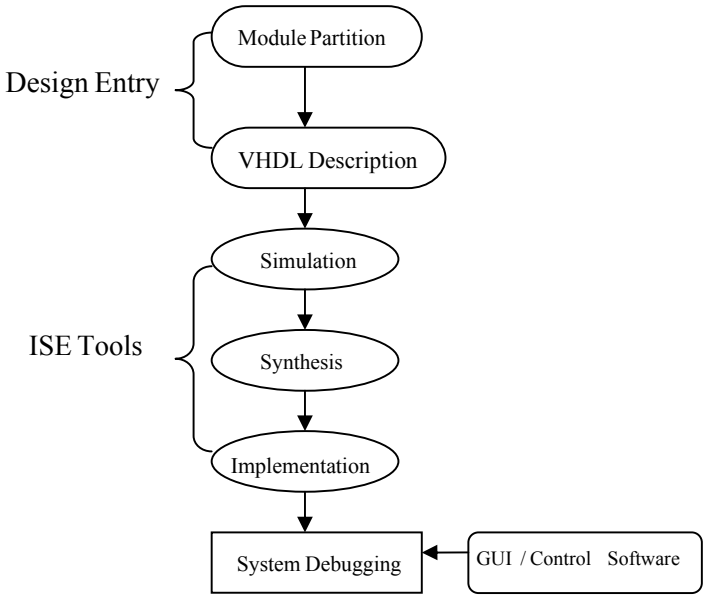


Figure 4.5: FPGA implementation design flow adapted from [50]

Design Entry

Module partition: The functional blocks of the system were configured as shown in the upper section of Figure 4.6, the UART is shown as two separate blocks.

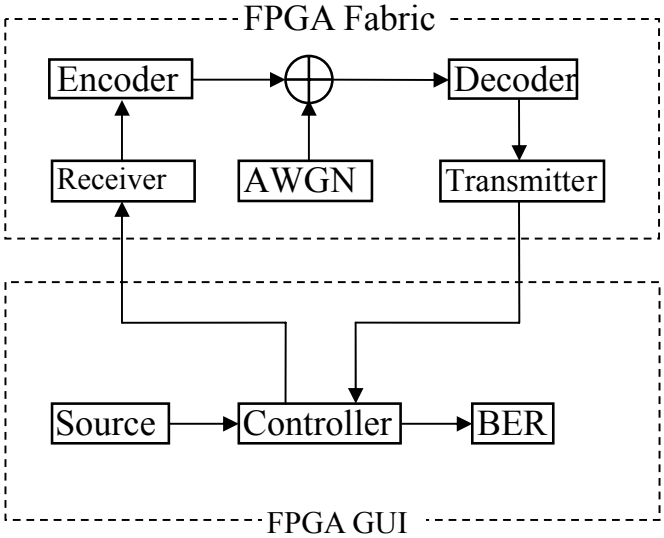


Figure 4.6: System block diagram

The design requires interaction between the PC and FPGA chip. It was therefore decided that the communication system should be established in advance. The communications are described in section 4.1.2.

VHDL description: The functionality of each module that would be implemented on FPGA was described using VHSIC Hardware Description Language (VHDL) according to the design from the previous stage. This was done in the Xilinx *Project Navigator 6.3i.*

ISE Tools

Once the design had been documented using VHDL language, the integrated process of simulation, synthesis and implementation commenced using Xilinx software. At this stage, the designer is unable to modify the design created in the design entry stages. Any errors or unexpected results detected in this stage require the designer to make the modifications in the partitioning and description phases and redo the ISE phases [38] [49].

Simulation: At this stage the function of each module was verified using software tools. Third party software *ModelSim XE II 5.8c* was selected as the simulator for this design.

Synthesis: This process converts the VHDL code to a netlist of primitive modules, such as gate, adder, multiplexer and flip-flop [50].

Implementation: Although logic elements and their interconnections based on the design are defined in a netlist, it does not specify their placement on the FPGA fabric. The implementation tool provided by Xilinx was used to place the elements specified in the netlist and allocate the routing resources for the connections [47] [22]. Timing and area constraints for the placement were specified. Finally, a binary file that included the configuration information of the FPGA was used to program the FPGA chip.

GUI/Control Software Design

The software was coded using Visual C++ and its function includes programming the FPGA and communicating with it through the serial port using API calls. The software can also transmit a set of random test data to the turbo code system and can calculate the bit error rate of the system based on the received decoded data.

A GUI was provided for user. The user can click on one or more command buttons to execute actions. These user actions are all regarded as events and typically result in particular codes in the GUI being executed. A GUI can vary in style types: single document interface (SDI) and multiple document interface (MDI). SDI was used.

System Debugging:

For hardware verification, the *ModelSim* simulator was found to be useful for functional testing at the initial phase. The shortcomings of simulation were shown up when bugs appeared at this stage. All problems identified were successfully removed by adjusting timing.

4.1.2 UART Module Description

Communication Cable and Pin Assignment

The communication system was proposed to be established as a first step. Utilising RS-232 serial communication, some specific commands and test data were sent to the FPGA and feedback data was received for debugging the hardware design at the same time.

RS-232 serial communication is a low-level protocol used to link two devices, originally a data terminal equipment device (DTE) and data Circuit-Terminating Equipment (DCE) [1].

In the current project a Windows PC was connected to the Xilinx ML310 development board. Both of these devices were already configured as DTEs, hence a null modem cable was applied to connect these two devices [1] as shown in Figure 4.7.

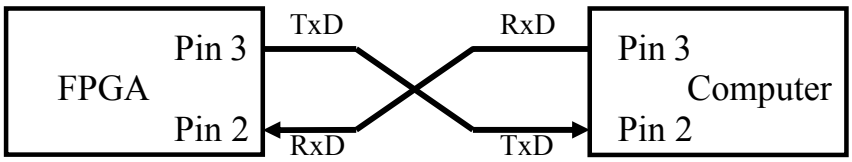


Figure 4.7: Null modem cable

The PC and ML310 both use 9-pin D-type connector. The standard pin assignment is shown in Table 4.1.

Table 4.1: Serial port pin and signal assignments [2]

Pin	Label	Signal Name	Signal Type
1	CD	Carrier Detect	Control
2	RxD	Received Data	Data
3	TxD	Transmitted Data	Data
4	DTR	Data Terminal Ready	Control
5	GND	Signal Ground	Ground
6	DSR	Data Set Ready	Control
7	RTS	Request to Send	Control
8	CTS	Clear to Send	Control
9	RI	Ring Indicator	Control

Only two pins RxD and TxD were used for the communication system. The data flow control pins RTS and CTS were not used. They were omitted in order to simplify the implementation of the UART.

The ML310 board has been designed to fit into a standard PC ATX case. There are three RS-232 ports on the ML310. Two ports are through a South Bridge controller connected to standard front panel interface ports within the ATX case. The third port is directly connected to the XC2VP30 FPGA through a MAX3232 transceiver and an RS-232 mini-cable adapter that converts the 10-pin header to a DB9 male connector as shown in Figure 4.8. This connection requires a UART

function to be implemented in FPGA fabric [1]. This is the simplest method for exchanging data between a PC and FPGA fabric.

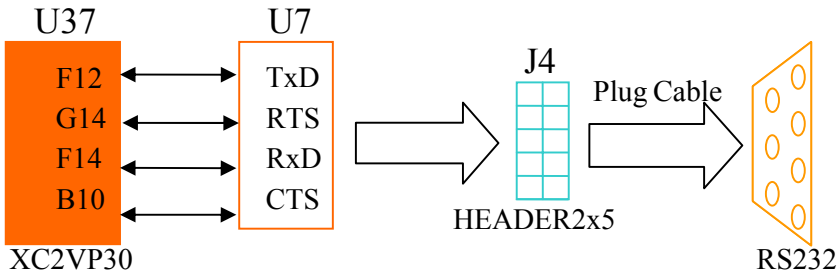


Figure 4.8: FPGA UART and RS-232 connectivity

Figure 4.8 shows 4 signals coming from the FPGA. They are control signals CTS and RTS, which are for data flow control. For simplicity of the UART implementation these were left open. The UART ports were designed as fabric ports F12 and F14 in the constraints file.

The UART is a popular serial communication device that permits duplex communication. The complex features of commercial UART products were not considered necessary for this elementary application. The UART was designed to provide basic half duplex data transmission.

The Structure of the UART

The designed FPGA circuit includes data bus interface, control unit, baud rate generator, receiver and transmitter. Figure 4.9 shows the elements of the UART configured on the FPGA.

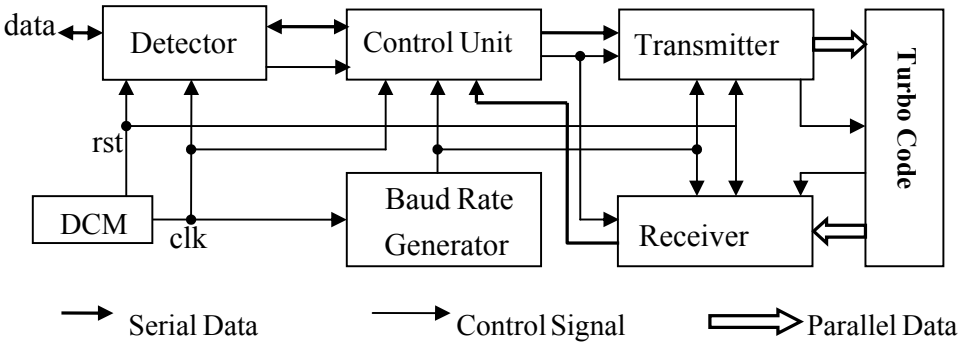


Figure 4.9: The block diagram of UART

Digital Clock Management

Several clocks are distributed throughout the ML310 as illustrated in Figure 4.10. The main system clock is a 100MHz oscillator, X10.

The system clock is commonly used to generate multiple clocks with varying frequency and phases within the FPGA fabric by using the Virtex-II Pro Digital Clock Managers (DCMs).

There are eight DCMs distributed along the top and bottom edges of the XC2VP30 [4]. In this project, the system clock X10 was selected as clock to access DCM X03Y1 through pin B15 of the fabric.

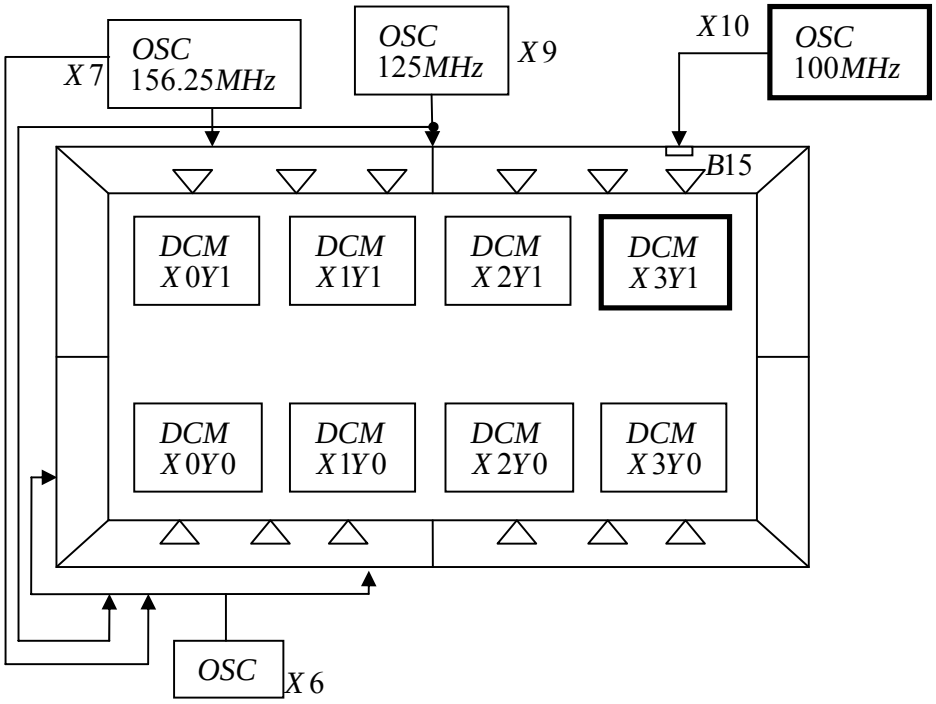


Figure 4.10: Digital clock management and clock distribution on ML310

Figure 4.11 shows the pin connection of DCM X03Y1 that was configured for this project.

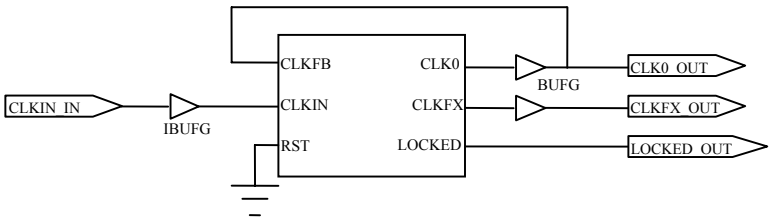


Figure 4.11: Configuration of digital clock management

Frequency synthesis is one of the important functions of the DCM that was used for the UART, because the input frequency does not satisfy the requirement of the UART system. The clock input at the *CLKIN_IN* pin is 100 MHz that can be driven by buffer *IBUFG* and the output from the *CLKFX_OUT* pin should be 58.9824 MHz that is the requirement of UART clock frequency input.

The *LOCKED_OUT* signal was used as a reset signal to guarantee that the system clock is established prior to UART initialisation. When the positive-going edge of *LOCKED_OUT* is detected, the UART is woken up and its variables initialised.

Baud Rate Generator

For achieving high throughput a data rate of 115200 bit/s was chosen for the UART system. The DCM can provide a frequency of 58 982 400 Hz as mentioned in last section. To obtain the chosen data rate of serial communication, a frequency divider was built into the UART. The following VHDL code is a divide-by-512 frequency divider which was incorporated in the UART.

```
process (temp_rst1,temp_rst2,clk,receive_control)
begin
    if temp_rst1 = '1' and temp_rst2 = '0'then
      clk_divder <= "000000000" ;
   elsif clk'event and clk = '1' then
      if receive_control='1' then
             clk_divder <= clk_divder + "000000001";
      end if;
   end if ;
end process ;
clk1x <= clk_divder(8) ;
```

The program was simulated by *ModelSim* and implemented on the FPGA. The *ChipScope* tools were used to analyze the UART circuit on the FPGA fabric in real time. These tools can integrate logic analyzer hardware components with the UART component inside the Xilinx XC2VP30 and communicate with these

components and provide a complete waveform analysis. Figure 4.12 shows the waveform that was gained by using ChipScope Pro Analyzer [53].

Figure 4.12: The waveform of the divide-by-512 frequency divider

Receiver and Transmitter

The serial data format that was selected uses one start bit, eight data bits, and one stop bit. No parity bit was used in this project. Figure 4.13 illustrates the serial data format [3].

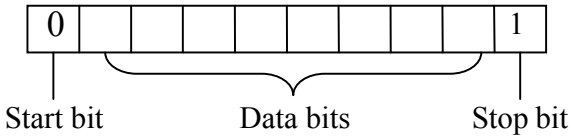


Figure 4.13: Data format of serial communication

By definition, serial data is transmitted one bit at a time. The transmission sequence is:

1. The start bit is transmitted with a value of logical 0.
2. The first data bit corresponds to the least significant bit (LSB), while the last data bit corresponds to the most significant bit (MSB). Altogether 8 bits are transmitted.
3. One stop bit is transmitted with a value of logical 1.

The top-level block diagram of UART that was generated by Synthesis tool is shown in Figure 4.14.

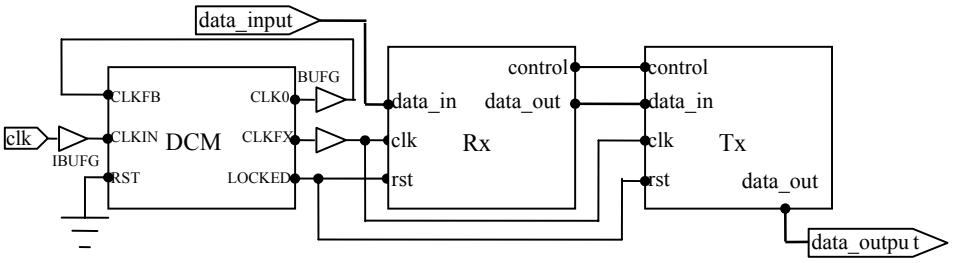


Figure 4.14: Block diagram of UART

The circuit diagrams of the blocks in Figure 4.14 are given in Appendix C.1 and C.2. The program flowcharts of receiver and transmitter are shown in Figure 4.15. The code corresponding to these flowcharts appears in Appendix A.1 and A.2.

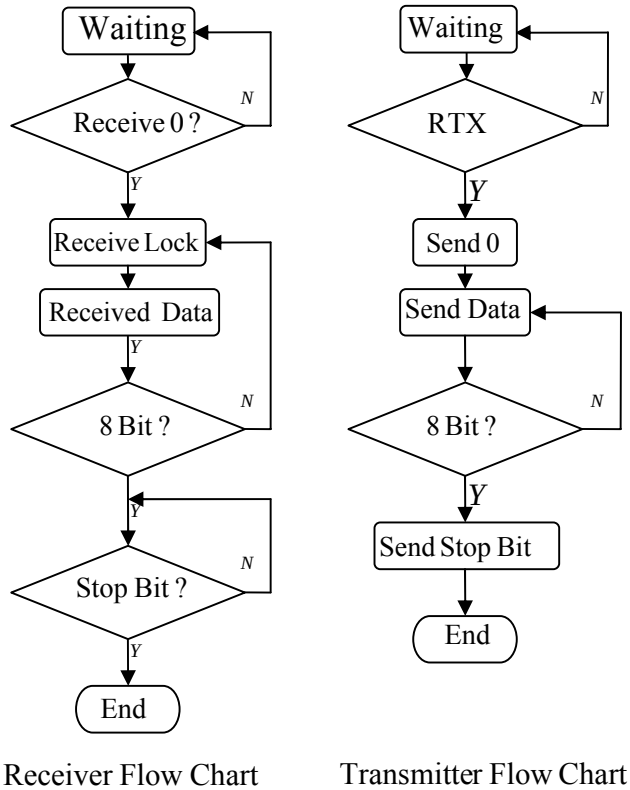


Figure 4.15: Program flowchart of receiver and transmitter

The Figure 4.16 shows the data transmission waveform of the UART.

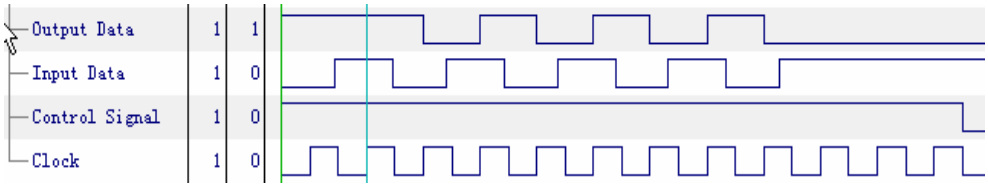


Figure 4.16: Data waveform of UART

4.1.3 Representation of Signal and Noise

Channel model

The noisy channel is modelled as additive white Gaussian noise (AWGN) so that

$$Y = X + G \tag{4.1}$$

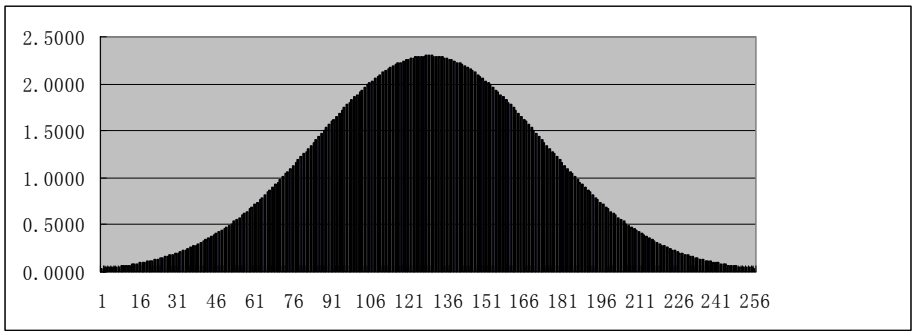
where Y is the channel output, X is the channel input and G is a zero mean Gaussian random variable with mean x and variance σ^2. This is in line with Hokfelt [40, page 5] which also provides the following expression for the probability density function of a random variable.

$$p_{Y|X}(y \mid x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(y-x)^2}{2\sigma^2}} \tag{4.2}$$

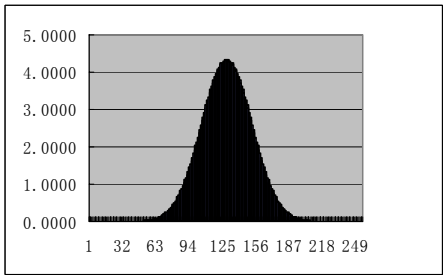
Numerical Representation of signals containing noise

The design started with a choice as to the format of the input random variable for the noise look-up table and the looked-up value. Experimentation with normal distribution curves on a spreadsheet showed that an 8-bit random integer coupled with an 8 bit representation of the resulting noise voltage level produced a visibly smooth graph. Using +25 and −25 units to represent the bipolar signal allowed the standard deviation to be adjusted to produce different S/N values. The practical limit at the high end is 8 dB because above that value no noise peaks

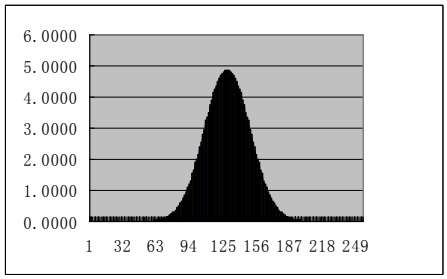
above 25 units are produced. At the low end, values below −5 dB produce a truncation of the Gaussian curve at the edges.



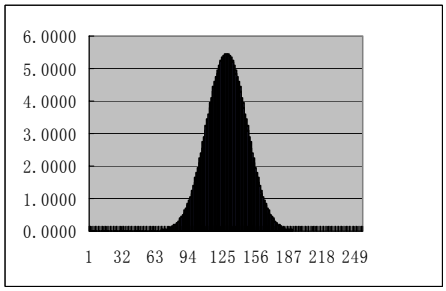
S/N=−5 dB



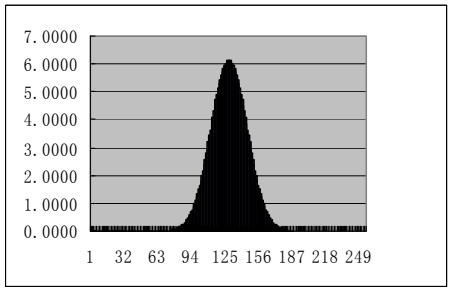
S/N=0.5 dB



S/N=1.5dB



S/N=2.5dB



S/N=3.5dB

Figure 4.17: Distribution of values in five look-up tables

Table 4.2: Look-up table characteristics

Target S/N	0.5 dB	1.5dB	2.5dB	3.5dB
Max	+63 unit	+56 unit	+50 unit	+44 unit
Min	−63 unit	−56 unit	−50 unit	−44 unit
σ	23.5	20.9	18.7	16.6
Actual S/N	0.54 dB	1.56 dB	2.52 dB	3.56 dB

In order to determine the word length for the noisy signal the maximum noise value of +127 units was added to the positive signal level of +25 units. The result (+152) requires 9 bits in two complement. Because continuous accumulation mathematical operations are used to calculate the trellis path metric, the magnitude of intermediate decoded values increases rapidly. As a result, overflow will inevitably happen during the decoding process unless a comparatively longer binary sequence is in use.

Although using a longer binary word for numerical representation can avoid the overflow problem, it would also increase the complexity of the circuit and utilization of the FPGA chip [6]. Therefore, it is necessary to find a suitable word length for representation of the noisy signal in the system. In this turbo code system it was decided to use a twelve bit integer allowing 4096 values from − 2048 to +2047. This was arrived at by running the FPGA system with progressively shorter experimental word lengths until the decoding clearly broke down in comparison with a PC simulation.

A simple single-sided baseband model is used with bandwidth B chosen to be

numerically equal to the bit rate R_b. This makes $S/N=E_b/N_o$. where E_b is energy per bit and N_o is noise power spectral density.

Noise Generator

To reduce on-chip memory utilization, a pseudo-random sequence generator was used to produce a random index for the look-up table. Figure 4.18 shows the structure of the noise generator on chip.

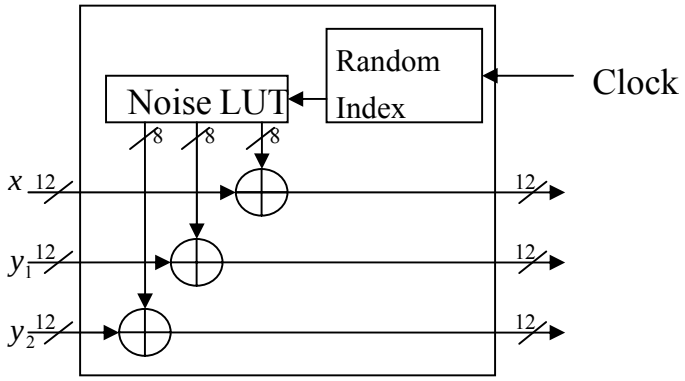


Figure 4.18: Noise generator

4.1.4 Encoder Implementation

The turbo encoder structure is shown in Figure 4.19. The generator function in binary is {111,101} as can be seen from the figure. It is fixed in this project because applying various encoder functions would have significantly increased the complexity of the decoding hardware.

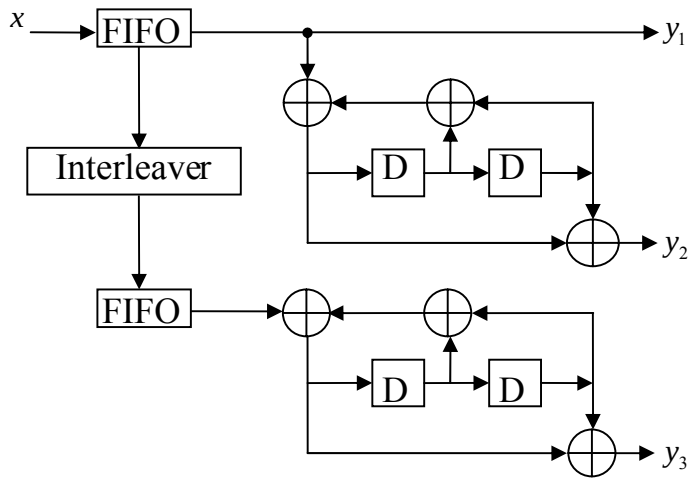


Figure 4.19: Encoder structure on FPGA

The input information bit stream x is stored in a shift register to form the data frame. The frame size was chosen as 10 bits for simplicity of implementation. This data frame is rearranged by an interleaver and then loaded into another shift register. The two RSC encoders then deal with their respective inputs and the encoded bit streams y_1, y_2 and y_3 are transmitted to the channel.

The encoder also was partitioned into modules: *encode_bit* module, *res_encoder* module and *encoderm* module. The VHDL code of each module appears in Appendix A.3.

4.1.5 Decoder Implementation

Working Principle

Following the principle of decoding discussed in section 3.1.1, the two SOVA elements were implemented on the XC2VP30 working in an iterative way. Figure 4.20 shows the components of the decoder and the data flow amongst these components.

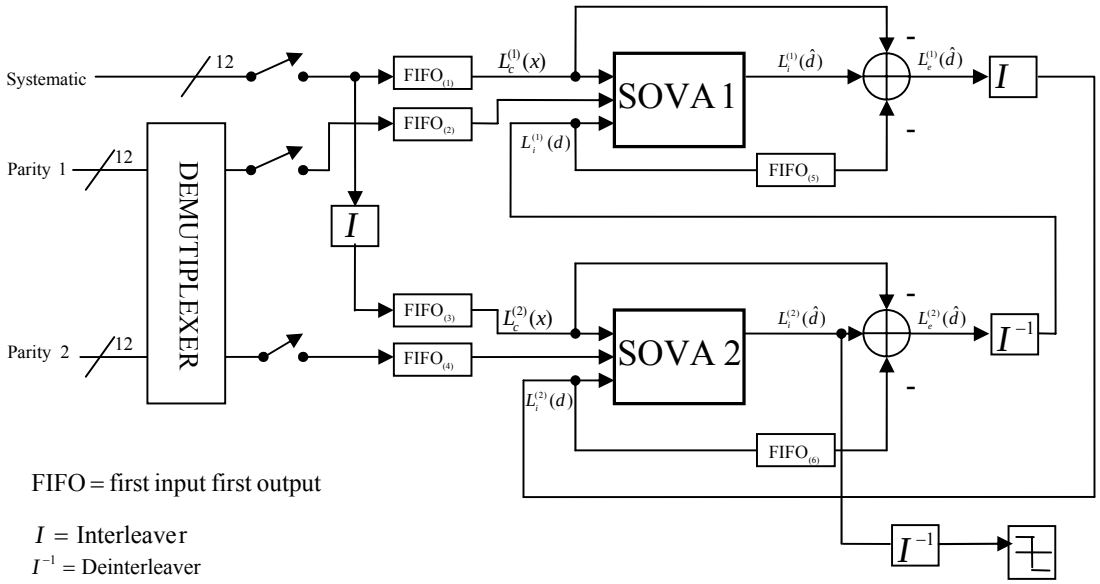


Figure 4.20: Block diagram of turbo decoder

The decoder receives soft channel values frame by frame. The incoming frame comprises of 30 analog values: 10 *systematic* values, 10 *parity 1* values and 10 *parity 2* values. These analog values were represented using twelve bits in the decoder, so every frame contains 360 bits.

Because a half-rate encoder was implemented in the FPGA, half of these *parity 1* and *parity 2* values are omitted in interval order in encoder, so the decoder must

insert zeros in the soft channel output parity sequence to replace this punctured information by using a de-mutiplexer component.

To increase speed of decoding, twelve parallel lines were used to connect the first-input-first-output (FIFO) buffer and the noise generator allowing twelve bits to be loaded into each buffer in one clock cycle.

Figure 4.20 shows the 10 systematic analog values are loaded into $FIFO_{(1)}$. The same values are rearranged by an interleaver and then are loaded into $FIFO_{(3)}$. The two array parity values are shifted into $FIFO_{(2)}$ and $FIFO_{(4)}$ respectively. Those buffers are applied to store values until they are needed.

The three switches in these lines are set in the open position to prevent the next frame from being processed until the current frame has been processed completely.

In the figure, the notation $L_c^{(1)}(x)$ is used to denote the 20 received channel values that are stored in $FIFO_{(1)}$ and $FIFO_{(2)}$. $L_c^{(1)}(x)$ contains the 10 transmitted systematic values and the 10 parity values.

SOVA 1 deals with the soft channel input values $L_c^{(1)}(x)$ and *a priori* information $L_i^{(1)}(d)$ to produce reliability value $L_1^{(1)}(\hat{d})$. In this notation the subscript 1 and superscript (1) indicate the *a posteriori* LLR that was gained in the first iteration

from SOVA 1. Note that there will be no *a priori* information for SOVA 1 in the first iteration, because there is no extrinsic value $L_e^{(2)}(\hat{d})$ available from SOVA 2. So *a priori* information $L_i^{(1)}(d)$ expressed as $L_1^{(1)}(d)$ is initialised to zeros in the first iteration. The content in $\text{FIFO}_{(5)}$ is also all zero.

After SOVA 1 has processed the input values. The extrinsic information $L_e^{(1)}(\hat{d})$ can be gained by the *a posteriori* LLR $L_1^{(1)}(\hat{d})$ of SOVA 1 in the first iteration minus the soft channel inputs values $L_c^{(1)}(x)$ and the *a priori* LLR $L_1^{(1)}(d)$ that have been stored in $\text{FIFO}_{(5)}$ using an adder component. The extrinsic information $L_e^{(1)}(\hat{d})$ that is output from the adder is then rearranged by an interleaver component and used as the *a priori* information $L_i^{(2)}(d) = L_1^{(2)}(d)$ (i is iteration steps) for SOVA 2. In addition $L_i^{(2)}(d)$ values are stored in $\text{FIFO}_{(6)}$ for later use.

Now SOVA 2 comes into play. The channel sequence $L_c^{(2)}(x)$ includes the received *systematic* information and the *parity 2* information which was stored in $\text{FIFO}_{(3)}$ and $\text{FIFO}_{(4)}$ respectively . In addition to the received channel sequence $L_c^{(2)}(x)$, SOVA 2 also receives *a priori* information $L_1^{(2)}(d)$ that was generated by SOVA 1 in the first iteration.

SOVA 2 uses the received channel sequence $L_c^{(2)}(x)$ and the *a priori* LLR $L_1^{(2)}(d)$ to calculate *a posteriori* LLR $L_i^{(2)}(\hat{d}) = L_1^{(2)}(\hat{d})$ (i is the iteration step). In this notation the subscript 1 and superscript (2) indicate *a posteriori* LLR in the first iteration from the SOVA 2.

The extrinsic information $L_e^{(2)}(\hat{d})$ can be gained by the *a posteriori* LLR $L_1^{(2)}(\hat{d})$ of SOVA 2 in the first iteration minus the soft channel inputs $L_c^{(2)}(x)$ in $\text{FIFO}_{(4)}$ and the *a priori* LLR $L_1^{(2)}(d)$ that have been stored in $\text{FIFO}_{(6)}$ using an adder. Then the extrinsic information $L_e^{(2)}(\hat{d})$ that is output from SOVA 2 is rearranged by a de-interleaver component and is used as the *a priori* information $L_2^{(1)}(d)$ for SOVA 1 in the second iteration. This concludes the description of the first iteration.

For the second iteration SOVA 1 also processes the same received channel sequence $L_c^{(1)}(x)$ in $\text{FIFO}_{(1)}$, and *a priori* LLR values $L_2^{(1)}(d)$ that were provided by SOVA 2 in the first iteration. Those *a priori* values are used to improve *a posteriori* LLR $L_2^{(1)}(\hat{d})$ output from SOVA 1 in the second iteration. The second iteration then continues using $L_2^{(2)}(d)$ which was output from SOVA 1 in second iteration to improve *a posteriori* LLR $L_2^{(2)}(\hat{d})$ of SOVA 2 in this iteration.

The precision of the *a posteriori* LLR is increased with every iteration. When the iteration number reaches the predetermined setting, the iteration process stops and the final *a posteriori* LLR values are outputted from SOVA2 through the log-likelihood function (LLF) component for the hard decision. This final step simply outputs a ’1’ for any positive input and a ’0’ for any negative output.

It is obvious from inspection of Figure 4.20 that the decoding operation can easily be split up into modules. By defining an interface between the modules, each of the modules can be designed to work with inputs from and provide outputs to other modules. The two major modules from Figure 4.20 are the interleaver/de-interleaver and SOVA modules.

Interleaver/De-interleaver Module

In the interleaver, the ten data bits are read sequentially and rewritten in a new sequence according to a fixed interleaving pattern.

Figure 21(a) shows interleaving of a six-bit frame. The interleaving pattern is an array with length equal to the frame length. The pattern values represent the address of memory containing the values to be interleaved. As shown in Figure 4.21(b), for de-interleaving in this scheme, the data is read linearly and written according to an interleaving pattern. The interleaver and de-interleaver must use the same interleaving pattern.

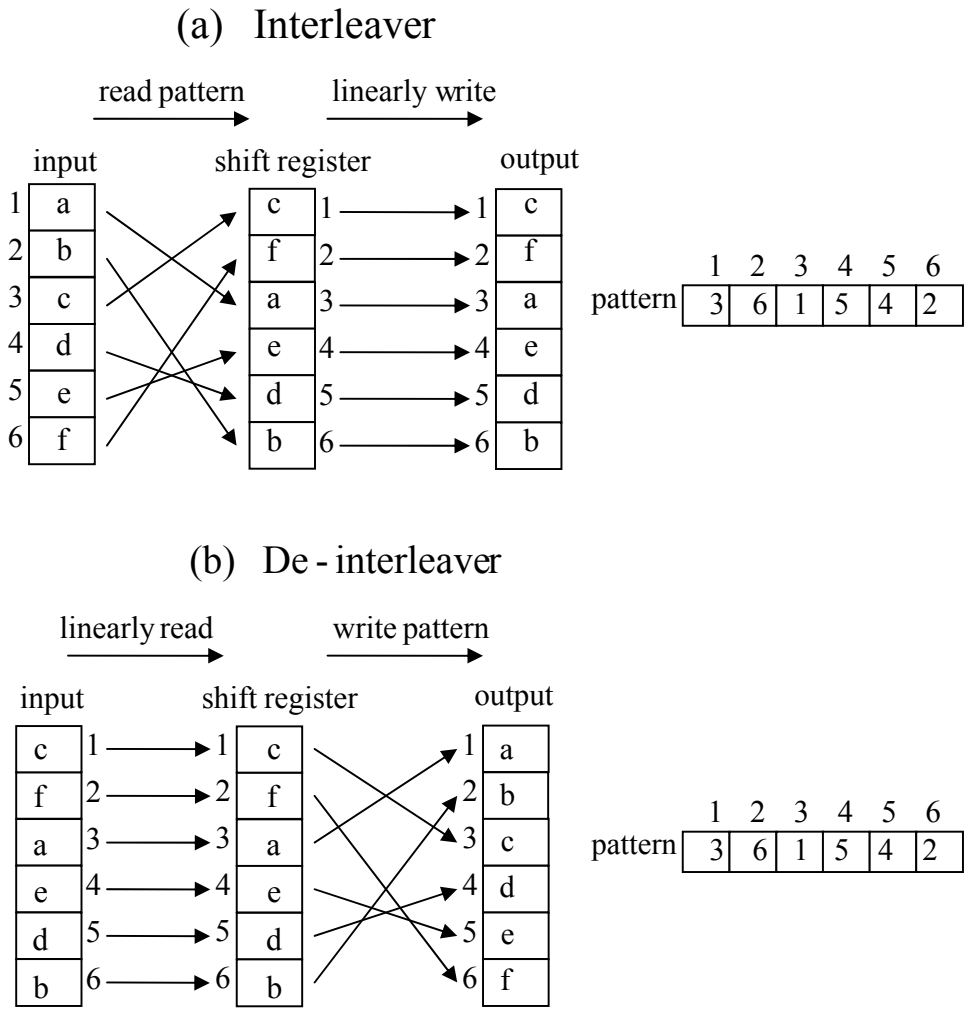


Figure 4.21: Interleaver and de-interleaver

Since it is possible to configure CLBs on the FPGA as RAM elements, the interleaving pattern can be stored on the FPGA. The major disadvantage of this method is that it can consume large portions of the chip area if the pattern length is large.

The Termination Problem

The two SOVA blocks in Figure 4.20, SOVA 1 and SOVA 2 are not totally identical. There is one small difference that is related to solving the trellis termination problem.

In section 2.1.3, a trellis termination method was used to rectify the problem by using the final M bits that remain in memory of the first RSC encoder to force the state to all-zeros at end of the block. Although this method will terminate the first RSC encoder correctly, it will not terminate the second RSC encoder. That is because the systematic bits are rearranged by an interleaver and then supplied to the second RSC encoder. That means that the SOVA 2 has no prior knowledge of the end state for its trace-back the on trellis [46].

To solve this problem, the trace-back process should give different start states for the two SOVAs. For SOVA 1, the trace-back trellis can be started from an all zero state, however for SOVA 2 the trace-back trellis should be started from the most likely state. That means finding the maximum path metric and using its state as start state.

As a means of saving fabric area, a single SOVA module was designed to be able to act as either the SOVA 1 or SOVA 2.

SOVA Implementation

The SOVA module can be implemented in the following steps which were derived from the Matlab code of Wu [30]. The VHDL Code of SOVA module appears in Appendix A.4.

1. Initialize the path metrics $M_{k=1}^{S_K=1}$ at the first state of the first stage to zero, and set the other path metrics to a negative infinite value, specifically -2048.
2. Initialize the trellis stage k to 1.
3. Initialize state S_k at the stage k to 1
4. In a binary trellis there will be two paths reaching state S_k at stage k.
   (a) Use Equation (3.57) to calculate the path metrics of those two paths

$$M_k^m = M_{k-1}^{m'} + \frac{1}{2}.(\pm L(d_k)) + \frac{L_c}{2}\sum_{i=1}^{n}(x_{ki}R_{ki}) \qquad (3.75)$$

   M_k^m is the accumulated path metric at the stage k.

   n is the codeword length.

   x_{ki} is the individual bit of transmitted codeword x_k in the trellis.

   R_{ki} is the symbol in received codeword R_k from the channel.

   L_c is the channel reliability value.

   $L(d_k)$ is the a priori reliability value at the stage k.

   (b) Calculate the absolute value between those two path metrics and store the value.

(c) Find maximal path metric for S_k state at stage k and store the maximal path metric and its associated bit.

5. Set state $S_k = S_k + 1$, go back to step 3 until $S_k > 4$.

6. Set stage $k = k + 1$, go back to step 2 until $k > 11$.

7. Trace back in the trellis to get the estimated bits, and the most likely path. For SOVA 1, the trace-back process starts from the first state of the last stage. For SOVA 2 the start state is the state which has the maximal path metric when the last stage is reached.

8. Initialize the trellis stage k to 1.

9. Initialize reliability value $L(d_1 \mid R) = +\infty$ and window size δ=5. In this implementation $+\infty$ is represented by +2047.

10. (a) Initialize window size δ=0

(b) Using Equation (3.81) to calculate reliability value for current stage.

$$L(d_k \mid R) \approx d_k \min_{\substack{i=k\cdots k+\delta \\ d_k \neq d_k'}} \Delta_i \qquad (3.81)$$

11. Set stage $k = k + 1$, go back to step 10 until the end of the trellis is reached.

4.1.6 Implementation Issues

During the implementation process, a problem appeared in the iterative decoding system. When the number of iterations was adjusted from one to two, the device utilisation also increased by a factor of two. This led to a systematic analysis which revealed that the SOVA module was being duplicated in the

implementation process. The SOVA module is the most complex in this scheme, about 37% of the fabric resource is required to implement a single iteration step. The reduplication of this module was therefore significantly increasing the chip utilization. As a consequence, just 2 iterations could be applied to the turbo code system.

As previously explained (section 4.1.5), the SOVA module was designed as a multiple-function module to allow reuse in every iteration step. It was found that the implementation tools did not allow SOVA module re-use in different iteration steps. That is because *on-chip* memory was used to store intermediate variables during decoding and no *off-chip* memory was adopted in this implementation. Those variables also occupy some resource of the chip. As a result, the variables and SOVA module were placed as a single unit by the implementation tools in every iteration.

Solving this design defect would allow longer frame size and more iteration steps of the turbo code implementation. However, it was realized that in order to rewrite the design of this implementation to add the external RAM, the entire VHDL code of the current design would have to be rewritten. In discussion with the author's supervisors it was decided to complete the project using the existing design, including its identified shortcomings.

4.2 Simulation Software

The iteration problem that limited the extent of the FPGA implementation was described in section 4.1.6. To make up for this shortcoming it was decided to implement a fully-functional PC turbo-code simulation as an additional educational tool.

The software was created by YuFei Wu [30] in Matlab. With a view to licence-free dissemination of this part of the system it was decided to remove its dependency on Matlab. Wu has indicated her general consent to this software being used for academic purposes in a note contained in the Matlab source code file [30].

The structure used by Wu is in turn based directly upon the structure described in the original paper by Berrou *et al* [10]. It permits the following adjustments to be applied:

- Puncturing On /Off
- SOVA or MAP decoding algorithm
- Constraint length from 3 to an arbitrary upper limit
- Interleaver size from 2 to an arbitrary upper limit

The solution chosen was to make a translation to C++ code using a trial copy of Matcom, a Matlab-to-C++ conversion utility [51]. Figure 4.22 was generated by the converted program.

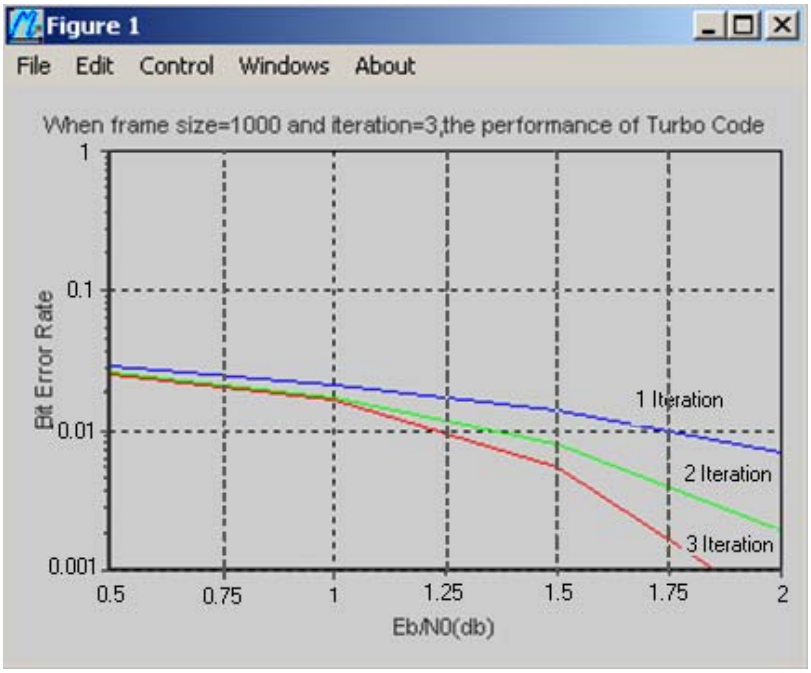


Figure 4.22: Bit error rate of turbo code

4.3 Graphical User Interface

This section describes a Graphical User Interface (GUI) which provides access to the two simulation models described in section 1.3. The functionality of the GUI is introduced in section 4.3.2 and 4.3.3, and some significant message handlers of the GUI are discussed in section 4.3.4.

4.3.1 Design of GUI

Six controls were used in the design. These are *static text, button, edit box, combo box, picture* and *check box.* The GUI is shown in Figure 4.23.

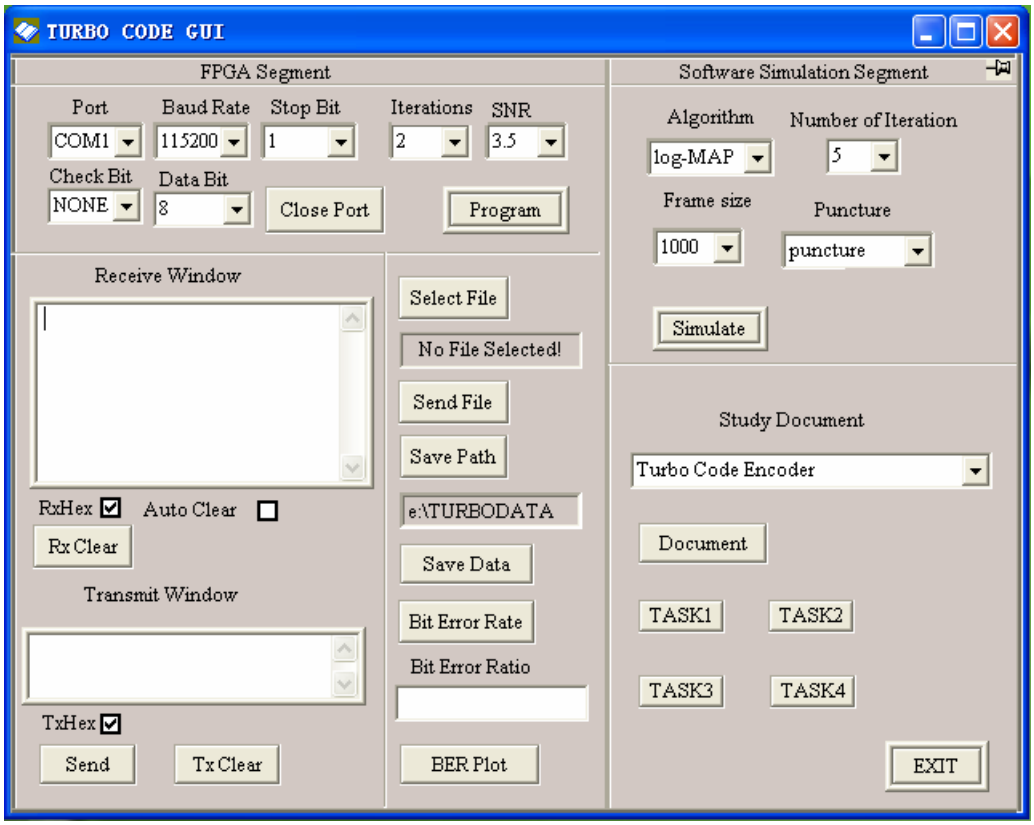


Figure 4.23: GUI of project

The GUI was partitioned into two completely independent parts, *FPGA Segment* and *Software Simulation Segment.*

4.3.2 FPGA Segment of GUI

Each simulation run produces one file containing the received version of the transmitted file. If it is required to produce a BER/SNR plot, a utility is provided to graph the result of current runs.

The operational requirements for the *FPGA Segment* are given below.

1. There are five *combo boxes* at upper left are used to select different parameters for the serial port.

   Although, in this implementation, the baud rate setting is fixed, and all the parameters are initialized when the GUI is launched, the optional serial port parameters make the system suitable for high transmission speeds.

   The button *Close Port* switches off the serial port.
2. The two *combo boxes* at upper right are used to select parameters for the simulation. The iteration limit is 1 or 2, and the SNR can selected as 0.5dB, 1.5dB, 2.5dB or 3.5dB.
3. The *Program* button will invoke FPGA configuration software to program the previously designed binary file associated with the chosen parameters.
4. After chip loading, the buttons *Select File* and *Send File* are used to transmit a test file to the FPGA. Alternatively, test data can be typed into *Transmit Window* and the data can be sent to the serial port by pressing *Send*.
5. The *Receive Window* is used to display the received decoded data. If the check box *Auto Clear* is checked, the *Receive Window* will automatically clear when it has displayed more than 400 characters. The check box *RxHex* is used to change the data format that will be displayed in the *Receive Window*. If *RxHex* is checked, the data format is hexadecimal. The button *RxClear* is used to clear the *Receive Window.*
6. The button *Save Data* is used to save the decoded data into a file.

7. The button *Bit Error Rate* is activated to calculate the bit error rate for the current simulation. When simulation for different SNR and iteration count are processed, a plot can be shown by pressing button *BER Plot*.

4.3.3 Software Simulation Segment of GUI

Operational requirements for *Software Simulation Segment* are discussed below:

1. The four *combo boxes* in the top block are used to select the following configuration parameters: SOVA or MAP algorithm; frame size: 10, 100, 1000 bits; iteration limit: 1, 3, 5; puncturing or no puncturing.
2. The button *Simulate* is use to invoke the simulation software which runs to completion without further intervention.
3. On-line tutorial material written by the author is accessible from the GUI. The user can open this material by clicking on *Document*.
4. In the bottom block, the user can use the *Task* button to access four predefined tutorial tasks

4.3.4 Utilities Controlled from GUI

Sections 4.3.2 and 4.3.3 describe the structure of the GUI which was created entirely by automatic code generation. When the user clicks on any of the controls, certain effects are produced. The current section describes these effects.

Serial Communications

For simple serial communication, a freeware serial port class was used in the GUI code. This class was created by Remon Spekreijse [28] and was named

CSerialPort. This class is very suitable for the communication requirement of this project.

The class was designed as a base class for driving hardware or reading hardware via the serial port [29]. To apply this to the GUI, code was written to provide GUI access to port parameters and permit monitoring of communication events. The event *WM_COMM_RXCHAR* is the only one used in the project. It signifies that a character has been received and placed in the input buffer. When the received character is detected it means that the turbo code on the FPGA has finished processing the current frame. The main program then invokes the *communication function* to display the received frame and send a new frame to the serial port. The *communication function* C code is listed in Appendix B.2.

GUI Control of FPGA Configuration

The Xilinx *Impact* software is used for the physical configuration (programming) of the FPGA chip. *Impact* supports a command line method to invoke program function. It was necessary to integrate this function with GUI operations in order to meet the educational specifications for the system.

The following command file was created to provide the command line instructions.

```
Setmode –bs
Setcable –p auto
identify
assignfile –p 2 –file uart_fpga.bit
```

```
program –p 2
quit
```

The first command line is used to set the configuration mode. There are four configuration modes for Virtex II Pro devices: Boundary-Scan, SelectMap, Master-Serial and Slave-Serial. The commonly-used Boundary-Scan mode was selected for this application. The second line sets up the cable port. The *identify* command in the third line means initialises the chain. The command in the fourth line assigns the binary file that is applied to configure the FPGA. The last two commands are the configuration command and quit command. This file was saved as *program.cmd*.

GUI execution of this programming function is implemented by the appropriate button:

```
void CSCOMMDlg::OnButtonProgram()
{
      GetModuleFileName(NULL,exeFullPath,MAX_PATH);
      CString strlpPath;
      strlpPath.Format("%s",exeFullPath);
      strlpPath.MakeUpper();
      strlpPath.Replace("TURBOCODE.EXE","");
      ShellExecute(NULL,NULL,_T("turbocodeprograme.bat"),NULL,
   strlpPath,SW_SHOW);
}
```

On clicking this button, the configuration program is invoked. Figure 4.24 shows the configuration interface.

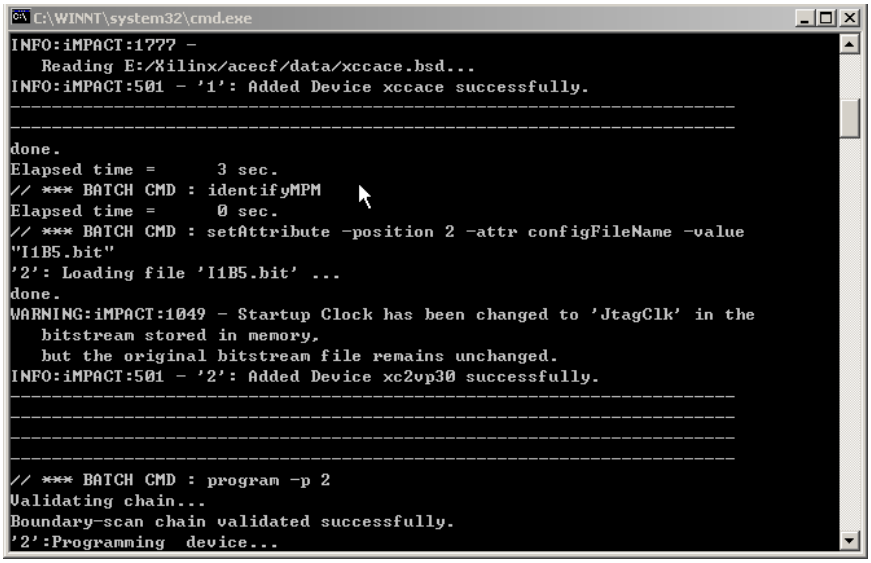


Figure 4.24: Configuration interface

Invoke Matlab to Plot

Writing VC++ code to plot a graphic of bit error rate was not considered convenient, so plotting of BER was achieved by invoking Matlab by clicking a control button.

To invoke Matlab from C++, the Matlab engine code must be compiled together with the GUI code. To accomplish this, the engine header file *engine.h* and its associated library files were added to the GUI program. The code for the plotting BER is given in Appendix B.1.

Chapter 5. Evaluation

This chapter reviews the performance of the hardware and software structures created in this project. In section 5.1 they are judged as technical turbo code systems and in section 5.2 as educational artifacts.

5.1 Technical Performance

5.1.1 Virtex II Pro Implementation

An ideal representation of the performance expected from the FPGA implementation was modelled using the original Matlab simulation code of Wu [30]. This is shown in Figure 5.1.

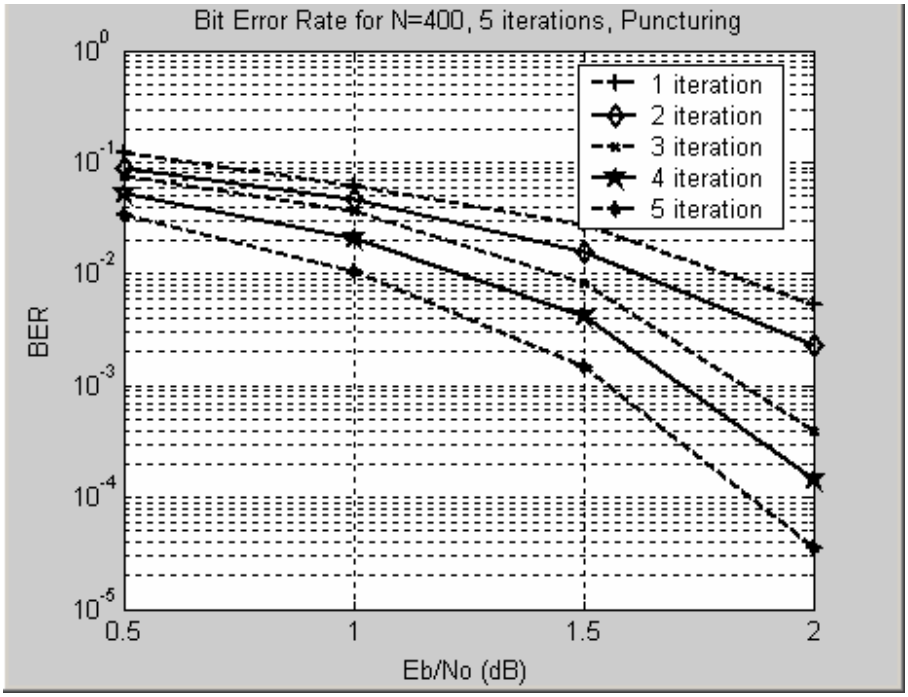


Figure 5.1: Proposed FPGA performance modelled in Matlab

The time taken for the 5-iteration simulation was 28 minutes. The intention was that the FPGA implementation would reduce this by at least one order of magnitude.

The turbo code parameters that were designed into the FPGA implementation are listed in Table 5.1.

Table 5.1: Parameters used for FPGA implementation

Parameter	Value
Frame Size	10
Constraint Length of RSC Encoders	3
Generator Function of RSC Encoders	{7.5}
Code Rate	1/2
SISO Decoding Algorithm	SOVA
Number of Decoding Iterations	1 or 2

The only adjustable parameter is the number of iterations. As explained in Section 4.1.6, two is the maximum in this implementation, so this was the value used. The frame size was also restricted to 10.

In order to test the functionality, the results from the FPGA implementation were compared with the results from a Matlab simulation using [30] with the same parameter settings.

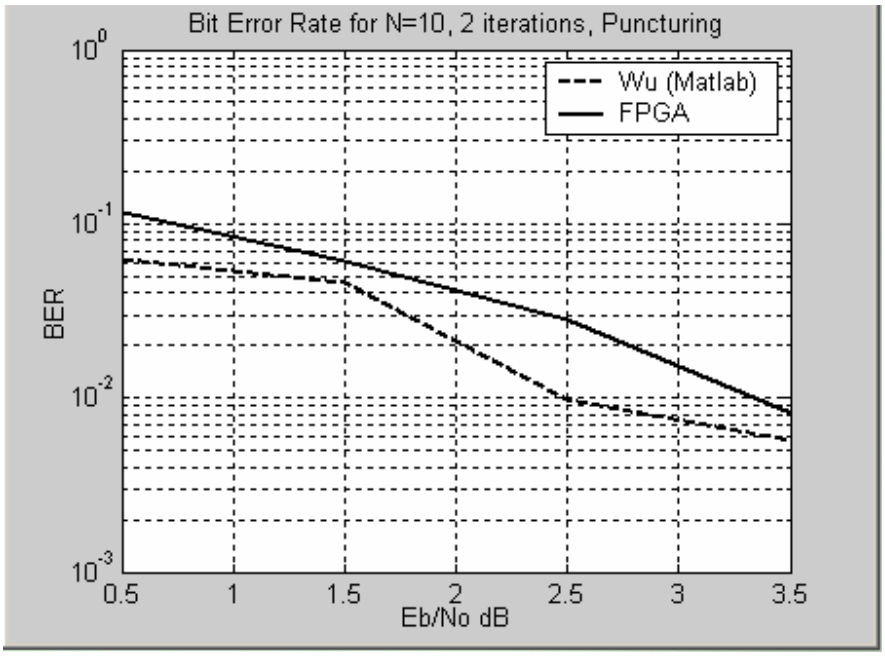


Figure 5.2: Comparison between FPGA implementation and Matlab simulation

Figure 5.2 shows plots for SNR from 0.5dB to 3.5dB. The Wu code is on average 0.7 dB better than the hardware implementation. This can be attributed to the use of floating point representation for the noisy signals instead of the 12-bit integers chosen for the FPGA. This difference was not regarded as a significant shortcoming. The discrepancy in performance between these two implementations was assumed to be due to the differing numerical representations for signal and noise. The software simulation uses floating point and the FPGA implementation uses an 8-bit integer.

Figure 5.3 uses the same parameters as Figure 5.2 and demonstrates that for the FPGA system the second iteration produces an improvement on the first. The expectation is that the application of an external memory solution to the chip

utilization problem outlined in 4.1.6 would produce progressively lower BER plots. This approach would also permit a realistic frame size.

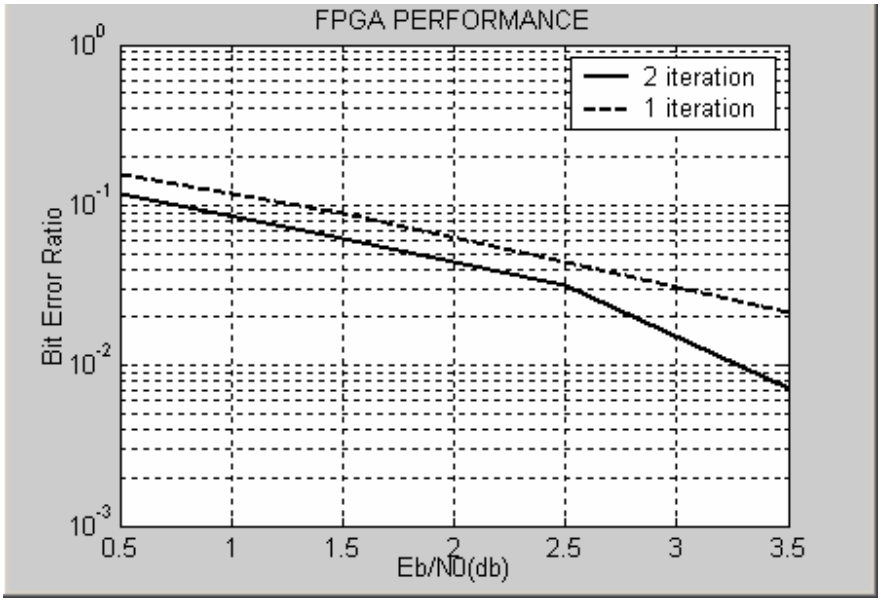


Figure 5.3: Two iterations of FPGA implementation

The time taken to complete two iterations was measured as 1.8s including transfer of data from the PC to the ML310 FPGA unit and back.

It is clear from the above that the performance levels initially visualised for the FPGA were not realised.

5.1.2 PC Simulation

One of the primary reasons for using a hardware implementation is speed of execution. In contrast with the FPGA speed reported above, the PC simulation runs about one order of magnitude slower.

C++ translation improved execution speed by a significant factor because it is compiled code not interpreted code. This is illustrated in Table 5.2 which uses the parameters given in Table 5.1 but with different iteration counts.

Table 5.2: Simulation time taken

Number of iteration	Matlab code	C++ code
1	61s	20.7s
3	78.3s	36.6s

The PC simulation still falls far short of the FPGA potential. The implications of this poor performance are discussed in section 5.2.

5.1.3 GUI

Both the FPGA and PC simulation GUI segments were found to provide the interface functionality described in sections 4.2.2 and 4.2.3. A schematic of the GUI structure is given in Figure 5.4.

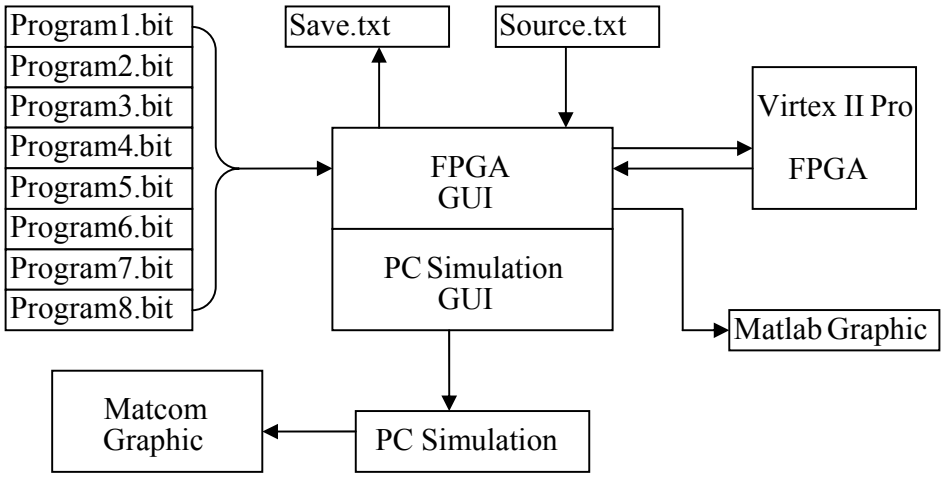


Figure 5.4: GUI software schematic

5.2 Operational Performance

It is well documented that learning is facilitated by rapid feedback of performance e.g. [48]. The original concept of a high speed FPGA demonstration module proved unattainable with the first-order design approach. This has effectively removed the intended basis for the educational utility of the FPGA demonstration. For this reason no evaluation of the FPGA demonstration system as an educational tool was undertaken.

The limitation of the FPGA approach led to the inclusion of the Matcom translation of the Matlab code of [30]. This does result in a slight speed improvement as shown in Table 5.2, but educationally useful results such as those produced by the test runs described in section 5.1.1 take as much as 28 minutes to complete. For this reason, further operational evaluation of this system in an educational context was not undertaken.

Chapter 6. Conclusions

This final chapter summarizes the degree of success of this project, and discusses possible extensions and improvements.

6.1 Achievements

- Turbo encoder and decoder algorithms were studied in general and the MAP and SOVA algorithms in particular.
- A turbo code encoder and decoder were implemented on an FPGA platform together with a simulated AWGN channel.
- A translation of a well known turbo code simulator was produced in order to allow its use without Matlab.
- A Graphical User Interface (GUI) was developed which provides full control of the two simulation models mentioned above.

6.2 Future Work

This section presents two improvements to the design that could be made.

6.2.1 Change Scheme of Existing Design

The huge resource requirement of the turbo decoder iterations using a large frame size is the principal difficulty to be overcome. Even with a tenfold increase in fabric area the current project would have encountered limitations.

The best approach would be to allow external RAM storage of intermediate values at the end of each iteration [22]. The use of an IP or embedded processor is rejected on the grounds of speed.

Jason R. Hess [46] proposes the following technique. The on-fabric modules use the external memory to get back their inputs and store their outputs. The memory provides a uniform interface between all the modules. Figure 6.1 illustrates the scheme.

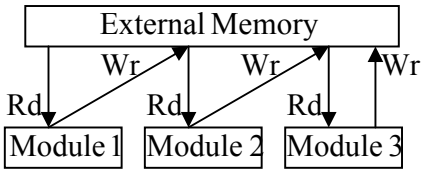


Figure 6.1: Memory as an interface between modules adapted from [46]

This structure would eliminate the hardware duplication problem caused by the iterations of the SOVA algorithm. At the same time, more iterations and a larger frame size could be applied.

6.2.2 Implement MAP on FPGA

The educational utility of the final system would be enhanced by allowing the user to select between MAP and SOVA.

6.2.3 Provide Connectivity to Other Educational Hardware

The ML310 board has facilities for direct connection of TTL data streams to the fabric. This would allow the system to be more truly representative of a hardware module than the current PC-controlled demonstrator.

References

[1] *Xilinx ML310 User Guide,* UG068 (v1.1.1), Xilinx Inc., October 2004.

[2] Serial Port Basics
http://202.84.17.102/javajk.htm

[3] Serial Communications with FPGA
http://www.mcu99.com/Article/FPGA/200605/935.html

[4] *Virtex-II Pro and Virtex-II Pro X Platform FPGAs*: *Complete Data Sheet*, Xilinx Inc., June 2004.

[5] W. G. Spady, *Outcome Based Education*: *Critical Issues (Paperback)*, American Association of School Administrators, April 1995.

[6] H. Michel and N. When, "Turbo-Decoder Quantization for UMTS," *IEEE Communications Letters*, Vol. 5, No. 2, February 2001.

[7] D. Divsalar and F. Pollara, "Turbo Codes for PCS Applications," *Proceedings of IEEE ICC'95*, Seattle, Washington, pp.54-59, June 1995.

[8] B. Sklar, "A Primer on Turbo Code Concepts," *IEEE Communications Magazine*, pp. 94-102, December 1997.

[9] B. Sklar, *Digital Communications*: *Fundamentals and Applications*, Prentice-Hall.

[10] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Transactions on Communications*, Vol. 44, No. 10, pp.1261-1271, October 1996.

[11] F. H. Huang, *Evaluation of Soft Output Decoding for Turbo Codes*, MSc dissertation, Virginia Polytechnic Institute and State University, May 1997.

[12] Agilent Advanced Design System
http://eesof.tm.agilent.com/products/ads_main.html

[13] H. R. Sadjapour, N. J. A. Sloane, M. Salehi and G. Nebe, "Interleaver Design for Turbo Codes," *Proceedings ISIT-2000*, p. 453, 2000.

[14] F. Chan, "Matched Interleavers for Turbo Codes with Short Frames," *Seventh Candian Workshop on Information Theory*, June 2001.

[15] G. D. Forney, "The Viterbi Algorithm," *Proc. of the IEEE*, Vol. 61, No. 3, pp.268-278, March 1967.

[16] L. Hanzo, T. H. Liew and B. L Yeap, *Turbo Coding*, *Turbo Equalisation and Space-Time Coding*, Wiley, 2002.

[17] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," *GLOBECOM 1989*, Dallas, Texas, pp.1680-1686, November 1989.

[18] C. Berrou, P. Adde, E. Angui and S. Faudeil, "A low complexity soft-output Viterbi decoder architecture," in Proceedings of the *International Conference on Communications*, pp.737-740, May 1993.

[19] S. A. Barbulescu, *Iterative Decoding of Turbo Codes and Other Concatenated Codes*, PhD thesis, University of South Australia, February 1996.

[20] J. Hagenauer, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory,* Vol. 42, No. 2, March 1996.

[21] P. Robertson, E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *in Proc. Int. Conf. Communications*, pp. 1009-1013, June 1995.

[22] W. B. Puckett, *Implementation and Performance of an Improved Turbo Decoder on a Configurable Computing Machine*, MSc dissertation, Virginia Polytechnic Institute and State University, July 2000.

[23] R. Mager and P. Pipe, *Criterion-Referenced Instruction 2ed*, Mager Associates, 1979.

[24] R&S WinIQSIM simulation software *http://www2.rohde-schwarz.com /product/winiqsim.html*

[25] J. Qi, *Turbo Code in IS-2000 Code Division Multiple Access Communication under Fading*, MSc dissertation, Wichita State University, Fall 1999.

[26] W. George, *Optimised Turbo Codes for Wireless Channels*, PhD dissertation, Communications Research Group, Department of Electronics University of York, UK, October 2001.

[27] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," *IEEE Procedings of the Int. Conf. on Communications*, Geneva, Switzerland, (ICC'93) pp.1064-1070, May 1993.

[28] K. V. Shibu, Implementing Serial Communication in Win9X/2000, October 2002. *http://www.codeguru.com/cpp/i-n/network/serialcommunications /article. php/c5395/*

[29] R. Spekreijse, A communication class for serial port, February 2000. *http://www.codeguru.com/Cpp/IN/network/serialcommunications/article. php /c2483*

[30] Turbo Code Demo *http://www.ee.vt.edu/~yufei/turbo.html.*

[31] M. C. Valenti, "Turbo codes and iterative processing," in *Proc. IEEE New Zealand Wireless Commun. Symposium*, Auckland New Zealand, November 1998.

[32] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," *TDA Progress Report 42-126*, *Jet propulsion Laboratory*, pp.29-39, California, February 1995.

[33] A. Mugaibel and M. A. Kousa, "Understanding Turbo Codes," *6th IEEE Technical Exchange Meeting*, pp. 163-167, Dhahran, Saudi Arabia, April 1999.

[34] V. Garousi, *Methods to Reduce Memory Requirements of Turbo Codes*, MSc dissertation, Electrical and Computer Engineering, Waterloo, Ontario, Canada, 2003.

[35] H.R.Sadjapour, M.Salehi, N. J. A. Sloane and G.Nebe, "Interleaver design for short block length Turbo codes," *IEEE International Conference on Communications*, Vol.2, pp.628-632, June 2000.

[36] R. M. Banakar, *A Low Power Design Methodology For Turbo Encoder and Decoder*, PhD dissertation, Indian Institute of Technology, Delhi, July 2004.

[37] K. Andrews, V. Stanton, S. Dolinar, V. Chen, J. Berner and F. Pollara, "Turbo Decoder Implementation for the Deep Space Network," *IPN Progress Report 42-148*, February 2002.

[38] J. Van der Spiegel, VHDL Tutorial. *http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html*

[39] J. G. Proakis, *Digital Communications*, *Fourth Edition*, McGraw-Hill, 2000.

[40] H. Johan, *On the Design of Turbo Codes*, PhD thesis, Department of Applied Electronics, Lund University, August 2000.

[41] L. Zhang, A. Yongacoglu, "Turbo Coding in ADSL DMT Systems," *The 2nd International Symposium on Turbo Codes*, Brest, France, pp.157-160, September 2000.

[42] S. MacPherson, M. McLeod and S. Shi, "A Cost-Sensitive Approach to Laboratory Work for a Fourth-Semester Course in Digital Modulation," *The 8th IEEE Africon Conference*, Windhoek, Namibia, September 2007.

[43] J. He, J. Costello,Y. F. Huang and R. L. Stevenson, "On the application of turbo codes to the robust transmission of compressed images," *ICIP (3)*, pp. 559-562, 1997.

[44] Turbo Codes Boost Wireless LAN Performance *http://www.bbwexchange.com/news/2002/mar/vocal032502.htm*

[45] S. Pietrobon and A. Barbulescu, "A Simplification of the Modified Bahl Decoding Algorithm for Systematic Convolutional Codes," in *International Symposium on Information Theory and its Applications*, pp.1073–1077, Sydney, Australia, September 1994.

[46] J. R. Hess, *Implementation of a Turbo Decoder on a Configurable Computing Platform*, MSc dissertation, Virginia Polytechnic Institute and State University, September 1999.

[47] *ISE In-Depth Tutorial*, Xilinx Inc., 2003.

[48] D. L. Butler and P. H. Winne, "Feedback and Self-Regulated Learning: A Theoretical Synthesis," *Review of Educational Research*, Vol. 65, No. 3, pp. 245-281, Fall 1995.

[49] D. L. Jiang, *VHDL Program Design and Application*, ISBN 7-5635-0522-9/TP51, July 2001.

[50] *Xilinx ISE 9.1i Software Documentation*, Xilinx Inc.

[51] MATCOM Tutorial *http://luzhenbo.88uu.com.cn/Matlab/matlab_vc_matcom.htm*

[52] L. Bahl, F. Jelinek, J. Raviv and F. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Information Theory,* Vol. 20, pp. 284-287, March 1974.

[53] *ChipScope Pro Software and Cores User Guide*, UG029 (v 6.3.1), Xilinx Inc., October 2004.

Appendix A VHDL Code Samples

A.1 Receiver

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

entity receiver is

port (rst:in std_logic;
      clk:in std_logic;
      data_input:in std_logic;
      TX_control:out std_logic;
      data_bus:out std_logic_vector (7 downto 0);
      clk1x_test:out std_logic;
      sampling_test:out std_logic
      );
end receiver;

architecture v1 of receiver is

signal temp_data_input1: std_logic;
signal temp_data_input2: std_logic;
signal receive_control: std_logic;
signal clk_divder: unsigned (8 downto 0);
signal shift_register: unsigned (7 downto 0);
signal number_bits_receivered: unsigned (12 downto 0);
signal clk1x: std_logic;
signal temp_rst1: std_logic;
signal temp_rst2: std_logic;


begin

process (clk)

begin
    if clk'event and clk = '1' then
       temp_rst2 <= temp_rst1;
       temp_rst1 <= rst;
    end if ;
end process ;


process (temp_rst1,temp_rst2,clk,number_bits_receivered)

begin
   if (temp_rst1 = '1' and temp_rst2 = '0')or
       std_logic_vector(number_bits_receivered) =
       "1100000000000" then
```

```
      receive_control <= '0';
      temp_data_input1<='1'; --initialize
      temp_data_input2<='1';
   elsif clk'event and clk = '1' then
      temp_data_input2<=temp_data_input1;
      temp_data_input1<=data_input;
      if ( temp_data_input1 = '0' and temp_data_input2='1')
then
      receive_control <= '1' ;
      end if ;
   end if ;
end process ;


process (temp_rst1,temp_rst2,clk) --frequency divider

begin
    if temp_rst1 = '1' and temp_rst2 = '0'then
       clk_divder <= "000000000" ;
    elsif clk'event and clk = '1' then
       if receive_control='1' then
          clk_divder <= clk_divder + "000000001";
       end if;
    end if ;
end process ;

clk1x <= clk_divder(8);
clk1x_test <= clk_divder(8); --test


process (clk1x,temp_rst1,temp_rst2,number_bits_receivered)

begin
   if temp_rst1 = '1' and temp_rst2 = '0' then
      shift_register <= "00000000";  --initialize shift
register
      data_bus <= "ZZZZZZZZ";         --initialize data bus
      TX_control<='1';                  --initialize control
signal
   elsif clk1x'event and clk1x = '1' then
       if (std_logic_vector(number_bits_receivered) >=
          "0001000000000")and(std_logic_vector(number_bits_re
          ceivered) <= "0001111111111") then
          shift_register(7) <= data_input;
          sampling_test <= data_input;
          TX_control <='1';

       elsif (std_logic_vector(number_bits_receivered) >=
          "0010000000000")and(std_logic_vector(number_bits_rec
          eivered) <= "0010111111111") then
          shift_register(6) <=data_input;
          sampling_test <= data_input;
```

```
         elsif (std_logic_vector(number_bits_received) >=
             "0011000000000")and(std_logic_vector(number_bits_re
             ceiv ered) <= "0011111111111") then
             shift_register(5) <=data_input;
             sampling_test <= data_input;
         elsif (std_logic_vector(number_bits_received) >=
             "0100000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "0100111111111") then
             shift_register(4) <=data_input;
             sampling_test <= data_input;
        elsif (std_logic_vector(number_bits_received) >=
             "0101000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "0101111111111") then
             shift_register(3) <=data_input;
             sampling_test <= data_input;
        elsif (std_logic_vector(number_bits_received) >=
             "0110000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "0110111111111") then
             shift_register(2) <=data_input;
             sampling_test <= data_input;
        elsif (std_logic_vector(number_bits_received) >=
             "0111000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "0111111111111") then
             shift_register(1) <=data_input;
             sampling_test <= data_input;
        elsif (std_logic_vector(number_bits_received) >=
             "1000000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "1000111111111") then
             shift_register(0) <=data_input;
             sampling_test <= data_input;
        elsif (std_logic_vector(number_bits_received) >=
             "1001000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "1001111111111") then
             data_bus <= std_logic_vector(shift_register) ;
             TX_control <='0';
        elsif (std_logic_vector(number_bits_received) >=
             "1010000000000")and(std_logic_vector(number_bits_re
             ceivered) <= "1010111111111") then
             TX_control <='1';
        else
             data_bus <= "ZZZZZZZZ";
        end if;
   end if;
end process;


process
(temp_rst1,temp_rst2,clk,receive_control,number_bits_receivere
d)

begin
     if (temp_rst1 = '1' and temp_rst2 = '0') or
         receive_control = '0' then
         number_bits_received <= "0000000000000";
```

```
     elsif clk'event and clk='1' then --the clock is
115200Hz/s
         if receive_control='1' then
             number_bits_received <= number_bits_received +
             "0000000000001";
         end if;
     end if;
end process;

end;
```

A.2 Transmitter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity transmitter is
    port (rst :in std_logic ;
          clk : in std_logic ;
          TX_control_in : in std_logic ;
          data_bus_in : in std_logic_vector(7 downto 0);
          RTS : out std_logic ;
          data_output : out std_logic
          );
end transmitter;

architecture v1 of transmitter is

   signal transmit_control : std_logic;
   signal clk_divder : unsigned (8 downto 0);
   signal clk1x : std_logic ;
   signal number_bits_transmited : unsigned (3 downto 0);
   signal temp_TX_control_in1 : std_logic;
   signal temp_TX_control_in2 : std_logic;
   signal shift_register : std_logic_vector (7 downto 0);
   signal temp_rst1 : std_logic;
   signal temp_rst2 : std_logic;

begin

process (rst,clk)

begin
if clk'event and clk = '1' then
temp_rst2 <= temp_rst1;
temp_rst1 <= rst ;
end if ;
end process ;


process
(temp_rst1,temp_rst2,clk,TX_control_in,temp_TX_control_in1,tem
p_T  X_control_in2)

begin
    if temp_rst1 = '1' and temp_rst2 = '0' then
        temp_TX_control_in1 <= '1';
        temp_TX_control_in2 <= '1';
    elsif clk'event and clk = '1' then
        temp_TX_control_in2 <= temp_TX_control_in1;
        temp_TX_control_in1 <= TX_control_in;
    end if ;
end process ;
```

```
process (temp_rst1,temp_rst2,clk)

begin
    if temp_rst1 = '1' and temp_rst2 = '0' then --reset singal
        transmit_control <= '0'; --initialize
    elsif clk'event and clk = '1' then
      if temp_TX_control_in1 = '0' and temp_TX_control_in2 =
'1' then
          transmit_control <= '1';
      elsif std_logic_vector(number_bits_transmited) = "1101"
then
          transmit_control <= '0';
      end if ;
    end if ;

end process ;

process (temp_rst1,temp_rst2,clk,transmit_control)

begin
    if temp_rst1 = '1' and temp_rst2 = '0' then
        clk_divder <= "000000000";
    elsif clk'event and clk = '1' then
        if transmit_control = '1' then
            clk_divder <= clk_divder + "000000001";
        end if ;
    end if ;

end process ;

clk1x <= clk_divder(8);


process (temp_rst1,temp_rst2,clk1x,number_bits_transmited)

begin

   if temp_rst1 = '1' and temp_rst2 = '0' then
      data_output <= '1';
      RTS <= '1';
      shift_register <= "11111111";
   elsif clk1x'event and clk1x = '1' then
     if std_logic_vector(number_bits_transmited) = "0000" then
        shift_register <= data_bus_in;
        RTS <= '0';
        data_output <= '0';
      elsif std_logic_vector(number_bits_transmited) = "0001"
then
          data_output <= shift_register(7);
      elsif std_logic_vector(number_bits_transmited) = "0010"
then
          data_output <= shift_register(6);
```

```
     elsif std_logic_vector(number_bits_transmited) = "0011"
then
         data_output <= shift_register(5);
     elsif std_logic_vector(number_bits_transmited) = "0100"
then
         data_output <= shift_register(4);
     elsif std_logic_vector(number_bits_transmited) = "0101"
then
         data_output <= shift_register(3);
     elsif std_logic_vector(number_bits_transmited) = "0110"
then
         data_output <= shift_register(2);
     elsif std_logic_vector(number_bits_transmited) = "0111"
then
         data_output <= shift_register(1);
     elsif std_logic_vector(number_bits_transmited) = "1000"
then
         data_output <= shift_register(0);
     elsif std_logic_vector(number_bits_transmited) = "1001"
then
         data_output <= '1';
     elsif std_logic_vector(number_bits_transmited) = "1010"
then
         RTS<='1';
     end if;
   end if;

end process


process (temp_rst1,temp_rst2,clk1x,transmit_control)

begin
    if (temp_rst1 = '1' and temp_rst2 = '0') or
        transmit_control = '0' then
         number_bits_transmited <= "0000";
    elsif clk1x'event and clk1x = '1' then
        if transmit_control = '1' then
          number_bits_transmited <= number_bits_transmited +
"0001";
        end if ;
    end if;

end process;

end;
```

A.3 Encoder

```
procedure encode_bit
     (input    : in std_logic;
      state_in : in std_logic_vector  (1 to 2);
      output   : out std_logic_vector (1 to 2);
      state_out: out std_logic_vector (1 to 2)) is

      variable temp: std_logic_vector (1 to 2);

begin

 for i in 1 to n loop
   temp(i):= g(i,1) and input;
   for j in 2 to k loop
      temp(i):=temp(i) xor ( g(i,j) and state_in(j-1) );
   end loop;
 end loop;

output:=temp;
state_out:=input&state_in(m-1);

end encode_bit;


procedure rsc_encoder
    (x          : in std_logic_vector (1 to 10);
     terminated: in std_logic;
     y          : out array2_10) is

     variable state      : std_logic_vector(1 to 2);
     variable d_k        : std_logic;
     variable a_k        : std_logic;
     variable output_bit : std_logic_vector (1 to 2);
     variable state_out  : std_logic_vector(1 to 2);
begin

 for i in 1 to m loop
    state(i):='0';
 end loop;

 --generate the codeword
 for i in 1 to L_total loop

   if (terminated = '0' or (terminated ='1' and i<= L_info) )
then
       d_k:=x(i);
   elsif (terminated='1' and i> L_info) then
       --terminate the trellis
       d_k:= (g(1,2) and state(1))xor(g(1,3) and state(2));
   end if;

   a_k:=(g(1,1)and d_k)xor(g(1,2)and state(1))xor(g(1,3)and
        state(2));
```

```
    encode_bit(a_k,state,output_bit,state_out);
    state:=state_out;
    --since systematic, first output is input bit
    output_bit(1):=d_k;
    y(1,i):=output_bit(1);
    y(2,i):=output_bit(2);
  end loop;

end rsc_encoder;


PROCEDURE encoderm
  (infor_bit : in  std_logic_vector (1 to 8);
   en_output : out std_logic_vector (1 to 20)) is

   variable interleaved : std_logic_vector (1 to 10);
   variable input : std_logic_vector (1 to 10);
   variable output: array3_10;
   variable y :array2_10;
   variable en_temp:std_logic_vector(1 to 20);
  begin
   --determine the constraint length (K), memory (m)
   --and number of information bits plus tail bits
   input:="0000000000";

   for i in 1 to 8 loop
  input(i):=infor_bit(i);
  end loop;

  rsc_encoder(input,terminated1,y);

   --make a matrix with first row corresponing to infomation
sequence
   --secon row corresponsing to RSC #1's check bits.
   --third row corresponsing to RSC #2's check bits.

    for i in 1 to 10 loop
      output(1,i):= y(1,i);
      output(2,i):= y(2,i);
    end loop;
    --interleave input to second encoder
    interleaved(1):=output(1,1);
    interleaved(2):=output(1,4);
    interleaved(3):=output(1,7);
    interleaved(4):=output(1,10);
    interleaved(5):=output(1,2);
    interleaved(6):=output(1,5);
    interleaved(7):=output(1,8);
    interleaved(8):=output(1,3);
    interleaved(9):=output(1,6);
    interleaved(10):=output(1,9);

    rsc_encoder(interleaved,terminated0,y);
```

```
 for i in 1 to 10 loop
    output(3,i):=y(2,i);
 end loop;
 --paralell to serial mutiplex to get output vector;

 if puncture='1' then
   for i in 1 to 10 loop
     for j in 1 to 3 loop
        en_output(3*(i-1)+j):=output(j,i);
     end loop;
   end loop;

elsif puncture='0' then
   for i in 1 to L_total loop
      en_temp(n*(i-1)+1):=output(1,i);
      -- odd check bits from RSC1
      if ((i rem 2)=1) then
        en_temp(n*i):=output(2,i);
        --even check bits from RSC2
      else
        en_temp(n*i):=output(3,i);
      end if;
   end loop;
end if;


 en_output:= en_temp;

end encoderm;
```

A.4 SOVA model

```
PROCEDURE sova
  (rsc     : in xin_dao;
   ind_dec : in std_logic;
   L_e_in  : in La_data;
   L_e_out : out La_data;
   L_all   : out La_data) is

   variable path_metric: path_metric_type;
   variable Mdiff : path_metric_type;
   variable prev_bit : prev_bit_type;
   variable rsc_s :xin_dao;
   variable L_e :La_data;
   variable L_a :La_data;
   variable terminated :std_logic;
   variable iter_L_all :La_data;
 begin

   rsc_s:=rsc;
   L_e:=L_e_in;
   if ind_dec='1' then
      interleave(L_e,L_a);
      terminated:='1';
   else
      deinterleave(L_e,L_a);
      terminated:='0';
   end if;

   sova_part(rsc_s,L_a,path_metric,Mdiff,prev_bit);

sova_part1(path_metric,Mdiff,prev_bit,terminated,iter_L_all);
   L_all:=iter_L_all;
   for i in 1 to 10 loop
      L_e(i):= iter_L_all(i)-rsc_s(2*i-1)-L_a(i);
   end loop;
   L_e_out:=L_e;
 end sova;



PROCEDURE sova_part1
   (path_in   : in path_metric_type;
    Mdiff_in  : in path_metric_type;
    prev_in   : in prev_bit_type;
    ind_dec   : in std_logic;
    L_all     : out La_data) is

    variable path_metric : path_metric_type;
    variable Mdiff       : path_metric_type;
    variable Mdiff_temp  : unit;
    variable prev_bit    : prev_bit_type;
    variable est_integer : last_unit ;
    variable state0      : unit;
```

```
    variable state1      : unit;
    variable mlstate       : mlstate_type;
    variable compare_state : compare_state_type;
    variable max_state     : state_unit;
    variable temp_state    : state_unit;
    variable est           : est_type;
    variable llr           : unit;
    variable ind_dec1      : std_logic;

 begin

     path_metric:=path_in;
     Mdiff:=Mdiff_in;
     prev_bit:=prev_in;

   if ind_dec ='1' then
      mlstate(11):=1;
      ind_dec1:='1';
   else
      for i in 1 to 4 loop
         compare_state(i):=path_metric(i,11);
      end loop;
      -- max_state(compare_state,max_state);
      mlstate(11):=max_state;
      ind_dec1:='0';
   end if;

 --Trace back to get the estimated bits, and the most likely
path
   for t in 10 downto 1 loop
      est(t):= prev_bit(mlstate(t+1),(t+1));
      mlstate(t):= last_state(mlstate(t+1),(est(t)+1));
   end loop;

 --Find the minimum delta that corresponds to a compitition
   path with different info. bit estimation.
 --Give the soft output

   for t in 1 to 10 loop
      if ind_dec1 ='1' then
         llr:=c10;
         for i in 0 to 5 loop
            if t+i<11 then
              est_integer:=1-est(t+i);

temp_state:=last_state(mlstate(t+i+1),(est_integer+1));
              for j in (i-1) downto 0 loop
                 est_integer:=prev_bit(temp_state,(t+j+1));

temp_state:=last_state(temp_state,(est_integer+1));
              end loop;

              if est_integer/=est(t) then
                 if llr> Mdiff(mlstate(t+i+1),(t+i+1)) then
```

```
                    llr:=Mdiff(mlstate(t+i+1),(t+i+1));
                 else
                    llr:=llr;
                 end if;
              end if;
           end if;
        end loop;
     elsif ind_dec1='0' then
           llr:=c1;
     end if;

     if est(t)=1 then
        L_all(t):=llr;
     else
        L_all(t):=-llr;
     end if;
  end loop;

end sova_part1;


PROCEDURE sova_part
  (rec_s     : in xin_dao;
   L_a       : in La_data;
   path_out  : out path_metric_type;
   Mdiff_out : out path_metric_type;
   prev_out  : out prev_bit_type) is

   variable path_metric  : path_metric_type;
   variable Mdiff        : path_metric_type;
   variable Mdiff_temp   : unit;
   variable prev_bit     : prev_bit_type;
   variable y1           : unit;
   variable y2           : unit;
   variable sym1         : unit;
   variable sym2         : unit;
   variable sym3         : unit;
   variable sym4         : unit;
   variable path1        : unit;
   variable path2        : unit;
   variable Mk0          : unit;
   variable Mk1          : unit;
   variable negL_a       : unit;
   variable posL_a       : unit;

 begin
   --SOVA window size. Make decision after 'delta' delay.
     Decide bit k when received bits
   --for bit (k+delta) are processed. Trace back from (k+delta)
to k.
   for t in 1 to 11 loop
      for state in 1 to 4 loop
         path_metric(state,t):=c_0;
         prev_bit(state,t):=0;
```

```
   end loop;
end loop;
--Trace forward to compute all the path metrics

for t in 1 to 10 loop
  for state in 1 to 4 loop
      y1:=rec_s(2*t-1);
      y2:=rec_s(2*t);

      if state=1 then
         sym1:=-y1;
         sym2:=-y2;
         sym3:=y1;
         sym4:=y2;
         path1:=path_metric(1,t);
         path2:=path_metric(2,t);
      elsif state=2 then
         sym1:=-y1;
         sym2:=y2;
         sym3:=y1;
         sym4:=-y2;
         path1:=path_metric(4,t);
         path2:=path_metric(3,t);
      elsif state=3 then
         sym1:=-y1;
         sym2:=-y2;
         sym3:=y1;
         sym4:=y2;
         path1:=path_metric(2,t);
         path2:=path_metric(1,t);
      else
         sym1:=-y1;
         sym2:=y2;
         sym3:=y1;
         sym4:=-y2;
         path1:=path_metric(3,t);
         path2:=path_metric(4,t);
      end if;

      Mk0:=sym1+sym2-L_a(t)+path1;
      Mk1:=sym3+sym4+L_a(t)+path2;

      if Mk0>Mk1 then
          path_metric(state,(t+1)):=Mk0;
         Mdiff(state,(t+1)):= Mk0-Mk1;
         prev_bit(state,(t+1)):=0;
      else
         path_metric(state,(t+1)):=Mk1;
         Mdiff(state,(t+1)):= Mk1-Mk0;
         prev_bit(state,(t+1)):=1;
      end if;
   end loop;
end loop;
path_out:=path_metric;
```

```
    Mdiff_out:=Mdiff;
    prev_out:=prev_bit;
end sova_part;
```

Appendix B C++ Code Samples

B.1 Bit Error Ratio Plot

```
if (!(ep = engOpen("\0")))
{
    fprintf(stderr, "\nCan't start MATLAB engine\n");
}

X=mxCreateDoubleMatrix(1,4, mxREAL);
T=mxCreateDoubleMatrix(8,1, mxREAL);
  //mxSetName("T",T);
memcpy((void *)mxGetPr(T), (void *)time, sizeof(time));
memcpy((void *)mxGetPr(X), (void *)dBN0, sizeof(dBN0));
engPutVariable(ep,"ber1",T);
engPutVariable(ep,"dBN0",X);
engEvalString(ep,"ber(1,1)=ber1(4);");
engEvalString(ep,"ber(2,1)=ber1(3);");
engEvalString(ep,"ber(3,1)=ber1(2);");
engEvalString(ep,"ber(4,1)=ber1(1);");
engEvalString(ep,"ber(1,2)=ber1(8);");
engEvalString(ep,"ber(2,2)=ber1(7);");
engEvalString(ep,"ber(3,2)=ber1(6);");
engEvalString(ep,"ber(4,2)=ber1(5);");
engEvalString(ep,"semilogy(dBN0,ber,'LineWidth',2);");
engEvalString(ep,"title('FPGA PERFORMANCE');");
engEvalString(ep,"Grid on");
engEvalString(ep,"xlabel('Eb/N0(db)');");
engEvalString(ep,"ylabel('Bit Error Ratio');");
mxDestroyArray(T);
mxDestroyArray(X);
```

B.2 Communication Function

```
LONG CSCOMMDlg::OnCommunication(WPARAM ch, LPARAM port)
{
   if (port <= 0 || port > 4)
       return -1;
       rxdatacount++;
   if(m_ctrlHexSend.GetCheck())
   {
      rxcountHEX++;
      CString temp1;
      CString temp2;
      CString temp3;
      CString m_str;
      m_str+=m_filedata;
      int hownumberHEX=(long)((m_str.GetLength()+1)/3);
      if (rxcountHEX<hownumberHEX)
      {
          temp1=m_str.GetAt(3*rxcountHEX);
          temp2=m_str.GetAt(3*rxcountHEX+1);
          temp3+=temp1 + temp2;
          char data[1024];
          int len=Str2Hex(temp3,data);
          m_Port.WriteToPort(data,len);
       }
   }

   CString strTemp;
   strTemp.Format("%ld",rxdatacount);
   strTemp="RX:"+strTemp;
   m_ctrlRXCOUNT.SetWindowText(strTemp);
   if(m_bStopDispRXData)
   return -1;
   if((m_ctrlAutoClear.GetCheck())&&(m_ctrlReceiveData.GetLine
      Count()>=50))
  {
     m_ReceiveData.Empty();
     UpdateData(FALSE);
  }

  if(m_ctrlReceiveData.GetLineCount()>400)
    {
      m_ReceiveData.Empty();
      m_ReceiveData="***The Length of the Text is too long,
      Emptied Automaticly!!!***\r\n";
      UpdateData(FALSE);
    }

  CString str;
  if(m_ctrlHexReceieve.GetCheck())
     str.Format("%02X ",ch);
  else
     str.Format("%c",ch);
```

```
    int nLen=m_ctrlReceiveData.GetWindowTextLength();
    m_ctrlReceiveData.SetSel(nLen, nLen);
    m_ctrlReceiveData.ReplaceSel(str);
    nLen+=str.GetLength();
    m_ReceiveData+=str;
    return 0;
}
```

B.3 Bit Error Ratio Calculation

```
void CSCOMMDlg::OnButtonBer()
{
   int Txlength=m_nTxBer.GetLength();
   int Rxlength=m_nRxBer.GetLength();
   float Bercount=0;
   float Berbitcount=0;
   int framsize=Rxlength/3;
   CString temp1Tx;
   CString temp2Tx;
   CString temp3Tx;
   CString temp1Rx;
   CString temp2Rx;
   CString temp3Rx;
   CString RxBit;
   CString TxBit;
   double BER;
   double BER1;

   for( int i= 0; i<framsize; i++ )
   {
     temp1Tx=m_nTxBer.GetAt(3*i);
     temp2Tx=m_nTxBer.GetAt(3*i+1);
     temp3Tx=temp1Tx + temp2Tx;
     temp1Rx=m_nRxBer.GetAt(3*i);
     temp2Rx=m_nRxBer.GetAt(3*i+1);
     temp3Rx=temp1Rx + temp2Rx;
     CString outputstr;
     RxBit=Str2Str(temp3Rx,outputstr);
     TxBit=Str2Str(temp3Tx,outputstr);
     int RxBitlength=RxBit.GetLength();

     for (int t=0; t< RxBitlength; t++)
     {
     if (RxBit[t]!=TxBit[t])
        Berbitcount=Berbitcount+1;
     }

     if (temp3Tx!=temp3Rx)
        Bercount=Bercount+1;
   }

   BER=Bercount/(framsize);

   switch(m_ninter)
   {
   case 66:
      switch(m_nber)
      {
      case 97:
         switch(m_npunc)
         {
        case 78:
```

```
        switch(m_niter)
        {
        case 49:
          BER1=Berbitcount/(framsize*8);
          TEMP[0]=BER1;
          m_TEMPTEST.Format("%f",TEMP[0]);
          m_Display_Fram.Format("%d",framsize);
          m_Display_Ber1.Format("%f",TEMP[0]);
          UpdateData(FALSE);
          break;
        case 50:
          BER1=Berbitcount/(framsize*8);
          TEMP[4]=BER1;
          m_TEMPTEST.Format("%f",TEMP[4]);
          m_Display_Fram.Format("%d",framsize);
          m_Display_Ber1.Format("%f",TEMP[4]);
          UpdateData(FALSE);
          break;
        }
        break;
    }
    break;
 case 98:
    switch(m_npunc)
    {
    case 78:
        switch(m_niter)
        {
        case 49:
          BER1=Berbitcount
          TEMP[1]=BER1;
          m_TEMPTEST.Format("%f",TEMP[1]);
          m_Display_Fram.Format("%d",framsize);
          m_Display_Ber1.Format("%f",TEMP[1]);
          UpdateData(FALSE);
        break;
        case 50:
          BER1=Berbitcount/(framsize*8);
          TEMP[5]=BER1;
          m_TEMPTEST.Format("%f",TEMP[5]);
          m_Display_Fram.Format("%d",framsize);
          m_Display_Ber1.Format("%f",TEMP[5]);
          UpdateData(FALSE);
        break;
    }
    break;
}
break;
case 99:
    switch(m_npunc)
    {
    case 78:
        switch(m_niter)
        {
```

```
            case 49:
              BER1=Berbitcount/(framsize*8);
              TEMP[2]=BER1;
              m_TEMPTEST.Format("%f",TEMP[2]);
              m_Display_Fram.Format("%d",framsize);
              m_Display_Ber1.Format("%f",TEMP[2]);
              UpdateData(FALSE);
            break;
            case 50:
              BER1=Berbitcount/(framsize*8);
              TEMP[6]=BER1;
              m_TEMPTEST.Format("%f",TEMP[6]);
              m_Display_Fram.Format("%d",framsize);
              m_Display_Ber1.Format("%f",TEMP[6]);
              UpdateData(FALSE);
            break;
          }
          break;
        }
        break;
      case 100:
         switch(m_npunc)
         {
         case 78:
             switch(m_niter)
             {
             case 49:
               BER1=Berbitcount/(framsize*8);
               TEMP[3]=BER1;
               m_TEMPTEST.Format("%f",TEMP[3]);
               m_Display_Fram.Format("%d",framsize);
               m_Display_Ber1.Format("%f",TEMP[3]);
               UpdateData(FALSE);
             break;
             case 50:
               BER1=Berbitcount/(framsize*8);
               TEMP[7]=BER1;
               m_TEMPTEST.Format("%f",TEMP[7]);
               m_Display_Fram.Format("%d",framsize);
               m_Display_Ber1.Format("%f",TEMP[7]);
               UpdateData(FALSE);
             break;
             }
          break;
          }
     break;
     }
break;
}
temp1[0]=TEMP[0];
temp1[1]=TEMP[1];
temp1[2]=TEMP[2];
temp1[3]=TEMP[3];
temp1[4]=TEMP[4];
```

```
    temp1[5]=TEMP[5];
    temp1[6]=TEMP[6];
    temp1[7]=TEMP[7];
}
```

B.4 GUI Initialization

```
BOOL CSCOMMDlg::OnInitDialog()
{
    BOOL b = CDialog::OnInitDialog();
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
      {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
          {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
            strAboutMenu);
          }
      }
      SetIcon(m_hIcon, TRUE);
      SetIcon(m_hIcon, FALSE);

      m_Com.SetCurSel(0);
      m_Speed.SetCurSel(11);
      m_Parity.SetCurSel(0);
      m_DataBits.SetCurSel(0);
      m_StopBits.SetCurSel(0);
      m_iter.SetCurSel(0);
      m_punc.SetCurSel(0);
      m_ber.SetCurSel(0);
      m_inter.SetCurSel(0);
      m_diedai.SetCurSel(0);
      m_fram.SetCurSel(0);
      m_algorithm.SetCurSel(0);
      m_jiaoziqi.SetCurSel(0);
      m_chuanci.SetCurSel(0);
      m_study.SetCurSel(0);
      m_example.SetCurSel(0);
      m_nBaud=115200;
      m_nCom=1;
      m_cParity='N';
      m_nDatabits=8;
      m_nStopbits=1;
      m_dwCommEvents = EV_RXFLAG | EV_RXCHAR;
      CString strTurboStatus;
      m_niter='1';
      m_npunc='N';
      m_nber='a';
      m_ninter='B';
      strTurboStatus.Format("TurboCode: %c Iteration,%c BER,%c
      Interleaver,%c  Puncture",m_niter,m_nber,m_ninter,m_npunc);
      m_ctrlTurboStatus.SetWindowText(strTurboStatus);

      CString strTurboStatus2;
```

```
        m_ndiedai='1';
        m_nfram='1';
        m_nalgorithm='M';
        m_njiaoziqi='R';
        m_nchuanci='U';
        strTurboStatus2.Format("TurboCode: %c Iteration,%c
        FRAM,%c Algorithm,%c Interleaver,%c Puncture",m_ndiedai,
        m_nfram,m_nalgorithm,m_njiaoziqi,m_nchuanci);
        m_ctrlTurboStatus2.SetWindowText(strTurboStatus2);

        CString strStatus;
        if (m_Port.InitPort(this, m_nCom,
             m_nBaud,m_cParity,m_nDatabits,m_nStopbits,m_dwCommEv
             ents,1024))
          {
             m_Port.StartMonitoring();
             strStatus.Format("STATUS: COM%d OPENED,
             %d,%c,%d,%d",m_nCom,m_nBaud,m_cParity,m_nDatabits,m_
             nStopbits);
             m_ctrlIconOpenoff.SetIcon(m_hIconRed);
          }
        else
          {
             AfxMessageBox("can't find port");
             m_ctrlIconOpenoff.SetIcon(m_hIconOff);
          }
             m_ctrlPortStatus.SetWindowText(strStatus);

             CEdit* pEdit=(CEdit*)GetDlgItem(IDC_EDIT_CYCLETIME);
             CString strText;
             strText.Format("%d",m_nCycleTime);
             pEdit->SetWindowText(strText);


             m_ctrlAutoClear.SetCheck(0);
             m_ctrlHexReceieve.SetCheck(1);
             m_ctrlHexSend.SetCheck(1);

             m_ctrlEditSendFile.SetWindowText("No File
             Selected!");
             m_animIcon.SetImageList(IDB_BITMAP1,3,RGB(0,0,0));

             SetTimer(4,200,NULL);

             UpdateData(FALSE);

             ShowWindow(SW_SHOW);
}
```

B.5 Transmitter File

```
void CSCOMMDlg::OnButtonSendfile()
{
  CFile fp;
  if(!(fp.Open((LPCTSTR)m_strSendFilePathName,CFile::modeRead)))
     {
     AfxMessageBox("File Can not Open!");
     return;
     }
     fp.SeekToEnd();
     unsigned long fplength=fp.GetLength();
     m_nFileLength=fplength;
     char* fpBuff;
     fpBuff=new char[fplength];
     fp.SeekToBegin();
     if(fp.Read(fpBuff,fplength)<1)
     {
        fp.Close();
        return;
     }
     fp.Close();

     m_nTxBer=(LPCTSTR)fpBuff;
     m_ctrlBer.EnableWindow(FALSE);
     CString strStatus;
     if (m_Port.InitPort(this, m_nCom, m_nBaud, m_cParity,
          m_nDatabits, m_nStopbits, m_dwCommEvents, fplength))
       {
          m_Port.StartMonitoring();
          m_ctrlIconOpenoff.SetIcon(m_hIconRed);
          m_bSendFile=TRUE;
          m_strTempSendFilePathName=m_strSendFilePathName;
          m_ctrlEditSendFile.SetWindowText("Send Now.....");
          m_ctrlManualSend.EnableWindow(FALSE);
          m_ctrlAutoSend.EnableWindow(FALSE);
          m_ctrlSendFile.EnableWindow(FALSE);

          if(m_ctrlHexSend.GetCheck())
            {
              CString texpx;
              CString temp1;
              CString temp2;
              CString temp3;
              CString temp4;
              rxcountHEX =0;
              temp1=(LPCTSTR)fpBuff;
              m_filedata=temp1;
              temp2=temp1.GetAt(0);
              temp3=temp1.GetAt(1);
              temp4+=temp2+temp3;
              char data[4096];
              int len=Str2Hex(temp4,data);
              m_Port.WriteToPort(data,len);
```

```
            }
            else
            {
            m_nTxBer=(LPCTSTR)fpBuff;
            m_Port.WriteToPort((LPCTSTR)fpBuff,fplength);
            }
}
```

B.6 FPGA Programming

```
void CSCOMMDlg::OnButtonProgram()
{
    TCHAR exeFullPath[MAX_PATH];
    GetModuleFileName(NULL,exeFullPath,MAX_PATH);
    CString strlpPath;
    strlpPath.Format("%s",exeFullPath);
    strlpPath.MakeUpper();
    strlpPath.Replace("TURBOCODE.EXE","");

    switch(m_ninter)
    {
    case 66:
        switch(m_nber)
        {
        case 97:
             switch(m_npunc)
             {
             case 78:
                  switch(m_niter)
                  {
                  case 49:
                      ShellExecute(NULL,NULL,_T("I1B5.bat"),NULL
                      ,strlpPath,SW_SHOW);
                      break;
                  case 50:
                      ShellExecute(NULL,NULL,_T("I2B5.bat"),NULL
                      ,strlpPath,SW_SHOW);
                      break;
                  }
                  break;
             }
             break;
        case 98:
             switch(m_npunc)
             {
             case 78:
                  switch(m_niter)
                  {
                  case 49:
                      ShellExecute(NULL,NULL,_T("I1B15.bat"),NUL
                      L,strlpPath,SW_SHOW);
                      break;
                  case 50:
                      ShellExecute(NULL,NULL,_T("I2B15.bat"),NUL
                      L,strlpPath,SW_SHOW);
                      break;
                  }
                  break;
             }
             break;
        case 99:
             switch(m_npunc)
```

```
            {
            case 78:
                switch(m_niter)
                {
                case 49:
                    ShellExecute(NULL,NULL,_T("I1B25.bat"),NULL,
                    strlpPath,SW_SHOW);
                    break;
                case 50:
                    ShellExecute(NULL,NULL,_T("I2B25.bat"),NULL,
                    strlpPath,SW_SHOW);
                    break;
                }
                break;
            }
            break;
        case 100:
            switch(m_npunc)
            {
            case 78:
                switch(m_niter)
                {
                case 49:
                    ShellExecute(NULL,NULL,_T("I1B35.bat"),NULL,
                    strlpPath,SW_SHOW);
                    break;
                case 50:
                    ShellExecute(NULL,NULL,_T("I2B35.bat"),NULL,
                    strlpPath,SW_SHOW);
                    break;
                }
                break;
            }
            break;
    }
}
```

B.7 Combo Boxes

```
void CSCOMMDlg::OnSelendokCOMBOjiaoziqi()
{
   char jiaoziqi;
   int i=m_jiaoziqi.GetCurSel();
   switch(i)
   {
    case 0:
        jiaoziqi='R';
        break;
    case 1:
        jiaoziqi='B';
        break;
   }
    m_njiaoziqi=jiaoziqi;
    CString strTurboStatus2;
    m_ctrlTurboStatus2.SetWindowText(strTurboStatus2);
}


void CSCOMMDlg::OnSelendokComboFram()
{
   char fram;
   int i=m_fram.GetCurSel();
   switch(i)
   {
    case 0:
        fram='3';
        break;
    case 1:
        fram='1';
        break;
    case 2:
        fram='4';
        break;
   }
    m_nfram=fram;
    CString strTurboStatus2;
    m_ctrlTurboStatus2.SetWindowText(strTurboStatus2);
}


void CSCOMMDlg::OnSelendokCOMBOchuaici()
{
    char chuanci;
    int i=m_chuanci.GetCurSel();
    switch(i)
    {
    case 0:
         chuanci='P';
         break;
    case 1:
         chuanci='P';
```

```
        break;
   }
    m_nchuanci=chuanci;
    CString strTurboStatus2;
    m_ctrlTurboStatus2.SetWindowText(strTurboStatus2);
}

void CSCOMMDlg::OnSelendokCOMBOalgorithm()
{
    char algorithm;
    int i=m_algorithm.GetCurSel();
    switch(i)
    {
     case 0:
         algorithm='M';
         break;
     case 1:
         algorithm='S';
         break;
    }
    m_nalgorithm=algorithm;
    CString strTurboStatus2;
    m_ctrlTurboStatus2.SetWindowText(strTurboStatus2);
}

void CSCOMMDlg::OnSelendokCOMBOdiedai()
{
    char iter;
    int i=m_diedai.GetCurSel();
    switch(i)
    {
    case 0:
        iter='1';
        break;
    case 1:
        iter='2';
        break;
    case 2:
        iter='3';
        break;
    case 3:
        iter='4';
        break;
    }
    m_ndiedai=iter;
    CString strTurboStatus2;
    m_ctrlTurboStatus2.SetWindowText(strTurboStatus2);
}
```

Appendix C Circuit Diagrams

C.1 Receiver

C.2 Transmitter

C.3 Encoder