2005

# The design of an asynchronous BCJR/MAP convolutional channel decoder.

Kristofer Patrick Perta
*University of Windsor*

# NOTE TO USERS

# The Design of an Asynchronous BCJR/MAP Convolutional Channel Decoder

by

**Kristofer Patrick Perta**

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the Department of Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada
2004

*Your file Votre référence*
*ISBN: 0-494-00498-3*
*Our file Notre référence*
*ISBN: 0-494-00498-3*

The Design of an Asynchronous BCJR/MAP Convolutional Channel Decoder

by

Kristofer Patrick Perta

APPROVED BY:

Dr. R. Lashkari, External Examiner
Department of Industrial and Manufacturing Systems Engineering

Dr. H. Wu, Internal Reader
Department of Electrical and Computer Engineering

Dr. K. Tepe, Supervisor
Department of Electrical and Computer Engineering

Dr. B. Shahrrava, Chair
Department of Electrical and Computer Engineering

December 17, 2004

# *Abstract*

The digital design alternative to the everyday synchronous circuit design paradigm is the asynchronous model. Asynchronous circuits are also known as handshaking circuits and they may prove to be a feasible design alternative in the modern digital Very Large Scale Integration (VLSI) design environment. Asynchronous circuits and systems offer the possibility of lower system power requirements, reduced noise, elimination of clock skew and many other benefits.

Channel coding is a useful means of eliminating erroneous transmission due to the communication channel's physical limits. Convolutional coding has come to the forefront of channel coding discussions due to the usefulness of turbo codes.

The niche market for turbo codes have typically been in satellite communication. The usefulness of turbo codes are now expanding into the next generation of handheld communication products. It is probable that the turbo coding scheme will reside in the next cellular phone one purchases [1].

Turbo coding uses two BCJR decoders in its implementation. The BCJR decoding algorithm was named after its creators Bahl, Cocke, Jelinek, and Raviv (BCJR). The BCJR algorithm is sometimes known as a Maximum Priori Posteriori (MAP) algorithm. This means a very large part of the turbo coding research will encompass the BCJR/MAP decoder and its optimization for size, power and performance.

An investigation into the design of a BCJR/MAP convolutional channel decoder will

be introduced. This will encompass the use and synthesis of an asynchronous Hardware Definition Language (HDL) called Balsa. The design will be carried through to the gate implementation level. Proper gate level analysis will identify the key metrics that will determine the feasibility of an asynchronous design of that of the everyday clocked paradigm.

# *Acknowledgments*

There are many people I would like to offer thanks and appreciation. The many people that have helped me along the way have proven to be titanic and priceless.

First and foremost, I would like to thank my advisor for truly caring and making these 2 years fly by. Dr. Tepe's expert guidance and colossal technical expertise paved the way for a truly enriched and pleasant graduate program.

I like to extend my thanks to the RCIM group, especially Till Kuendiger for his endless patience and tremendous expertise. I would also like to offer thanks to my committee members, Dr. Huapeng Wu and Dr. Reza Lashkari for their patience and guidance.

To my friends, especially my University of Windsor fellow alumni, Pedram Mokrian, Mike Howard, Alan Soltis, Collin Hayes, Marianne Dent, Colleen Middaugh and Rita Turchi. I would like to say thanks for helping in every conceivable way and giving me great advice.

I'd like to give a warm thanks to my family for putting up with me for the past 2 years and motivating me to get things done.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# *Introduction*

## 1.1 Asynchronous Circuits And Systems

The digital VLSI designers are faced with many challenges. Some of the biggest challenges they face include [2]:

- Lowering power consumption
- Addressing clock skew issues
- Decreasing noise
- Increasing performance

These challenges have been in existence for some time. They will become increasingly prevalent in future digital VLSI designs due to scaling issues with technology. There are additional concerns that need to be addressed; however, the issues stated above, are staples in the designer's healthy diet of problems.

To overcome these current obstacles, temporary fixes that simply patch up the greater need for a concise solution are currently employed. The many advantages that asynchronous

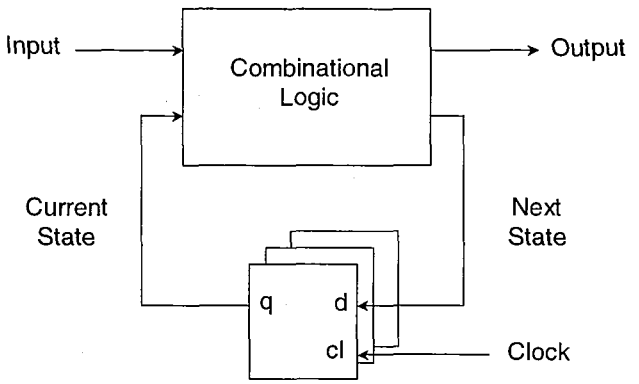



Figure 1.1: Synchronous Circuit [2]

circuits and systems have, e.g., lower power and noise systems than in synchronous systems, are an attractive alternative and the key to the reduction of VLSI design problems.

### 1.1.1 Fundamental Asynchronous Concepts

To understand the fundamental concepts of asynchronous circuits and systems, one must first examine a synchronous circuit shown in Figure 1.1.

The synchronous circuit works in the following way [2]:

1. The current state is stored in an array of registers.
2. The next state is then computed from the current state and the input to the combinational logic circuit.
3. When the clock makes a transition from low to high, registers are enabled and the next state gets copied into the register. Thus, becoming the next state.

An important note to make is that the clock period depends on the length of time that the combinational logic circuit takes to compute the input and deliver an output.

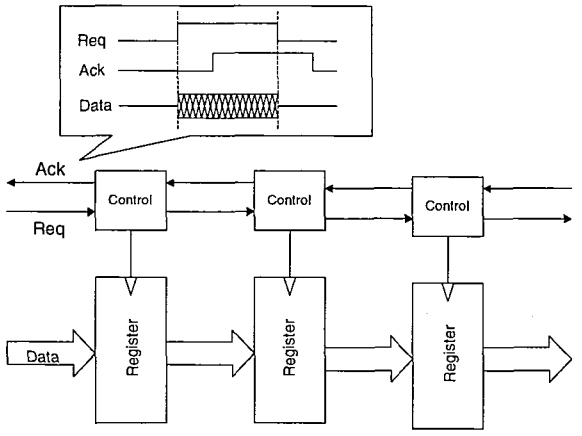



Figure 1.2: Asynchronous Circuit [3]

In an asynchronous system, instead of having a clock as the main synchronizer, coordination is accomplished through the use of acknowledge (ack) and request (req) signals. This is also known as handshake signaling.

In the asynchronous system in Figure 1.2, each stage of the system contains a register and control circuitry. The control circuit communicates with the preceding and succeeding stages by handshake signaling. This in turn controls the state of the registers, i.e., transparent (open) or opaque (closed), which allows the input to pass towards the next register and so forth [2]. The different protocols will be discussed in the next chapter.

### 1.1.2 Benefits Of Asynchronous Systems

Asynchronous Systems (AS) offer many advantages over the common clocked Synchronous System (SS) design. According to [4], these advantages are as follows, but not limited to:

1. **Elimination of Clock Skew** - With the need for larger, more complex integrated systems, the synchronization of the clock's arrival time, at different areas of the circuit, is increasingly more difficult to achieve. AS utilizes handshaking protocols to achieve

the synchronization needed within its circuitry, thus eliminating the clock.

2. **Average Case Performance** - SS performance is dictated by worst-case condition. AS performance is dictated by average case conditions, which may lead to increases in performance.

3. **Adaptivity to Processing and Environmental Variations** - SS have their clock rate set to allow for correct operation under some allowed variation. AS operate under all variations and simply speed up or slow down as necessary.

4. **Component Modularity and Reuse** - AS have no need to synchronize with a global clock, therefore simple system integration with other sub-systems and reuse are simpler than with SS.

5. **Lower System Power Requirements** - Additional clock networks are not needed in AS. 15-45 % of electrical power consumed by a SS chip must be devoted to the clock network [5]. There is no waste of energy during spurious transitions, reducing global dynamic power consumption.

6. **Reduced Noise** - Activity is uncorrelated in AS, resulting in a more distributed noise spectrum and lower peak noise values.

### 1.1.3 Recent Developments In Asynchronous Applications

There are many facets that encompass asynchronous circuit design. However, the idea that asynchronous circuits alone are going to revolutionize VLSI design is a false presumption.

The amalgamation of the asynchronous and synchronous conceptual frameworks will offer promising solutions such as Central Processing Units (CPU) that are synchronous internally, yet communicate asynchronously with memory [6].

An important advantage that AS have is their ability to be modular, meaning that they are easily incorporated into other synchronous or asynchronous systems. Commercially, this has been shown with the use of asynchronous circuits in the UltraSPARC IIIi synchronous processor at Sun Microsystems [7].

In the field of communications, Brackenbury et al. [8] have proposed that asynchronous techniques can be used in VLSI implementations of communication systems yielding lower power consumption, specifically the Viterbi decoder. Their design yields 23-29 % less area than a selection of other synchronous implementations with the same design parameters which use the same fabrication process and cell library.

## 1.2 Channel Coding

In a communication system, the aim is to transmit information from a source to a target. A communication engineer faces many challenges. According to [9], these challenges are as follows, but not limited to:

- Thermal noise
- Changes in signal power
- Short losses of signal power (Erasure)

To tackle these challenges a useful tool is channel coding. When performing channel coding in a communication system, redundant information is introduced into the original signal. This redundancy, at the receiving end, can aid in recovering the original signal that may have been corrupted during transmission.

### 1.2.1 Basic Concepts

C.E. Shannon [9] demonstrated that channel coding helps realize "reliable" communication over noisy channels. Many landmark achievements have been proposed and used, for instance Convolutional Coding (CC). CC is an error correction channel coding technique that differs from the block coding technique. In block coding, the data stream is divided into a number of blocks and each block is encoded into a code word. In CC, the entire data stream is encoded into a code word.

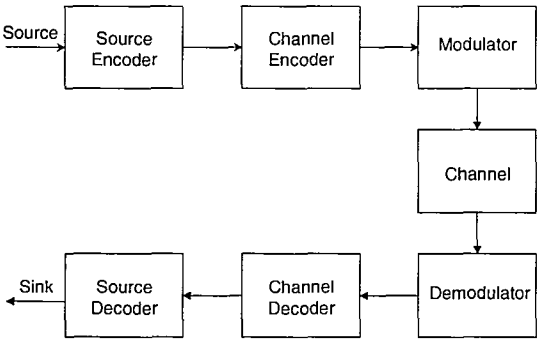



Figure 1.3: Basic Digital Communication System

Coding is a two part process, which involves the encoding of the original information and then followed by decoding at the receiving end. Figure 1.3 shows the system diagram of a basic digital communication system [9].

First, the source produces a message, a digital signal, which will be transmitted to a particular target. If the source produces an analog signal, that signal needs to be made digital.

The source encoder removes most redundancies from the message. This is done so that the efficiency of the overall system increases.

The channel encoder inserts redundancies in a controlled manner so that at the receiver, the system can correct or detect any errors that may have been introduced by the channel or the receiver's front end.

The modulator gives the coded information a new form so that it has the best chance to pass through the channel with as little error as possible.

After passing through the channel, the demodulator, channel decoder and source decoder reverse the original operation of their equivalent partner to produce the original message.

### 1.2.2 Channel Coding Techniques

With channel coding, the main source of importance lies with the decoder. This is because the decoder is more complex than the encoder. The research that has been done has largely comprised the decoder.

There are many different types of decoding algorithms that are used for convolutional codes. The most popular are the Viterbi Algorithm (VA) and BCJR/MAP. The Viterbi is a maximum-likelihood decoding algorithm [9]. The BCJR is known as a symbol-by-symbol 'Maximum A Posteriori' (MAP) algorithm. The main reason that we use the BCJR/MAP is because it can calculate soft information for the output symbols, which is used in iterative decoding schemes like Turbo Codes [9], [1].

As mentioned earlier, Brackenbury et al. [8] have proposed that asynchronous techniques can be used with application to VLSI implementations of communication systems, specifically the Viterbi decoder.

Applying asynchronous techniques, as previously mentioned, can conceivably advance or offer a feasible design alternative where synchronous techniques cannot. Asynchronous techniques, as seen in [8], can offer the potential for lower power systems.

The natural extension of the work done at the University of Manchester with the asynchronous decoder is to apply asynchronous techniques to other decoding algorithms such as the BCJR/MAP. This thesis investigates the design of an asynchronous convolutional decoder using the BCJR/MAP algorithm.

# Chapter 2

# *Asynchronous Circuits And Systems*

## 2.1 Introduction

To fully understand asynchronous circuitry, the different types of protocols and building blocks used must be examined.

The following chapter examines the various types of protocols used and the building blocks required for the implementation of these protocols. This chapter will assist in the knowledge required to understand asynchronous circuits and systems.

## 2.2 Bundled Data (BD) Or Single Rail (SR) Protocols

The two main protocols in asynchronous circuits and systems are the Bundled Data Protocol(BDP) and the Dual Rail Protocol (DRP) [3].

The BDPs, in different literature, are sometimes known as the Single Rail Protocols (SRP). These terms refer to the separate single req and ack wires that are bundled together with the data signals, see Figure 2.1.

The dot in Figure 2.1 denotes the active side of communication. The receiver does not have a dot because it is passive, i.e., only responds (ack) when communicated to (req). This

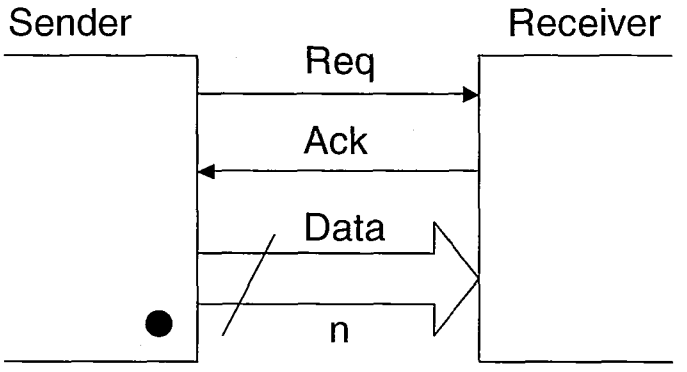



Figure 2.1: Bundled Data Channel

type of communication channel is called a push channel. The 'n' denotes the number of data bits.

In push channels, data is always valid before the request signal is put high (for 4-phase) or put into a different Boolean state (for 2-phase). The 4 and 2 phase protocols make up the BD and SR protocol.

### 2.2.1 4 Phase Bundled Data Protocol (4PBDP)

The 4-Phase Bundled Data Protocol (4PBDP) or in other texts the Return to Zero Bundled Data Protocol (RTZBDP), works using the following steps [3], see Figure 2.2:

1. The sender supplies data on the data line and sets the request line high. When the request line is put high, constant valid data must be provided on the data line.
2. The receiver then absorbs the valid data and sets the acknowledge signal high.
3. The sender then responds to the receiver by putting the request line low. When the request line is low, the data on the data line is no longer valid.

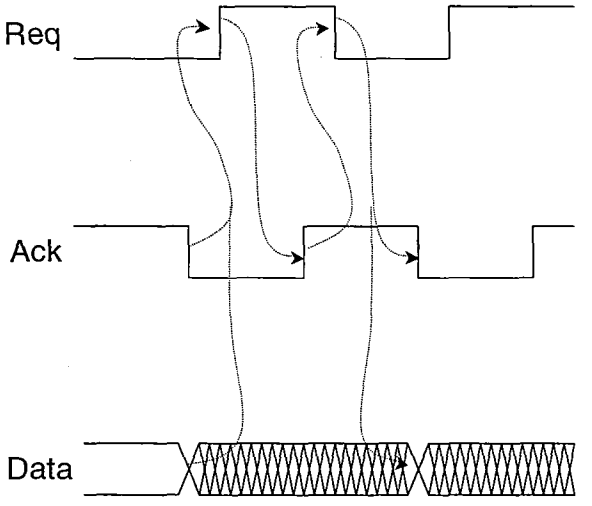



Figure 2.2: 4-Phase Bundled Data Protocol (4PBDP)

4. The receiver acknowledges the sender by putting the acknowledge line low. After the acknowledge line is low the sender can then start the process again.

The advantage of the 4PBDP is its familiarity to most designers. The disadvantage is that the protocol makes use of the Return To Zero (RTZ) behaviour, which wastes time and energy, i.e: Extra transitions to RTZ in each communication cycle will consume more power and time rather than the Non Return To Zero (NRTZ) behaviour.

The way a certain asynchronous system 'responds to events' is a complex issue. This means, that saying at the beginning of a design that protocol A is better than protocol B because A usually is quicker, may not be true or a realistic assumption.

Note: Depending on the convention of the designer, the circuit's handshaking may be initialized on a rising or falling edge.

### 2.2.2 2 Phase Bundled Data Protocol (2PBDP)

The 2-Phase Bundled Data Protocol (2PBDP) [3] or in other texts the Non Return To Zero Bundled Data Protocol (NRTZBDP) is different than the 4PBDP in that it uses the NRTZ

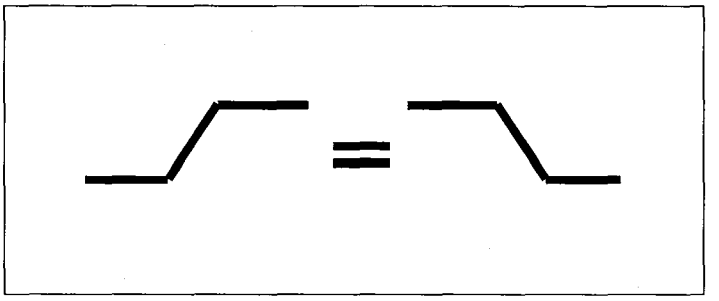


Figure 2.3: Transition Signaling Paradigm [13]

behaviour.

The 2PBDP also uses a different signaling paradigm called transition signaling or event signaling. Normally when circuits are enabled, an adjustment to the Boolean level is how the circuit interprets active or not active.

In the transition signaling paradigm, the circuit is enabled through a transition from '0 to 1' or '1 to 0', see Figure 2.3. The Boolean level in transition signaling is not important; the event or transition is the relevant measurement.

The 2PBDP works using the following steps, see Figure 2.4:

1. The receiver sends a transition (in the first case from '0 to 1') on the acknowledge line to signal that it is ready to receive data. The transition on the acknowledge line is also used to signal to the sender that the receiver is done with the previous cycle's data.

2. The sender then puts new valid data on the data line and sends a transition on the request line to indicate that valid data is present. Then the process starts anew.

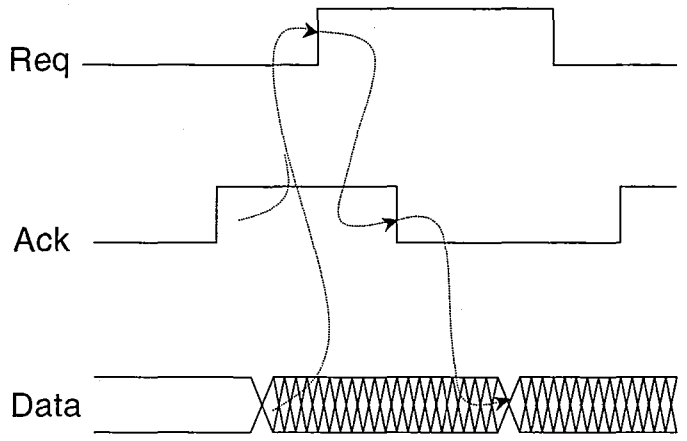



Figure 2.4: 2-Phase Bundled Data Protocol (2PBDP)

Note: When the communication starts again, the transitions will not be going from '0 to 1' as mentioned. The next cycle will always produce an opposite transition, e.g., from '1 to 0' if the communication starts from '0 to 1'. This opposite direction in the transition is irrelevant in the transition paradigm because a transition from '0 to 1' holds the same meaning as a transition from '1 to 0'.

## 2.3 Dual Rail Protocols (DRP)

In the BD [3] convention, there exists a need for delay matching. This delay matching is needed so that the order of the signals' events on the sender's end is preserved on the receiver's end. This is done so that valid data is always produced when needed.

There exists another protocol that is insensitive to delays; this protocol is the Dual Rail Protocol (DRP). It contains two main protocols called the 4-phase and 2-phase DRPs.

The '1-of-2' name refers to the protocols use of 2 wires to encode 1-bit of data information.

Table 2.1: 1-bit Channel - Encoding Chart

| For n=1 | d.t | d.f |
|---|---|---|
| Empty | 0 | 0 |
| Valid '0' | 0 | 0 |
| Valid '1' | 0 | 0 |
| Not Used | 0 | 0 |

### 2.3.1 4 Phase Dual Rail Protocol (4PDRP) Or 1-of-2 RTZ Protocol

The 4-Phase Dual Rail Protocol (4PDRP) [3], also known as the 1-of-2 RTZ protocol is very similar to the 4PBDP except that the request signal is encoded into the data signals. The protocol uses two wires per bit of information, d; one wire d.t. is used to transmit a Boolean logic '1' (true) and one wire d.f. is used to transmit a Boolean logic '0' (false). Once, again 'n' refers to the number of information bits, see Table 2.1 .

This method of communication is robust since the communication of the sender and receiver can be done reliably, regardless of delays. Hence, the protocol is 'delay insensitive'.

The 4PBDP works in the following way, see Figure 2.5:

1. The sender supplies a valid codeword, which is also a request put into the high level.
2. The receiver absorbs the valid codeword and sets acknowledge high.
3. The sender then responds to the acknowledge by supplying an empty codeword.
4. The receiver then sets acknowledge low. After the acknowledge line is low, the process can start over again.

### 2.3.2 2 Phase Dual Rail Protocol Or 1-of-2 NRTZ Protocol

The 2-Phase Dual Rail Protocol (2PDRP) otherwise known as the 1-of-2 NRTZ protocol encodes the data signals in the request signals. It also uses 2 wires {d.t, d,f} per bit. The

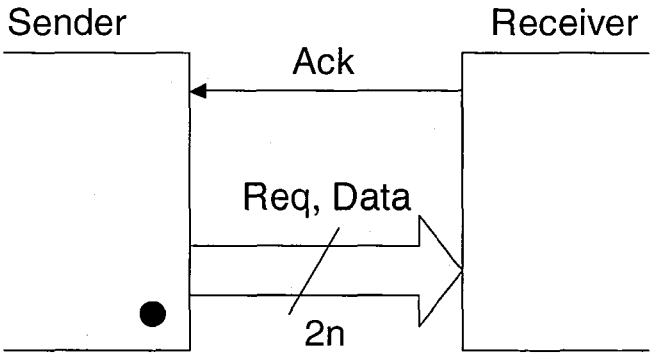



Figure 2.5: 4-Phase Dual Rail Protocol

information is encoded as transitions (events) as discussed in Section 2.2.2. Unlike the 4-phase system in Section 2.2.1, there is no empty value. Valid information is acknowledged, followed by another valid messaged and then the process continues. Figure 2.6 demonstrates a 2-bit wide channel using the 2PDRP.

## 2.4 Discussion On Protocol Choice

The previous sections have outlined the main protocols used to design asynchronous circuits. However, these 4 protocols can be combined to offer a plethora of protocol possibilities. In addition, the dual rail protocol can be further manipulated to offer extra encoding per bit of information, e.g., 4 wires to encode 1-bit of data, known as 1-of-4.

To give a general idea of the many possibilities of protocols used to design asynchronous circuits, the other channel types used must be briefly mentioned. Thus far, the 'push' channel was the only channel discussed.

In fact, there are 3 other channels used. The 'nonput' channel, is a channel that passes no data between channels. It is simply used as a coordinating link, i.e: req, ack and data

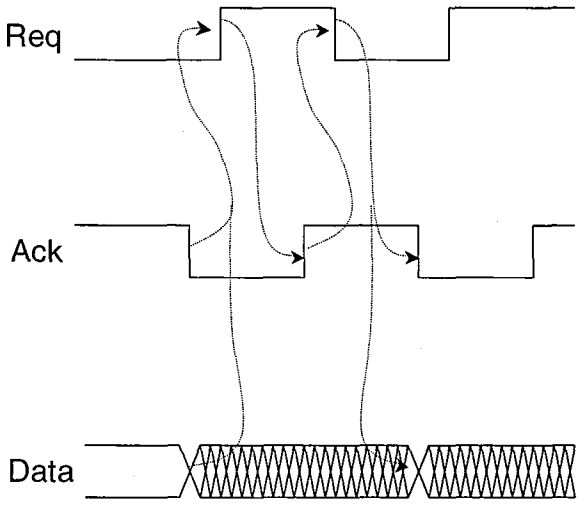



Figure 2.6: 2-Phase Dual Rail Protocol For A 2-Bit Wide Channel

link.

The 'pull' channel is the opposite of the push channel in that the receiver is the active party in the initiation of communication and the sender is the passive member. Figure 2.7 shows the different data validity schemes for the different types of channels.

The 'biput' channel is where the data is passed in both direction along with the acknowledge and request signals, see Figure 2.8.

The choice of protocol offers different features that may or may not suit the needs of the application. The protocol is an important choice and one that will influence the applications behaviour.

In [14], a comparison on asynchronous design styles were presented. They compared the 2PBDP, 2PDRP, 4PDRP, 1-of-4 RTZ and 1-of-4 NRTZ.

The comparison study yielded interesting results with respect to the potential for the 1-of-4 protocol. However, a general view of all protocols is outlined, which tends to show that the 2PBDP is a very reliable protocol with respect to area and power, see Table 2.2.

The 2PBDP offers a potential increase in circuit speed and decrease in area (wires/bit)

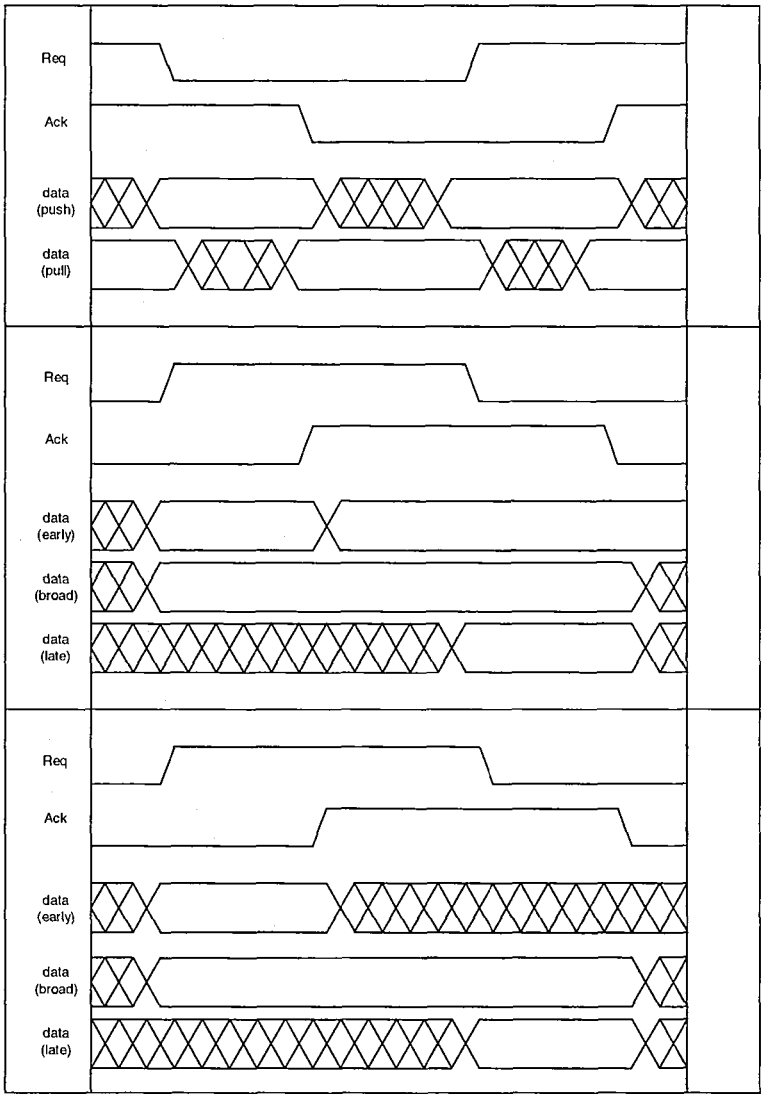



Figure 2.7: 2 Phase, 4 Phase (Push) And 4 Phase (Pull) Protocols

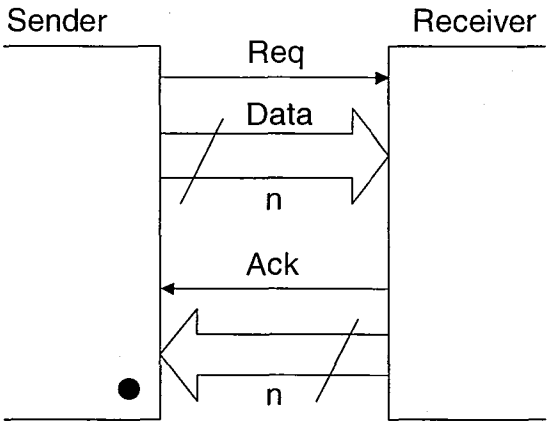



Figure 2.8: Binput Handshake Channel

Table 2.2: Comparison Of Protocols [14]

| | Area (wires/bit) | Energy (transitions/bit) |
|---|---|---|
| 2PBDP | 1 | 1/2 (avg.) |
| 2PDRP | 2 | 1 |
| 4PDRP | 2 | 1 |
| 1-of-4 (RTZ) | 2 | 1 |
| 1-of-4 (NRTZ) | 2 | 1/2 (avg.) |

Table 2.3: C-Element Truth Table

| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | Retain previous Z value |
| 1 | 0 | Retain previous Z value |
| 1 | 1 | 1 |

and energy (transitions/bit). This increase in speed and decrease in energy and area costs is directly correlated to the NRTZ behaviour that the 'transition signaling' paradigm offers.

The downfall to the 2PBDP is the overhead required to implement the delay matching circuitry and the extra level of complexity of transition signaling circuitry. However, the speed gained in using the 2PBDP would be a beneficial due to the computationally heavy nature of the BCJR/MAP algorithm. This will be investigated in later chapters.

## 2.5 Muller Basics

Muller [15] invented the Muller C-Element in 1959. It is the most often used primitive in asynchronous circuit implementations. The Muller pipeline, which is used in most asynchronous protocol implementations, is the backbone for handshaking circuitry [3].

### 2.5.1 The Muller C-Element

In asynchronous circuits, signals are required to be valid at all times. Indication and acknowledgment plays an important role in the design of these circuits. This means that every signal transition has a meaning and that hazards and races should be avoided [3].

The C-Element primitive is better suited to these types of constraints. The C-Element is a state holding element that acts as an 'OR' element for events. Unlike the 'OR' element, the C-Element can represent an acknowledgment when both inputs are '1' or '0', see Figure 2.9, Table 2.3 and Table 2.4.

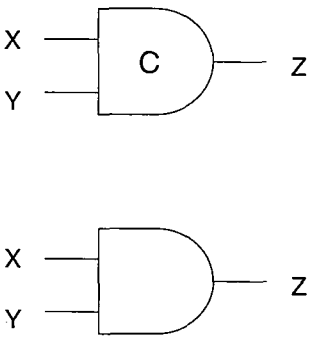



Figure 2.9: The C-Element (Denoted By A 'C') And The 'OR' Element Schematic

Table 2.4: 'OR' Truth Table

| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

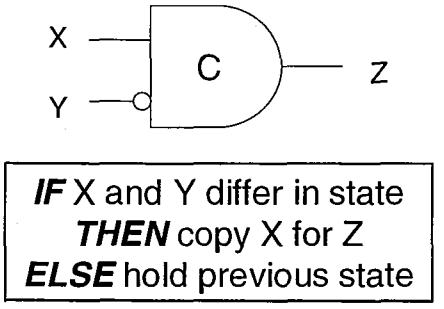



Figure 2.10: Behavior Of C-Element With Inverter [13]

### 2.5.2 The Muller Pipeline (MP)

In most asynchronous protocols, when analyzing the circuit implementation, the backbone that is always present is that of the MP. The MP is built from C-Elements and inverters. The MP is the structure that controls the handshaking in asynchronous circuits.

To understand the concept of the MP [3], the MP must be first put in its initial state. The initial state of all C-Elements are X='0' and Y='1', therefore Z='0'.

When an inverter (denoted by a circle) is placed on the input of the C-Element, it works in the following way, see Figure 2.10.

Network A, in Figure 2.11, is ready to send data to the MP. It sends a constant '1' on the req(in) line. This then starts a ripple effect through the MP. Note: The initial state of all C-Elements are X='0' and Y='1', therefore Z='0'.

Examining 'node 1', we see that after A has sent a '1', the value at 'node 1' will become a '1'. This changes the Req signal heading to the 2nd C-Element. This also sends an ack signal back to A.

'Node 2', then changes to a '1' bringing the value at 'node 3' to '1'. This has no effect on first C-Element. The first C-Element is now in a ready state. The '1' from the Req(in) line, propagates through all the C-Elements and puts each C-Element in a ready state (X='1', Y='1' and Z='1' (previous state)).

Once all C-Elements are in the ready state, the system can either send a constant '0' or '1' on the req(in) line. This '0' or '1' will propagate through the MP as did the first request of '1'. The MP acts as a flow through First-In First-Out (FIFO) circuit.

In an abstract way, the MP can be thought of as a wave. Where the wave carries a '1' through the MP or a '0'.

There are few other points that should be mentioned about the MP. Data doesn't necessarily have to flow from A to B. The symmetry of the MP makes the MP bi-directional. The complexity from A to B is the same from B to A. Therefore, communication can start from either 'A to B' or 'B to A'.

The MP also can be used for 2-phase or 4-phase protocols. The only difference between the two protocols is how the designer interprets the signals.

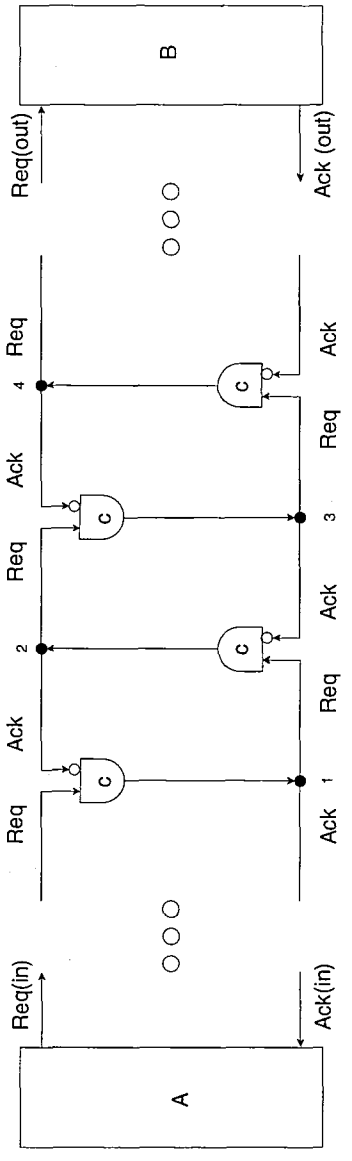



Figure 2.11: The Muller Pipeline

# Chapter 3

# *Convolutional Codes (CC)*

## 3.1 Introduction

P. Elias [9] proposed CC in 1954. CC is an error correction channel coding scheme that is the alternative to block codes.

As previously mentioned, in block coding, a data stream is broken up into separate blocks of information and then each block is encoded into an n-bit codeword. CC differs from block coding in that they encode the complete data stream into a single codeword [10].

There are several types of decoding algorithms used for CC. This chapter will focus on the Viterbi Algorithm and the BCJR/MAP algorithm.

## 3.2 Encoding

Before the decoding algorithms are discussed, the encoding process must be examined. In CC, encoding is accomplished through shift registers and adders ('XOR' elements), see Figure 3.1.

The rate or ratio ($R_c$) of source bits ($k$) to code bits ($n$) is known as the code rate.

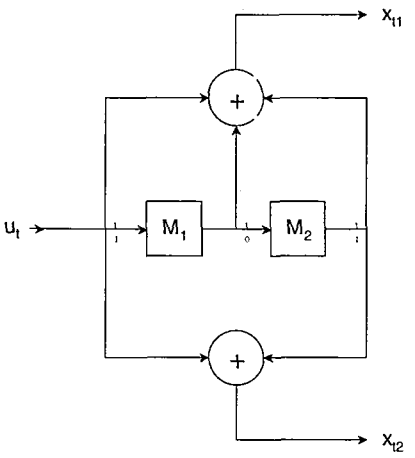



Figure 3.1: $(7,5)$ Convolutional Encoder With $R_c = \frac{1}{2}$ And $L = 3$ [9]

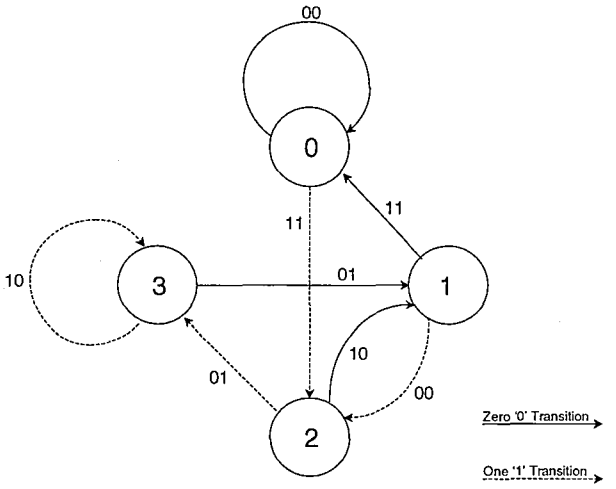



Figure 3.2: Finite State Machine (FSM) For $(7,5)$ Encoder [9]

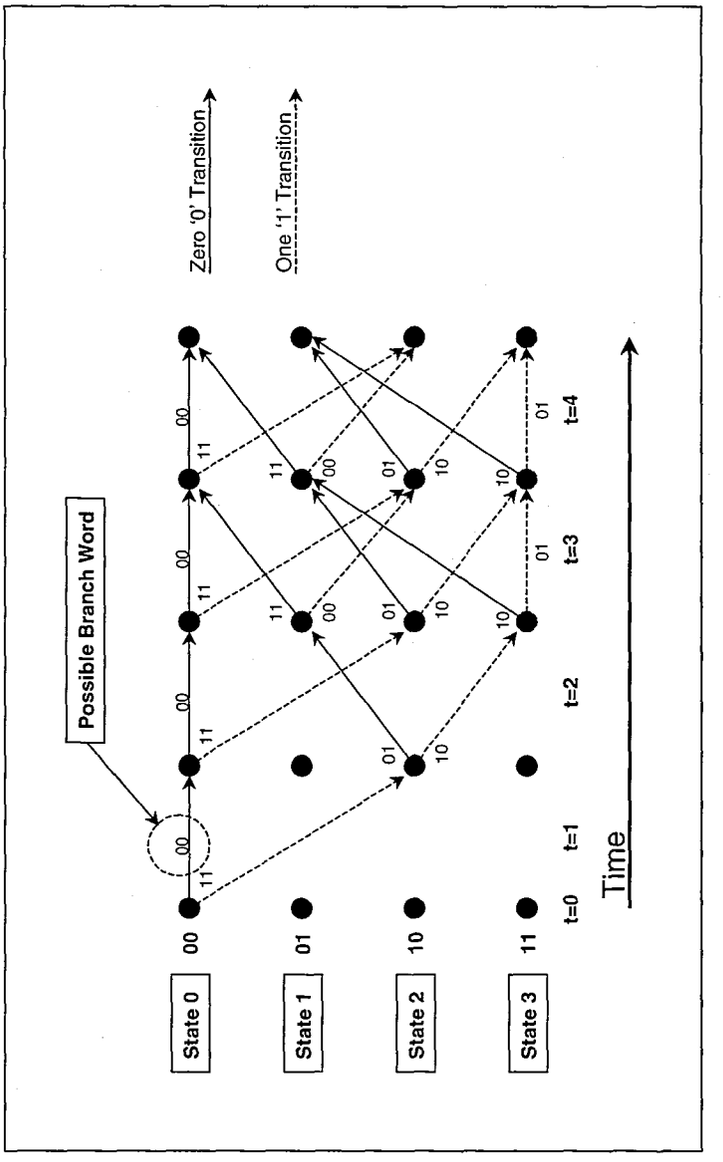



Figure 3.3: Trellis Diagram For The $(7,5)$ Encoder [9]

The constraint length ($L$) is defined as $M+1$, where $M$ is the number of Memory elements needed. The memory describes how many bits in the input sequence will affect the output sequence at a time. The generator sequence can be thought of how the actual wiring of the encoder is carried out, i.e., where a binary '1' mean there exists a connection and a binary '0' represents an absence of a connection. The standard for the sequence is usually represented by the octal number system, e.g., $(7,5)_8 = (111,101)_2$. Figure 3.1 shows a $(7,5)$ convolutional encoder with $R_c = \frac{1}{2}$ and $L = 3$.

The encoder, before the source bits of information enter it, is filled with zeros, i.e., the all-zero state. This brings the encoder to an initial state, this process is sometimes known as 'padding the systems with zeros', 'flushing' or 'trellis termination'. This 'flushing' makes the decoding process easier but causes rate loss. The process is satisfactory for long block lengths but for relatively short block lengths, the loss is unacceptable and causes degradation to the system's performance.

'Flushing' works in the following way; the information bits are entered, $k$ bits at a time. The corresponding code bits are then transmitted over the channel. This process continues and when all sets of $k$ bits are sent, the encoder is sent zeros to put the encoder in the all-zero state. Then the system is ready for the next transmission. The amount of zeros needed for padding is $k(L-1)$. For the $(7,5)$ encoder example, the systems needs $k(L-1) = 1(3-1) = 2$ 'zeros'.

For convolutional encoding, there are two main types of encoders. They are the Recursive Systematic Convolutional (RSC) see Figure 3.4 and Non-Recursive Convolutional (NRC) encoders see Figure 3.1. In a turbo encoder, typically we would find 2 NRC encoders. The NRC encoders are used due to their advantage in performance at low Signal To Noise Ratio (SNR) values in the communication channel. [16]. However, in terms of a standalone BCJR/MAP decoder, using one type of encoder over another does not matter in terms of performance.

RSC encoders are different than NRC encoders due to the feedback nature in their shift registers implementations, see Figure 3.4. NRC encoders are implemented as Finite Impulse Response (FIR) Filters, however, RSC are implemented as Infinite Impulse Response (IIR)

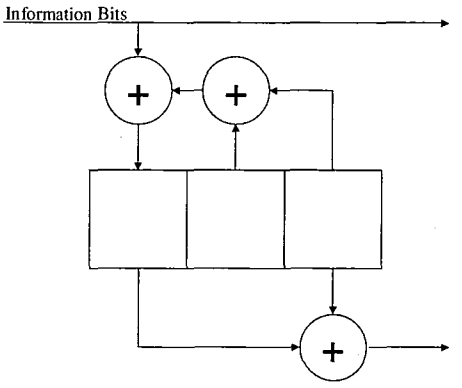



Figure 3.4: Recursive Systematic Convolutional (RSC) Encoder [17]

filters. For the sake of simplicity with this project, a NRC encoder was utilized. This has ramifications when the Log Likelihood Ratio (LLR) calculation is executed in the decoder, i.e., more storage and additions are needed with the RSC implemention because all three metrics (alpha,beta and gamma) are needed for the LLR calculation unlike the NRC implementation were only two metrics are needed (alpha and beta). Therefore, an NRC code implementation will more than likely have smaller area and lower power consumption than RSC without the expense of performance.

The convolutional encoder can be presented in various forms, e.g., functional diagram shown in Figure 3.1, finite state diagram shown in Figure 3.2 and the trellis diagram shown in Figure 3.3.

## 3.3 Decoding

Decoding is the reverse act of encoding. Decoding compared to encoding is more complex in terms of the amount of computation and memory requirements needed. There are two types of decoding frameworks, known as hard and soft decision decoding.

Hard-decision decoding is the process of using two quantized levels from the demodulator as the symbols used for comparison methods. In hard decision decoding, the quantized levels

are binary values of '0' and '1'. Soft-decision decoding uses more than two quantized levels and this yields better performance. Hard-decision decoding produces an extra 2 to 3 $dB$ performance loss over that of soft-decision decoding [11].

## 3.4 Channel Model

Before the decoding algorithms are demonstrated, the channel model used must be first discussed. A channel is the medium that "transfers or stores information" that is used in a communication cycle [9]. The physical medium can be wire, e.g., twisted pair communication, air (wireless), glass (fibre-optic), etc.

In this study, the modulator, channel and demodulator make up the channel model. There are three important channel models used. They are the Binary Symmetric Channel (BSC) and Binary Erasure Channel (BEC) which are forms of the Discrete Memoryless Channel (DMC), see Figure 3.5 and Figure 3.5. The channel model used in this thesis is the Additive White Gaussian Noise (AWGN) Channel which is a type of discrete-time channel characterized by a set of probability density functions, see Figure 3.6.

The input, $X = \{0, 1\}$, to a channel in a digital communication system is a sequence of binary numbers. The output, $Y = \{0, 1\}$, is also a sequence of binary numbers. Suppose the channel introduces "statically independent errors" in the transmitted binary sequence with the average probability $p$.
Therefore,

$$Pr\,(Y = 0 \mid X = 1) = Pr\,(Y = 1 \mid X = 0) = p \tag{3.1}$$

$$Pr\,(Y = 1 \mid X = 1) = Pr\,(Y = 0 \mid X = 0) = p - 1 \tag{3.2}$$

Thus, the BSC is obtained, see Figure 3.5.

Another DMC is the BEC, with inputs, $X = 0, 1$, and outputs, $Y = 0, e, 1$, where $e$ is an 'erasure', meaning the channel cannot decide if the output is a '1' or a '0', see Figure 3.5.

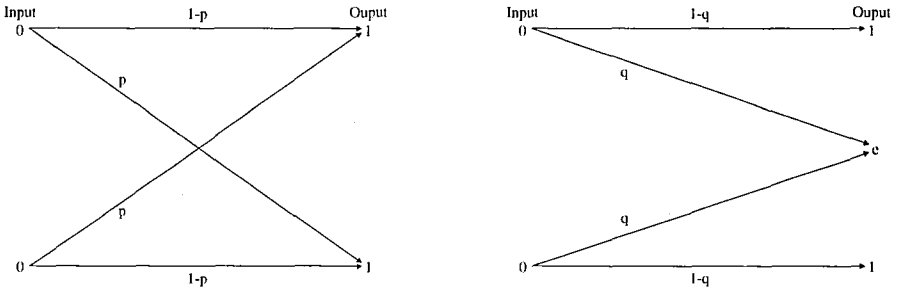



Figure 3.5: Binary Symmetric Channel (BSC) And Binary Erasure Channel (BEC) [9]

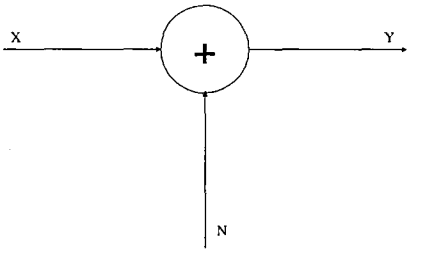



Figure 3.6: Additive White Gaussian Noise (AWGN) Channel [9]

As mentioned, the AWGN channel is the channel model used in this thesis. It is shown in Figure 3.6. This channel model includes the "output alphabet equal to the entire real line", $Y = \{-\infty, \infty\}$, and the "input alphabet has a finite number of symbols", $X = \{x_1, x_2, \ldots, x_{t-1}\}$. The AWGN channel is characterized by the following probability density function,

$$p(y \mid X = x_i) = \quad \text{where } i = 1, 2, \ldots, l-1 \tag{3.3}$$

The channel equation is as follows,

$$Y = X + N \tag{3.4}$$

where N is a zero mean Gaussian random variable with variance $\sigma^2$ and $X = x_i$ where $i = 1, 2, \ldots, l-1$. The probability distribution function of $Y$ with a given $X$ is Gaussian with mean $x_i$ and variance $\sigma^2$. Then,

$$p\left(y \mid X = x_i\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(y-x_i)^2/2\sigma^2} \tag{3.5}$$

Therefore, for a given input sequence, $x_{t,i}$, of symbols, $n$, where $t = 1, 2, \ldots, n$ and $x_{t,i} \in X$, the output sequence for a time, $t$, is given by the following equation,

$$y_t = x_{t,i} + n_t \tag{3.6}$$

The condition of the channel when memoryless can be expressed as,

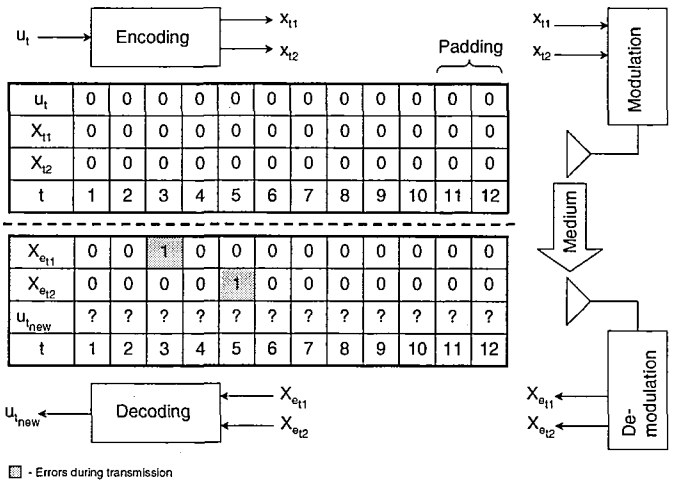
$$\begin{aligned} Pr\left(y_1, y_2, \ldots, y_n \mid X = x_{1,t}, \ldots, X\right) &= \Pi_{t=1}^{n} Pr(y_t \mid X = x_{t,i}) \\ &= \left(\frac{1}{2\pi\sigma^2}\right) \exp\left\{-\frac{1}{2\sigma^2}\sum_{t=1}^{n}(y_t - x_{t,i})^2\right\} \end{aligned} \tag{3.7}$$

The channel also, in our case, depends on the modulation used. This thesis uses the Binary Phase Shift Keying (BPSK) modulation technique. This means that any level value of '0' gets mapped to '-1' and any level value of '1' gets mapped to '1'.

## 3.5 Sample Values

Section 3.6 will use Figure 3.7 to obtain encoder output with corresponding channel errors, which is the input to the decoder after demodulation.

Note: Channel errors in Figure 3.7 are highlighted in red.




| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $u_t$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $X_{t1}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $X_{t2}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X_{e_{t1}}$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $X_{e_{t2}}$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $u_{t_{new}}$ | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

Figure 3.7: Sample Encoder Output Values With Corresponding Channel Error

## 3.6 Viterbi Algorithm

The Viterbi is a maximum-likelihood decoding algorithm [9]. Figure 3.8 shows how an example of the Viterbi Algorithm works. The bold line along the top is the final most likely decoded sequence, all '0' transitions, which is the correct decoded sequence from the sample. The lightly coloured lines are the lines during the algorithm in which we eliminate as being not a likely transition. The circled numbers signify the local winner. The final circled number at the top right signifies the global winner.

In its simplest form, a specific sequence is chosen as the most likely sequence if the likelihood of that sequence is larger than the likelihood of all other possible sequences.

The algorithm uses the following computational properties to find the most likely sequence:

- *Branch Word or Label*: Is the encoder output due to the transition of an encoder state to another, see Figure 3.8.
- *Branch Metric*: Is the distance between the sequence received and the possible branch

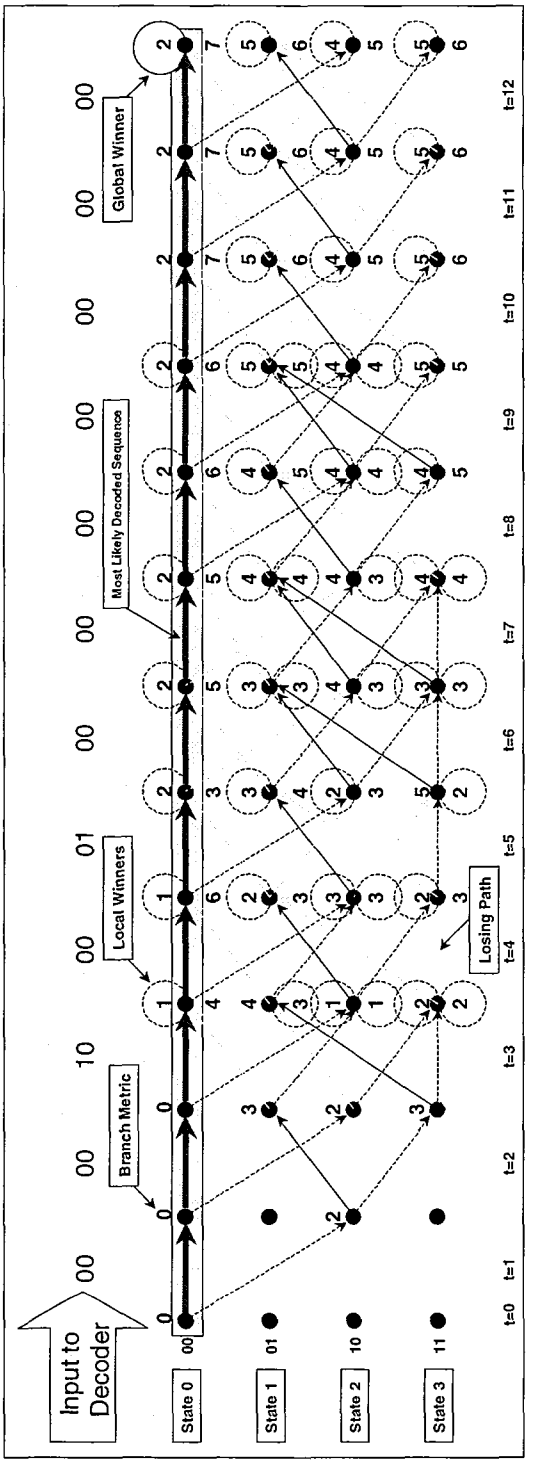



Figure 3.8: Viterbi Algorithm Decoding The Sample Encoder Output Values

word. For hard-decision decoding, it is usually Hamming distance and Euclidean distance for soft-decision decoding. Hamming Distance between two code words, $c_i$ and $c_j$, is the number of components that the two words differ from each other. Hamming distance is denoted by $d(c_i, c_j) = d_{ij}^H$. Euclidean distance, $d_{ij}^E$ can be expressed in terms of Hamming distance,

$$\left(d_{ij}^E\right)^2 = 4d_{ij}^H\mathcal{E} \tag{3.8}$$

where $\mathcal{E}$ is the signal energy [17]

- *Path Metric*: Is the summation of branch metrics.
- *Trellis*: Is the basis for a visual and structural representation of the calculations that need to be carried out for the algorithm. The trellis shows the states or nodes of the encoder at different moments in time.

Now that the structural metrics have been defined, the following steps make up the Viterbi algorithm:

1. The path metrics must be found for every path at each node. This is done by summing the correct branch metric to the related survived path metric.
2. The two paths, there will be two paths for the $K = 3$ case, that enter each node or state must be compared. The path with the smallest path metric is selected. This path is the survivor path metric. This step needs to be performed in parallel for $2K - 1$ states.
3. This procedure needs to be repeated until the end of the trellis is reached.

When the end of the trellis is reached, one will notice that all paths merge into a common state (if your system is padded or flushed). This means that only one path survives. If you trace-back that path, you can obtain the most likely original signal.

The depth of trellis in practice is five times the constrain length. There is no performance increase if you deepen the trellis. However, the depth is dependant on the rate of the encoder.

## 3.7 BCJR/MAP

Bahl et al. [24], first published the BCJR algorithm otherwise known as the 'symbol by symbol' Maximum A Priori (MAP) algorithm, in 1974 [24]. This algorithm was put aside for decoding convolutional codes due to its inherent complexity. However, the BCJR/MAP algorithm gained a second wind with the introduction of Turbo Codes. The main reason to use this algorithm is that it produces soft likelihood value of symbols on the output.

"The BCJR/MAP is an optimum decoding algorithm used to minimize the symbol error rate for the terminated convolutional codes. The BCJR/MAP estimates the A Posterior Probability (APP) of the states and the state transitions of a convolutional code and then the APPs of the symbols. [9]"

The BCJR/MAP algorithm, though complex, is very similar to the Viterbi algorithm. As mentioned above, the Viterbi, in its simplest form, chooses the most likely sequence as the final decoded sequence. This is done through several computational steps in a forward motion (left to right) through the trellis.

Like the Viterbi Algorithm, the BCJR/MAP algorithm search Forward and Backwards (FB) through the trellis for a 'received channel sequence' defined by $Y_1^L = \{Y_1, Y_2, ..., Y_L\}$ created by a 'channel input sequence' (encoder) $X_1^L = \{X_1, X_2, ..., X_L\}$. The BCJR/MAP is otherwise known as the FB algorithm. Depending on the channel model, $Y_t = (y_{t,1}, y_{t,2}, \ldots, y_{t,n})$ and $X_t = (x_{t,1}, x_{t,2}, \ldots, x_{t,n})$ for a $1/n$ code and $x_{t,i}$ and $y_{t,i}$ , $i = 1, 2, \ldots, n$. In this thesis, we use AWGN with BPSK, therefore $x_{t,i} \in \{-1, 1\}$ and $y_{t,i} \in \{-\infty, \infty\}$. This FB recursion will produce state probabilities defined as $\lambda_t(\cdot)$ or state transition probabilities defined as $\sigma_t(\cdot)$.

To produce the state probabilities or the state transitions 3 different values are needed:

- The forward recursion probability function, $\alpha(\cdot)$.
- The backward recursion probability function, $\beta(\cdot)$.
- The state transition matrix, $\Gamma(m', m)$.

The forward recursion is defined as the joint probability that the state of the system is $m$ at time $t$, $S_t = m$, and the received channel sequence from time 1 to $t$ is $Y_1^t$,

$$\alpha_t(m) = Pr\left\{S_t = m; Y_1^t\right\} \tag{3.9}$$

The backward recursion is defined as the probability that the received channel sequence from time $t+1$ to $L$ is $Y_{t+1}^L$, given that the state $m$ at time $t$ is $S_t = m$. Then,

$$\beta_t(m) = Pr\left\{Y_{t+1}^L \mid S_t = m\right\} \tag{3.10}$$

The state transition probability matrix is defined as the conditional probability that the state $m$ at time $t$ is $S_t = m$ and the received channel sequence at time $t$ is $Y_t$, while being at the state $m'$ at time $t-1$, $S_{t-1} = m'$. Then,

$$\Gamma\left(m', m\right) = Pr\left\{S_t = m; Y_t \mid S_{t-1} = m'\right\} \tag{3.11}$$

The next step involves estimating the APPs of the states and the state transitions for the received channel sequence. They are defined as follows,

$$Pr\left\{S_t = m \mid Y_1^L\right\} = \frac{Pr\left\{S_t = m; Y_1^L\right\}}{Pr\left\{Y_1^L\right\}} \tag{3.12}$$

and

$$Pr\left\{S_t = m; S_{t-1} = m' \mid Y_1^L\right\} = \frac{Pr\left\{S_t = m; S_{t-1} = m'; Y_1^L\right\}}{Pr\left\{Y_1^L\right\}} \tag{3.13}$$

The received sequence probability, $P\left(Y_1^L\right\}$, is constant during calculation, then the joint probabilities of the states and the received channel sequence denoted as $\lambda(m)$, and the joint probability of the state transitions and the received channel sequence denoted as $\lambda(m)$ can be calculated. They are defined as followed,

$$\lambda_t(m) = Pr\left\{S_t = m; Y_1^L\right\} \tag{3.14}$$

and

$$\sigma_t(m', m) = Pr\left\{S_{t-1} = m'; S_t = m; Y_1^L\right\} \tag{3.15}$$

Mathematically, $\lambda_t(m)$ and $\sigma_t(m', m)$ can be expressed as $\alpha(m)$, $\beta(m)$ and $\Gamma(m', m)$. By using Markov and DMC properties [9] for the derivations,

$$\begin{aligned} \lambda_t(m) &= Pr\left\{S_t = m; Y_1^L\right\} \cdot Pr\left\{Y_{t+1}^L \mid S_t = m; Y_1^t\right\} \\ \lambda_t(m) &= \alpha(m) \cdot Pr\left\{Y_{t+1}^L \mid S_t = m\right\} \\ \lambda_t(m) &= \alpha(m) \cdot \beta(m) \end{aligned} \tag{3.16}$$

and

$$\begin{aligned} \sigma_t(m', m) &= Pr\left\{S_{t-1} = m'; Y_1^{t-1}\right\} \cdot Pr\left\{S_t = m; Y_t \mid S_{t-1} = m'\right\} \cdot Pr\left\{Y_{t+1}^L \mid S_t = m\right\} \\ \sigma_t(m', m) &= \alpha_{t-1}(m') \cdot \Gamma_t(m', m) \cdot \beta_t(m) \end{aligned} \tag{3.17}$$

When $\lambda_t(m)$ or $\sigma_t(m, m')$ are found, the APPs of the data symbols can be calculated. Figure 3.9 shows the system diagram and the basic building blocks of the BCJR/MAP decoding system.

### 3.7.1 Forward Recursion

The forward recursion probability function, $\alpha(\cdot)$, is the sum of all probabilities of the state $m$ for the received channel sequence from time 1 to $t$. It is described as follows,

$$\alpha(m) = Pr\left\{S_t = m; Y_1^t\right\} = \sum_{m'}^{2^{K-1}-1} Pr\left\{S_{t-1} = m'; S_t = m; Y_1^t\right\} \tag{3.18}$$

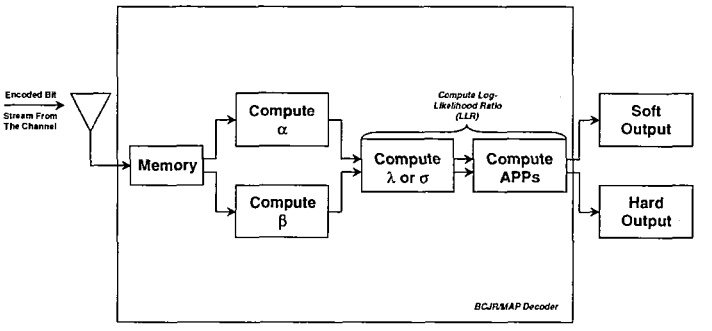



Figure 3.9: System Diagram Of The BCJR/MAP Decoding Algorithm

Rewriting equation 3.18 using the Markov chain and the DMC properties, the equation becomes,

$$\begin{aligned} \alpha(m) &= \sum_{m'}^{2^{K-1}-1} Pr\left\{S_{t-1}=m'; Y_1^{t-1}\right\} \cdot Pr\left\{S_t=m; Y_t \mid S_{t-1}=m'\right\} \\ \alpha(m) &= \sum_{m'}^{2^{K-1}-1} \alpha_{t-1}(m') \cdot \Gamma(m',m) \quad for\ t=1,2,\ldots,L \end{aligned} \tag{3.19}$$

Now using vector notation, we can take $\alpha_t$ as an $N$ tuple row vector with the $m^{th}$ entry being the probability of being in the $m^{th}$ state at time $t$, where $N$ is the number of states for memory, $M$, then, $N = 2^M$. Therefore, equation 3.19 is a row vector multiplied with a matrix. Note, the bold character refers to a matrix. Therefore,

$$\alpha_t = \alpha_{t-1} \cdot \mathbf{\Gamma}_t \tag{3.20}$$

$\alpha_t(m)$ requires normalization during each iteration to prevent underflow, since $Pr\left\{Y_1^L\right\}$ is not used in the calculation. Then, the normalized forward recursion, $\bar{\alpha}_t(m)$, is,

$$\bar{\alpha}_t(m) = \frac{\alpha_t(m)}{\sum_{m'=0}^{N-1} \alpha_t(m')} \tag{3.21}$$

The forward recursion expressed with normalization is as follows,

$$\alpha_t(m) = \sum_{m'=0}^{N-1} \bar{\alpha}_{t-1}(m') \cdot \Gamma_t(m', m) \tag{3.22}$$

$\alpha_0$, the initial conditions, can be determined by analyzing the starting state of the encoder. When the encoder starts at state $m$, then $\alpha_0 = (0, \ldots, 1, \ldots, 0)$ where the $m^th$ entry is 1.

### 3.7.2 Backward Recursion

Using the same procedure as in the forward recursion, the backward probability function can be obtain , $\beta_t(m)$. Therefore,

$$\beta_t(m) = Pr\left\{Y_{t+1}^L \mid S_t = m\right\} = \sum_{m'=0}^{N-1} Pr\left\{S_{t+1} = m'; Y_{t+1}^L \mid S_t = m\right\} \tag{3.23}$$

Rewriting equation 3.23 using the Markov chain and the DMC properties, the equation becomes,

$$\begin{aligned} \beta_t(m) &= \sum_{m'=0}^{N-1} Pr\left\{Y_{t+2}^L \mid S_{t+1} = m'\right\} \cdot Pr\left\{S_{t+1} = m'; S_t = m\right\} \\ \beta_t(m) &= \sum_{m'=0}^{N-1} \beta_{t+1}(m') \cdot \Gamma_{t+1}(m, m') \quad for\ t = 1, 2, \ldots, L-1 \end{aligned} \tag{3.24}$$

Using vector notation, equation 3.24 becomes,

$$\beta_{t-1} = \mathbf{\Gamma}_t \cdot \beta_t \tag{3.25}$$

Due to the previous subsections reasons, the backward recursion needs normalization, which is as follows,

$$\bar{\beta}_t(m) = \frac{\beta_t(m)}{\sum_{m'=0}^{N-1} \beta_t(m')} \tag{3.26}$$

The backward recursion in terms of the normalized values is then,

$$\beta_t(m) = \sum_{m'=0}^{N-1} \bar{\beta}_{t+1}(m') \cdot \Gamma_{t+1}(m, m') \tag{3.27}$$

If the final state is know and it is $m$ or the encoder is 'terminated', then $\beta_L$, the initial conditions, are $(0, \ldots, 1, \ldots, 0)$ where 1 is the $m^{th}$ entry.

### 3.7.3 State Transition Matrix

An $NxN$ matrix, where $N$ is the number of states in the code, is known as the transition matrix, $\Gamma(m', m)$, and is defined as follows,

$$\Gamma(m', m) = Pr\{S_t = m; Y_t \mid S_{t-1} = m'\} \tag{3.28}$$

Applying Bayes' rule we can show the following,

$$\Gamma(m', m) = \sum_U Pr\{U_t \mid S_t = m; S_{t-1} = m'\} \cdot Pr\{S_t = m \mid S_{t-1} = m'\} \cdot Pr\{Y_t \mid X_t\} \tag{3.29}$$

To describe equation 3.29, it's individual pieces can be broken up and analyzed:

- $Pr\{U_t \mid S_t = m; S_{t-1} = m'\}$: This value is either 0 or 1. A $1/n$ rate convolutional code contains two transitions from each state for a binary input $U_t$, e.g., transition 1 and transition 0.

- $Pr\{S_t = m \mid S_{t-1} = m'\}$: These are the APPs of the symbols at time $t$. The encoder input symbol probabilities are the same as the state transition probabilities. With turbo coding, i.e., iterative coding, the soft symbol information given by another decoder will be used in place of this term for the later iterations. The first iteration uses 0.5 for all symbols, given that the decoder has no priori probabilities of the symbols. The BCJR/MAP decoder that has been designed will be a standalone decoder. Therefore, 0.5 should be used for all symbols, however, it has been concluded through experimentation that we can neglect these APPs without any loss to performance see Section 5.3.1.

- $Pr\{Y_t \mid X_t\}$: This value depends on the channel property. Since, the AWGN channel with BPSK will be used, for a rate $1/n$ code, the AWGN channel with a single side power spectrum density value of $N_o/2$ gives,

$$Pr\{Y_t \mid X_t\} = \left(\frac{1}{\pi N_o}\right)^{n/2} \exp\left(-\frac{1}{N_o}\sum_{i=1}^{n}(y_{t,i} - x_{t,i})^2\right) \tag{3.30}$$

  where $x_{t,i} \in \{-1, 1\}$ and recall that in BPSK, bit level 0 is mapped to $-1$ and 1 is mapped to 1.

### 3.7.4 APPs Of The Symbols

Once $\Gamma_t(m', m)$, $\alpha_t(m)$, $\beta_t(m)$ are calculated it relatively simple to calculate $\lambda_t(m)$ defined as the joint probabilities of the states and the received channel sequence, and $\sigma_t(m', m)$, either one of the following equations are used to find the APPs of the symbols,

$$\lambda_t(m) = \alpha_t(m) \cdot \beta_t(m) \tag{3.31}$$

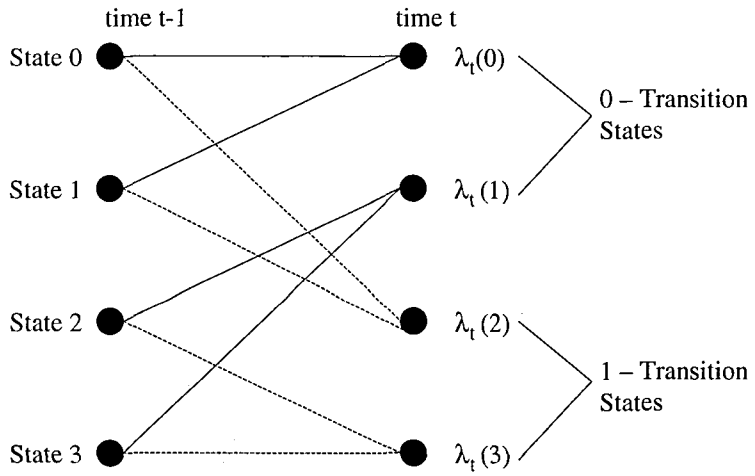



Figure 3.10: Transitions For NRC codes [9]

$$\sigma_t\left(m', m\right) = \alpha_{t-1}\left(m'\right) \cdot \Gamma_t\left(m'\right) \cdot \beta_t\left(m\right) \tag{3.32}$$

$\lambda_t\left(m\right)$ is typically used for NRC codes because each state can be either 'reached' by 1 or 0, but not both, unlike $\sigma_t\left(m', m\right)$. $\sigma_t\left(m', m\right)$ is typically used for RSC codes because each state can be 'reached' by both 1 or 0 transitions, see Figure 3.10.

The following equations show the remainder of the calculations needed,

$$Pr_{APP}\left[u_t = 1\right] = \frac{\sum_{m \in A} \lambda_t\left(m\right)}{\sum_{m \in All\ states} \lambda_t} \tag{3.33}$$

where A is all '1' transition states and

$$Pr_{APP}\left[u_t = 1\right] = \frac{\sum_{(m',m) \in B} \sigma_t\left(m', m\right)}{\sum_{(m',m) \in All\ branches} \sigma_t\left(m', m\right)} \tag{3.34}$$

where B is all '1' transition branches. The APPs of symbol '-1' at time $t$ for a given received channel sequence is

$$Pr_{APP}[u_t = -1] = 1 - Pr_{APP}[u_t = 1] \tag{3.35}$$

By choosing the APP with the largest value, the decoder can give hard outputs which are estimations of the input symbols, denoted $\hat{u}_t \in \{-1, 1\}$. Therefore, the decision rule is,

$$\hat{u}_t = \begin{cases} 1 & \text{if } Pr_{APP}[u_t = 1] \geq Pr_{APP}[u_t = -1] \\ -1 & \text{if } Pr_{APP}[u_t = 1] < Pr_{APP}[u_t = -1] \end{cases} \tag{3.36}$$

Equations 3.33 - 3.36 are based on the AWGN channel with BPSK.

# Chapter 4

# *Viterbi Decoder Using Asynchronous Techniques*

## 4.1 Introduction

The asynchronous paradigm can conceivably provide viable advantages to the problems caused by synchronous design. Applying asynchronous techniques, as seen in [8, 12] can offer the potential for lower power systems. The natural extension of the Amulet Group's (University of Manchester) paper is to apply asynchronous techniques to the BCJR/MAP algorithm. This chapter briefly introduces the ideas concerned with the asynchronous Viterbi decoder from [8, 12], as to better describe an asynchronous BCJR/MAP decoder.

## 4.2 Basic Building Block Concepts

The Amulet Group divided up the design into three main sub-systems shown in Figure 4.1:

1. Branch Metric Unit (BMU)
2. Path Metric Unit (PMU)

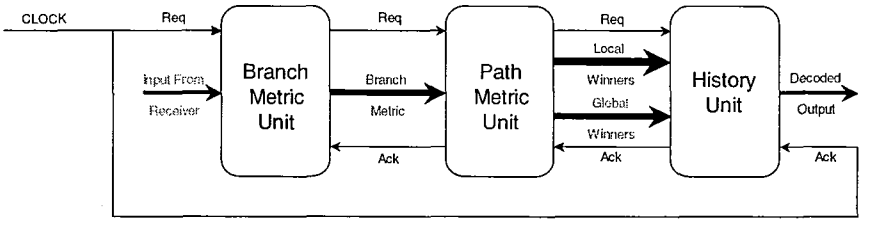



Figure 4.1: Viterbi Decoder Using Asynchronous Techniques - System Level [12, 8]

3. History Unit (HU)

The HU is sometimes called the Survival Metric Unit (SMU) in synchronous systems.

In Figure 4.1, the thick lines are the data signals and the thin lines are the handshaking control signals. The system's first request line and last acknowledge line is connected to a clock. This clock is solely to synchronize with an external synchronous systems' clock.

## 4.3 System Overview

The BMU takes the inputs from the receiver and calculates the distance between the received bits and the four ideal symbols (ideal branch words) that could have been transmitted, i.e., branch metric calculations. These weights are the inputs to the PMU.

Holding node weights, the PMU calculates the node plus branch weights, selecting the lower overall metric as the next metric for a node in a specific time period. The calculated node weights are then fed back into the PMU for the next calculation. The PMU also holds a bit of data for whether or not the winner was on the upper or lower branch entering the node.

For each time slot, the local winner information is inputted to the HU. The global winner, the state with the lowest node weight, is also passed on to the HU. This is done so that there is a known starting point when tracing back through the trellis history to find the most correct decoded sequence.

The HU, using the global and local node weights, reconstructs the trellis. It also traces back to find the path (most correct decoded sequence) from the global winner node to the local winner node in the oldest timeslot.

## 4.4 System Parameters

The system described used 4PBDP. The received bits to the decoder were soft coded; three bits were used to represent values. The system constrain length was 7. That means that there were 64 nodes/states and 128 paths or branches.

The system receives and decodes at a rate of $\frac{1}{2}$, however, other data rates are obtained by omitting some of the encoded symbols sent. Therefore, code rates of $\frac{2}{3}$,$\frac{3}{4}$,$\frac{5}{6}$ and $\frac{7}{8}$ can be obtained. As the code rate increases, less redundancy is included in the transmitted data signal, which results in increases in error rate from the decoder. The $\frac{1}{2}$ rate encoder has the most error-free output.

The system operates on a 90 MHz clock. This clock is used by all code rates, and the code rate defines two important ideas. First, the code rate defines the number of input bits to the number of transmitted encoded characters, which are the inputs to the decoder. Second, it specifies the ratio of the number of clocks containing encoded information and to the number of clock cycles. As an example, a $\frac{3}{4}$ rate means that three input bits result in four output transmitted characters and that there are three clocks out of four that contain encoded information. Therefore, in a $\frac{3}{4}$ rate code, every fourth clock cycle contains no

encoded information (Block-Valid Low). So a 90 $MHz$ clock rate using a $\frac{1}{2}$ rate encoder produces valid encoded information on every other clock cycle and yields a data rate of $\frac{1}{2} \times 90 = 45 \quad \frac{Msymbols}{sec}$.

## 4.5 Conclusions

The Viterbi decoder described in [3, 8] was compared in [12]. In many asynchronous designs, their synchronous counterpart cannot be directly translated into the asynchronous conceptual framework. This resulted in a design that was conceived from first principles and caused the designers to completely rethink how to implement the decoder.

The designers compared their design to the standard Viterbi decoder and the other decoders in the Power Reduction for System Technologies (PREST) project. The designers claim that the "power consumption was approximately an order of magnitude less than that of the other novel designs, and twice that again less than the reference design (when decoding an uncorrupted bit stream)".

The authors only mentioned that since the design is asynchronous, that low Electro Magnetic (EM) emissions would be desirable for a communication-related circuit.

The most promising aspect that was observed was that of the exploitation in the HU. Asynchronous designs most significant assets; doing nothing when nothing useful needs to be done was visibly shown in the power results. Since this is true when good channel conditions exist, this in no way impeded the system when the condition didn't exist. This aspect is the most difficult to achieve in a synchronous design.

The architecture used both high-speed serial operation (in the PMU) and lower speeds in the parallel operation (in the HU). These two architectures were used when seen to be most appropriate. The variety of architectures would have been difficult to achieve in a synchronous design due to the synchronization of different clocks.

# Chapter 5

# *Designing The BCJR/MAP Decoder*

## 5.1 Introduction

This chapter will introduce Balsa which is a Hardware Definition Language (HDL) used for asynchronous circuits and systems and a quick tool to assess the feasibility of an asynchronous design over a synchronous design.

The design constraints along with the cost saving design feature will be discussed.

## 5.2 Design Flow

### 5.2.1 High Level Languages And Tools

Asynchronous circuits are highly concurrent and the communication between modules is based on handshake channels. Therefore, a hardware language for the description of asynchronous circuits needs to incorporate these two characteristics.

The Communicating Sequential Processes (CSP) language [3] meets these constraints and contains the following characteristics:

1. Concurrent processes

2. Sequential and concurrent composition of statements within a process

3. Synchronous message passing over point to point channels (supported by primitives such as 'send', 'receive' and possibly 'probe')

CSP languages designed specifically for designing asynchronous circuits are:

1. Tangram, developed by Phillips and not available in the public domain

2. Balsa, developed by the University of Manchester and available in the public domain

### 5.2.2 CSP Vs. HDL

The CSP language [3] is not a standardized language as are other Hardware Definition Languages (HDL) such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog.

For instance, VHDL can be used to design asynchronous circuits, however, there are some disadvantages, such as:

1. VHDL lack primitives to implement the synchronous message passing needed by asynchronous circuits. It is possible to write low level code that implements the handshaking, however, it is undesirable to mix low level details into code whose purpose is to capture the high-level behaviour of the circuit.

2. VHDL lacks the statement level concurrency within a process.

VHDL does have advantages to its use, such as:

1. VHDL is well supported by existing Computer Aided Design (CAD) tool frameworks that provide simulators, pre-designed modules, mixed-mode simulation and tools for synthesis, layout and the back annotation of timing information.

2. The same simulator and test bench can be used throughout the entire design process form the first high-level specification to the final implementation in some target technology, i.e., standard cell layout.

3. It is possible to perform mixed mode simulation where some entities are modelled using behavioural specification and other are implemented using the components of the target technology.

4. Many real world systems include both synchronous and asynchronous subsystems and such hybrids systems can be modelled without any problems in VHDL.

Although, VHDL does have some excellent advantages, in terms of the study at hand. A CSP language offers immediate results. Balsa, a framework for synthesizing asynchronous hardware systems and a language for describing such systems will be the tool of choice for this study.

### 5.2.3 Balsa: Asynchronous Hardware Language And Synthesis Tool

Asynchronous circuits, at a hardware level, are complex systems that have not been addressed by industry standard Electronic Design and Automation (EDA) tools and companies. EDA tools are lacking and this is a sentiment expressed by Ian Sutherland, vice president, fellow and asynchronous expert at Sun Microsystems Laboratories [18].

VHDL and Verilog, industry standard hardware description languages, lack the needed "concurrency and platform for handshaking channels" [3] that are required by asynchronous systems. The problem is that the majority of tools are manufactured for the clocked design paradigm.

A recent free tool from the University of Manchester, one of the academic leaders in asynchronous design, is Balsa. Balsa is both a "framework for synthesis of asynchronous hardware systems and a language for describing such systems" [3]. Balsa uses the adopted approach of proprietary languages like Tangram, "syntax-directed compilation into communication handshaking components" [3]. This means that there is direct 'one-to-one' mapping between language and direct handshaking circuits produced.

Figure 5.1 details the design flow for a digital asynchronous circuit using Balsa version 3.4.1.

The Balsa circuit description is compiled using the *balsa-c* program to an intermediate

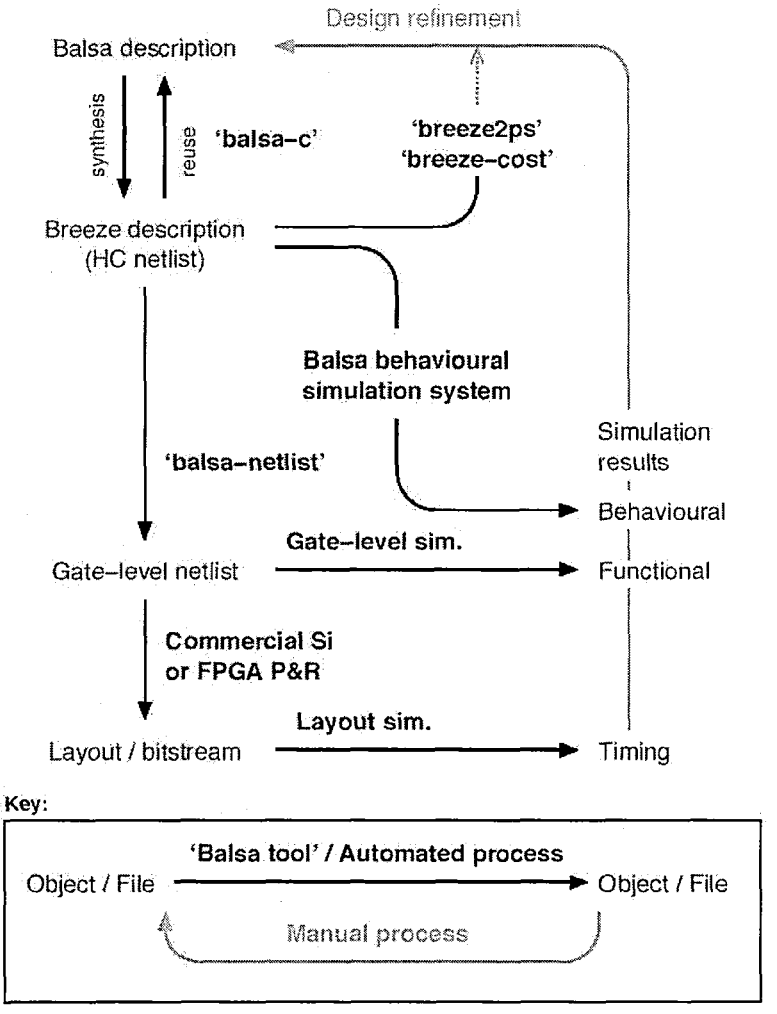



Figure 5.1: Balsa Asynchronous Design Flow [19]

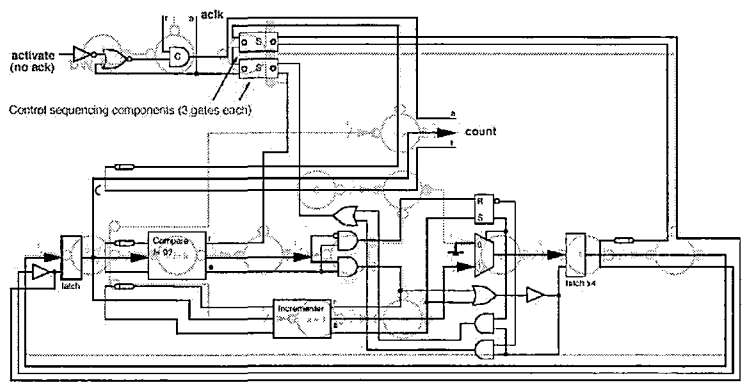



Figure 5.2: Gate-Level and Handshake Component Level Of A Modulo-10 Counter [19]

file type, called the breeze description. The breeze description is a network arrangement of a finite set of Handshake Components (HC) ( 45). The HC are connected across channels were the handshake signaling occurs. Figure 5.2 demonstrates a gate-level modulo-10 counter with the underlining handshake circuitry underneath drawn in light grey.

The majority of the tools in the Balsa package deal with the manipulation of the breeze files. The breeze files can be used by 'back-end' tools for different protocol implementations, however, the breeze files contain procedures and type definitions passed on from Balsa source files allowing breeze to be used as the package description format for Balsa. *breeze-sim* provides the behavioural simulation package for source level debugging, visualization of the channel activity at the handshake circuit level and it provides conventional waveform traces that can be viewed using GTKWave. GTKWave is a waveform viewer. The eventual target CAD package can be used to perform more accurate simulations and eventual physical layout.

After the gate-level netlist is obtained, commercial Silicon($Si$) or Field Programmable Gate Array (FPGA) packages can be used to continue the design flow. Since the Research Center for Integrated Microsystems (RCIM) at the University of Windsor does not have the rights for the supported technology mappings (from Balsa to Verilog Gate-Level Netlists) that the Amulet Advanced Processor Technologies (APT) Group from the University of

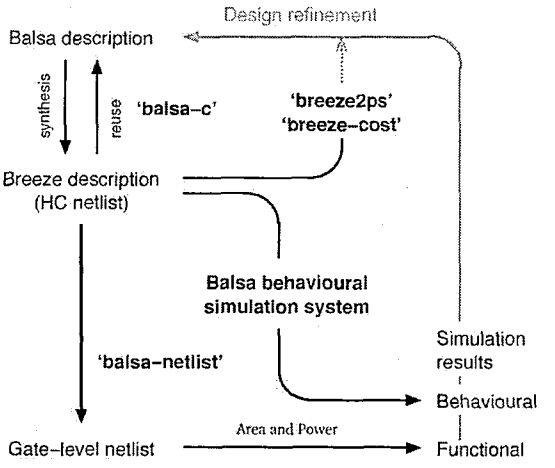



Figure 5.3: The Feasibility Asynchronous Design Flow (For The Area And Power Metrics)

Manchester provides; a mapping has to be created for our supported technology. The most recent fabrication technology available at the RCIM Lab is the Virtual Silicon Standard Cell Library for Taiwan Semiconductor Manufacturing Company (TSMC) 0.18-micron, single poly, six metal, salicide Complementary Metal Oxide Semiconductor (CMOS) process. See Appendix D for information regarding gate-level mappings and [19, 20].

### 5.2.4 Feasibility Design Flow

The feasibility design flow is the comparison of different metrics at an asynchronous gate-level (see Figure 5.3) to the synchronous gate-level. These metrics are dependant on the designer, but area and power are sufficient for this thesis. In other words, this study will consider everything up to the gate-level synthesis section of the design flow.

If the asynchronous metric(s) are less than that of the synchronous metric(s), then the design is feasible. These metrics are area and power dissipation. If the metrics (area and power) are equal, then there is no advantage (area and power wise) of using asynchronous over synchronous. However, one could hypothesize that noise spectral values may be more uniform compared to a synchronous implementation. This means that depending on the application and design constraints; the metrics should be chosen wisely.

Feasibility may mean to continue with the asynchronous design or that an asynchronous design may offer benefits greater than that of an synchronous design. This rule can be expanded for other metrics at different levels of abstraction, i.e., layout level will provide the most accurate results, however, it takes longer to reach that level of design.

## 5.3 The Asynchronous BCJR/MAP Decoder

The asynchronous BCJR/MAP consists of 4 main subsystems, they are:

- The Gamma Block, i.e., The Transition Matrix Block
- The Alpha Block, i.e., The Forward Recursion Block
- The Beta Block, i.e., The Backward Recursion Block
- The LLR Block, i.e., The APP Calculator and Decision Block

These subsystems naming conventions are very similar to the naming conventions of the Balsa procedures of the design, see Appendix C for the complete source files for the BCJR/MAP Balsa design.

### 5.3.1 System Constraints

There are some design constraints to be noted to provide an accurate description of the design constructed, Table 5.1 lists these constraints.

The channel model used was an AWGN channel. It is a benchmark for all communication channels. The channel was used because decoders that perform well in AWGN channels perform well in all channels. Rescaling equation 3.29,

$$\Gamma\left(m', m\right) = \sum_{U} Pr\left\{U_t \mid S_t = m; S_{t-1} = m'\right\} \cdot Pr\left\{S_t = m \mid S_{t-1} = m'\right\} \cdot Pr\left\{Y_t \mid X_t\right\}$$

The input channel variance can be neglected without any performance loss, see Section 5.3.3.

Table 5.1: Asynchronous BCJR/MAP System Constraints

| Constraint | Value |
|---|---|
| Encoder Type | NRC Encoder |
| Decoder Type | Max-Log MAP |
| Channel Model | AWGN |
| Modulation Used | BPSK |
| Code Rate | 1/2 |
| Constraint Length | 3-bit |
| Block Length | 20 bits |
| Decoder Input | 2 6-bit inputs (SXX.XXX) |
| Decoder Output | Variable [Hard(1-bit) or Soft(6-bit)] |
| Input APP | Neglected |
| Input Channel Variance | Neglected |

In Table 5.1 was that the input to the decoder did not use APPs. This was done for a several reasons. It would decrease the complexity of the decoder and that would mean no extra additions (in the logarithmic domain). Since a standalone decoder was being used, another decoding source would not be feeding this thesis' decoder with APPs. Therefore, in actuality, a standalone decoder would probably not receive APPs. If this decoder were to be used in a turbo system, this would be a constraint that would have to be altered because a turbo system requires iterative decoding between two decoders.

The block length input to the system is 20 bits long. Compared to most systems, this is a small number. However, the number of bits was chosen to simplify the design. Since, the feasibility of the design is of more importance, 20 bits was a sufficient block length.

The decoder outputs were hard outputs. This was done to check our system quickly with the MatLab simulation architecture, see the description of the simulation architecture in Section 6.2. Matrix Laboratory Software (MatLab) is "a high-level technical computing language and interactive development environment" [26].

### 5.3.2 The Log-MAP Algorithm And Max-Log-MAP Algorithm

Chapter 5 shows that the BCJR/MAP algorithm is computational heavy. In terms of VLSI implementation, the amount of multiplications and exponentials needed to carry out the computations of the algorithm are quite large. By using the logarithmic domain; multiplications and exponentials can be replaced by small Look Up Tables (LUTs), 'Maximum' operators and additions. These three replacements are more cost effective than large and bulky multiplications and exponentials.

The Log-MAP and Max-Log-MAP algorithms are basically the BCJR/MAP described in Section 3.7, except that an approximation called the Jacobian Logarithm [21, 23] (equation 5.1) is used to remove the implementation complexities of the BCJR/MAP. The Jacobian Logarithm is as follows,

$$\begin{aligned} MAX^* &= \ln\left(e^x + e^y\right) \\ &= MAX\left(x, y\right) + \ln\left(1 + e^{-|x-y|}\right) \end{aligned} \tag{5.1}$$

Essentially, the $MAX^*$ is a $MAX$ operator with a correction factor. This correction factor can be simply stored in a LUT with negligible effect on performance [22, 23].

The basic difference between the Max-Log-MAP and the Log-MAP is that the Max-Log-MAP disregards the correction factor in its computation. This leads to smaller area and a faster decoder, however, at the expense of performance [25]. This study uses the Max-Log-MAP algorithm due to its simplicity and better area and speed results than the Log-MAP.

### 5.3.3 Gamma Architecture

The gamma architecture that was implemented is shown in Figure 5.4. The data types used in the gamma system are shown in Figure 5.5. The sub-system architecture for the gamma sub blocks are shown in Figure 5.6 and Figure 5.7. For full coding details see Appendix C.

Recall equation 3.29, this equation was broken up into three parts.

1. $Pr\left\{U_t \mid S_t = m; S_{t-1} = m'\right\}$

2. $Pr\{S_t = m \mid S_{t-1} = m'\}$

3. $Pr\{Y_t \mid X_t\}$

'Part 1' determines the order of the values in the transition matrix. 'Part 2' was neglected, for various reasons discussed, see Section 3.7.3 and Section 5.3.1. 'Part 3' depends on the input to the system and the channel model, as discussed in Section 5.3.1, the channel variance can be neglected because it is not varying and does not depend on the input.

Therefore, 'part 3' can be written as follows,

$$Pr\{Y_t \mid X_t\} = \left(\frac{1}{\pi N_o}\right)^{n/2} \exp\left(-\sum_{i=1}^{n}(y_{t,i} - x_{t,i})^2\right) \tag{5.2}$$

In the logarithmic domain, the equation becomes,

$$\ln\left(Pr\{Y_t \mid X_t\}\right) = \ln\left(\left(\frac{1}{\pi N_o}\right)^{n/2}\right) + \ln\left(\exp\left(-\sum_{i=1}^{n}(y_{t,i} - x_{t,i})^2\right)\right) \tag{5.3}$$

The first term in the previous equation can be neglected because the Add-Compare Select property of the Alpha and Beta block will deem this value ineffective. For example,

$$\text{if } \; a = \text{constant} + 2 \;\; \text{and} \;\; b = \text{constant} + 1, \;\; \text{then } max(a, b) = a$$

The constant is irrelevant to the $max$ operator. Therefore, the equation becomes,

$$\ln\left(Pr\left(Y_t \mid X_t\right)\right) = -\sum_{i=1}^{n}(y_{t,i} - x_{t,i})^2 \tag{5.4}$$

This was also proven in MatLab simulations. A previously designed BCJR/MAP decoder, see Appendix B for all MatLab code, had its constant removed to see how the system would effect performance in terms of Bit Error Rate (BER) after decoding vs Signal To Noise

Ratio (SNR) of the communication channel. The units of BER are (# of bits with errors after being decoded \ # of bits transmitted) and the units of SNR are (dB).

Figure 5.8 shows that even by removing the constant it does not effect performance. The value labelled 'no sigma' is a BCJR/MAP decoder that has had it's constant removed. It achieves the same performance as the 'unquantized' system. Figure 5.8 also shows a small variation of the 'no sigma' values compared to the 'unquantized' values, i.e., not exactly the same. This is discrepancy is negligible due to the fact the input values are random generated values. The exact simulation may produce very similar results with new variations in the values.

### 5.3.4 Alpha Architecture

The alpha architecture that was implemented is shown in Figure 5.9. The data types used in the alpha system are shown in Figure 5.10. The sub-system architecture for the alpha sub blocks are shown in Figure 5.11 and Figure 5.12. For full coding details see Appendix C.

Regarding Figure 5.9, there is local logic that controls the sequence of iterative events, i.e., logic that initially feeds in locally stored variables (top left of schematic) and then feds the input the iterative output of the system (counter). $X = x + 1$, where initially x=0 and increments to 19.

Recall, equation 3.22, excluding the normalization correction. The normalization correction will be discussed in Section 5.3.7.

$$\alpha_t(m) = \sum_{m'}^{N-1} \alpha_{t-1}(m') \cdot \Gamma_t(m', m)$$

In the logarithmic domain the equation becomes,

$$\ln \alpha_t(m) = \ln\left(\sum_{m'}^{N-1} \alpha_{t-1}(m') \cdot \Gamma_t(m', m)\right)$$

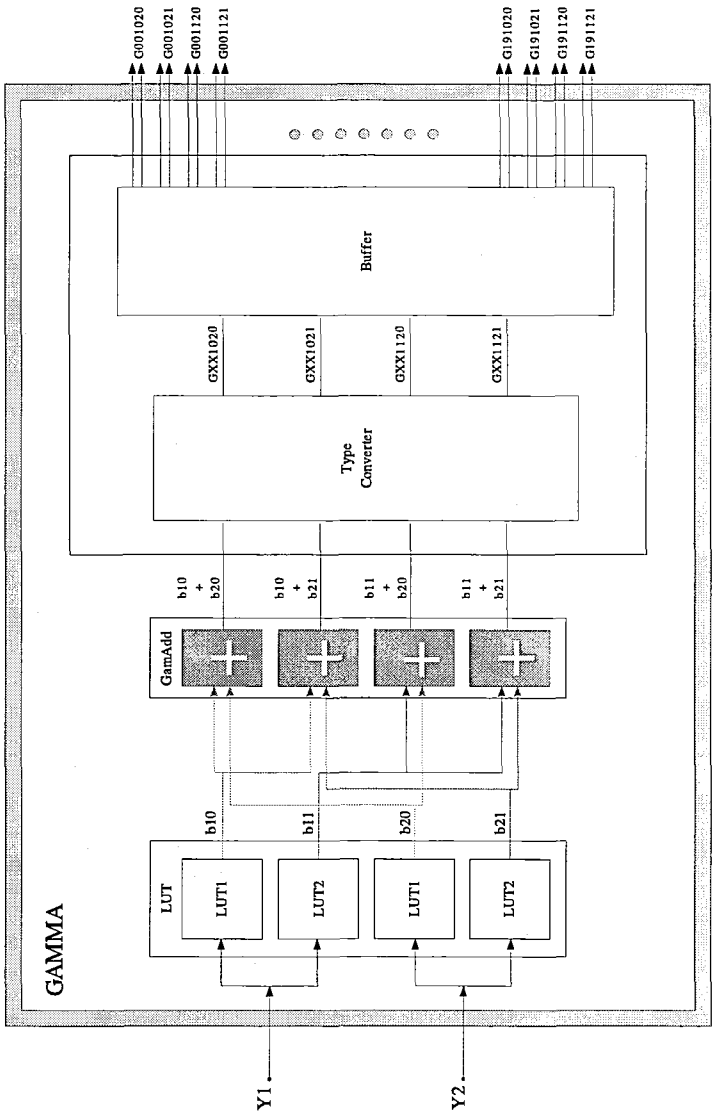



Figure 5.4: Gamma Architecture

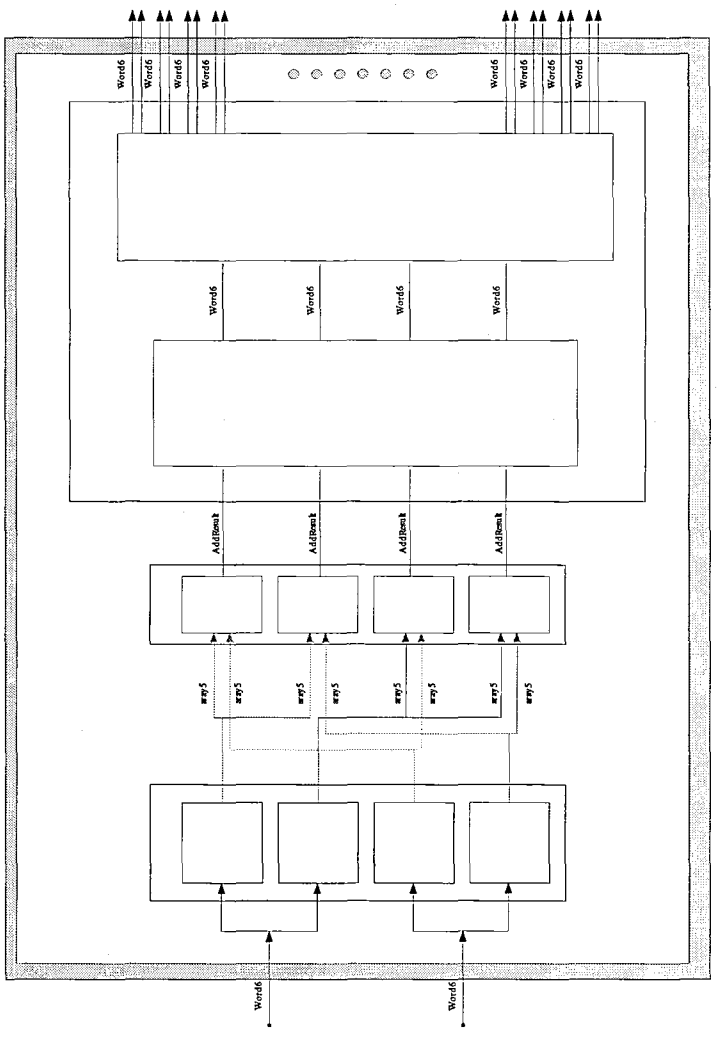



Figure 5.5: Gamma Architecture - Data Type Structures

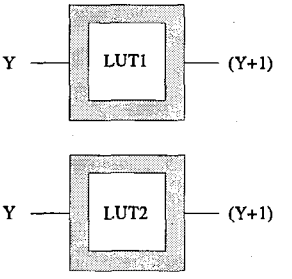



Figure 5.6: Gamma Sub-System Architecture - LUT

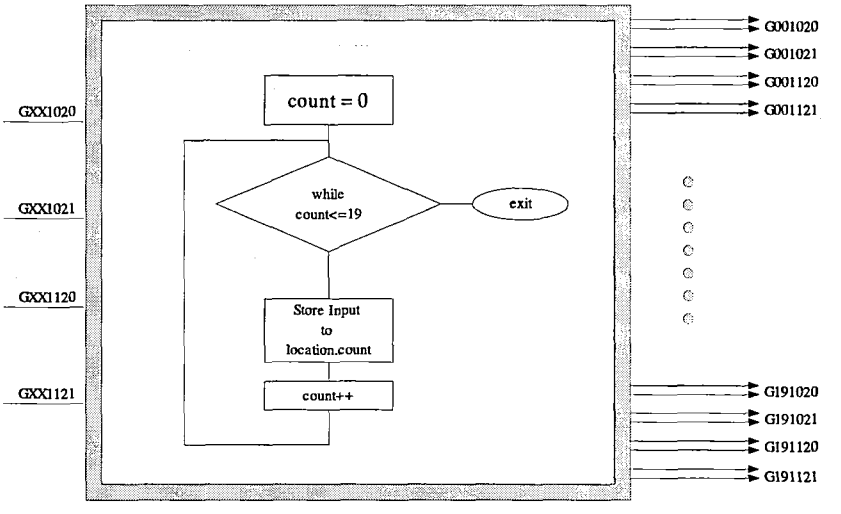



Figure 5.7: Gamma Sub-System Architecture - Buffer

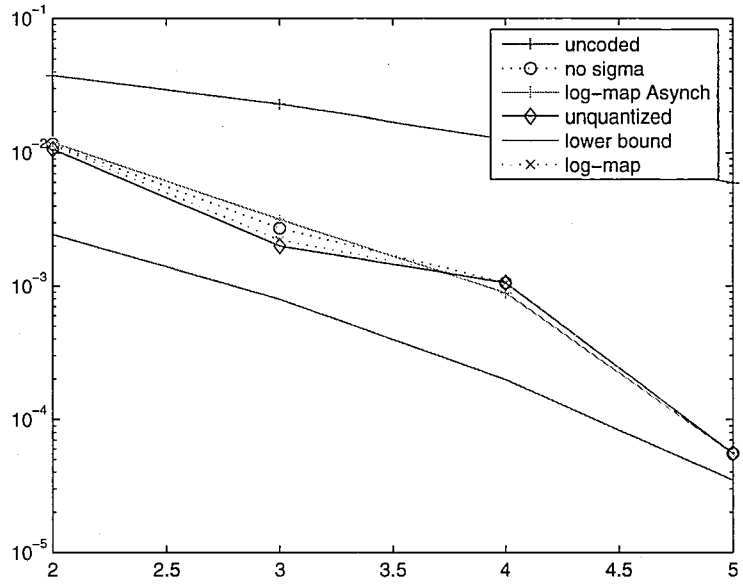



Figure 5.8: 1000 Blocks Transmitted Per dB Level - BER Vs. SNR

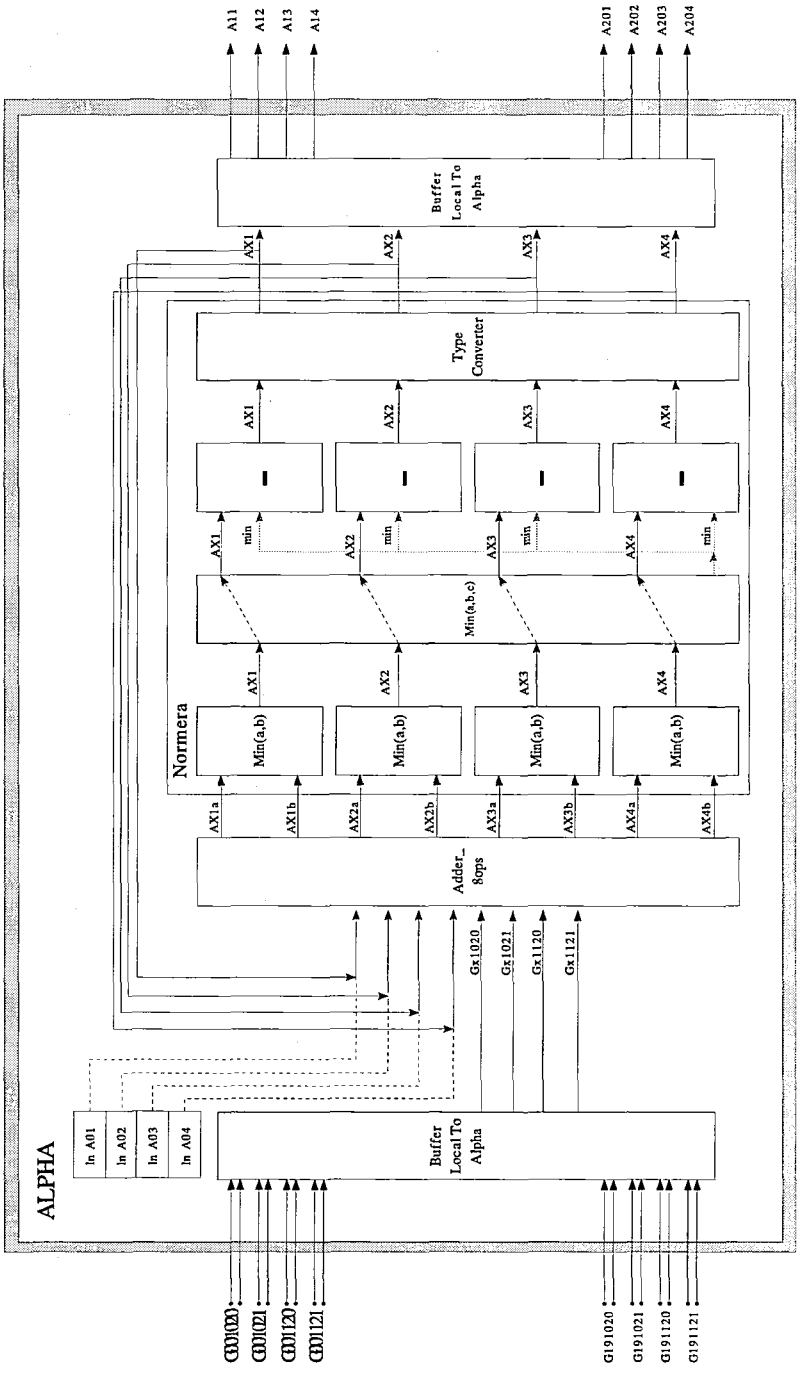



Figure 5.9: Alpha Architecture

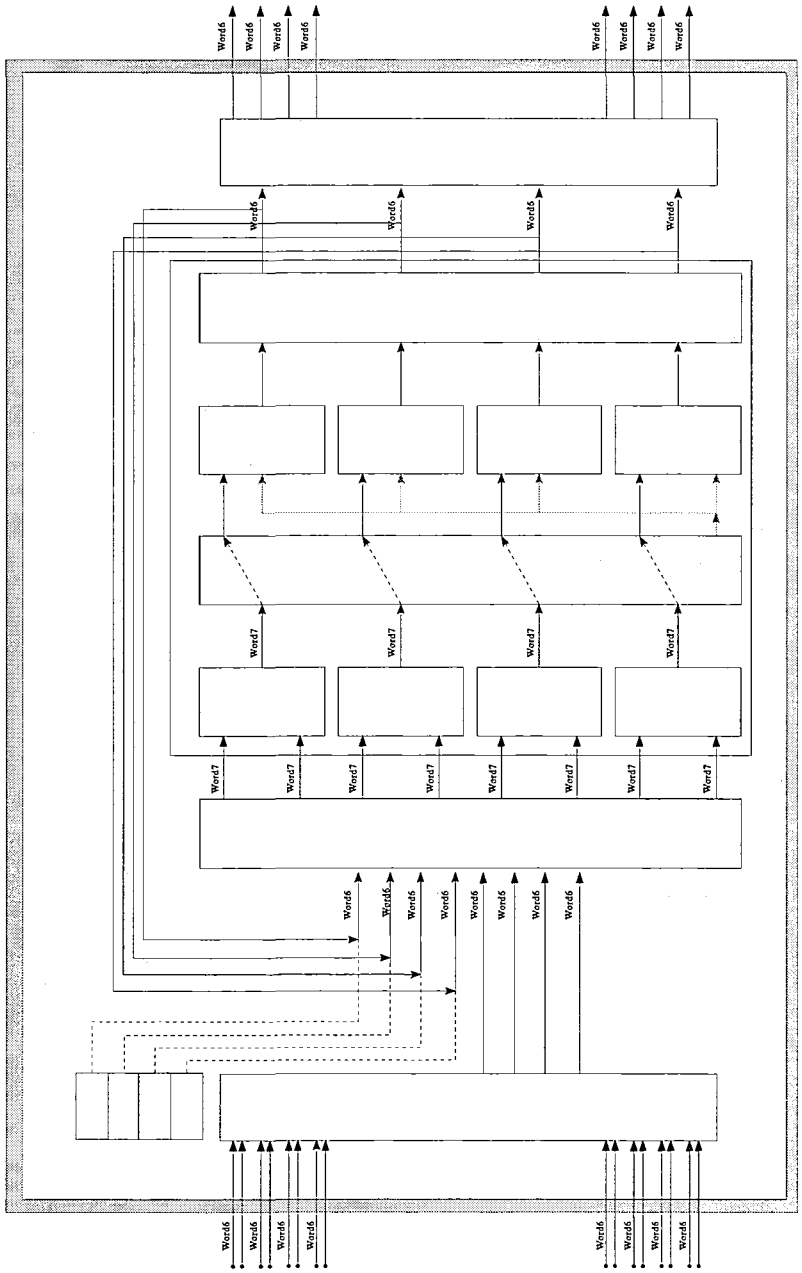



Figure 5.10: Alpha Architecture - Data Type Structures

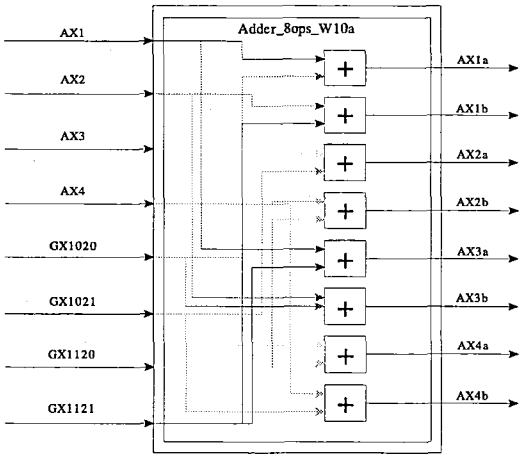



Figure 5.11: Alpha Sub-System Architecture - Adder

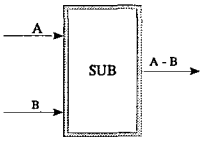



Figure 5.12: Alpha Sub-System Architecture - Subtractor

$$\begin{aligned}
\ln \alpha_t(m) &= \sum_{m'}^{N-1} \ln\left(\alpha_{t-1}(m')\right) \cdot \ln\left(\Gamma_t(m', m)\right) \\
\ln \alpha_t(m) &= \sum_{m'}^{N-1} \ln\left(\alpha_{t-1}(m')\right) + \ln\left(\Gamma_t(m', m)\right) \\
\ln \alpha_t(m) &= \ln\left(\sum_{m'}^{N-1} e^{\ln(\alpha_{t-1}(m'))+\ln(\Gamma_t(m',m))}\right) \qquad (5.5)
\end{aligned}$$

Let,

$$\begin{aligned}
\delta_1 &= \ln \alpha_{t-1}(m') \\
\delta_2 &= \ln \Gamma_t(m', m) \qquad (5.6)
\end{aligned}$$

and by using equation 5.1, the previous equation 5.5 becomes,

$$\ln \alpha_t(m) = MAX^*_{m'}(\delta_1, \delta_2) \qquad (5.7)$$

Rewriting, this equation becomes,

$$\ln \alpha_t(m) = MAX_{m'}(\delta_1, \delta_2) + \ln\left(1 + e^{-|\delta_1 - \delta_2|}\right) \qquad (5.8)$$

Since the Max-Log-MAP algorithm will be used for this design, the equation can be rewritten minus the correction factor,

$$\ln \alpha_t(m) = MAX_{m'}(\delta_1, \delta_2) \qquad (5.9)$$

### 5.3.5 Beta Architecture

The beta architecture that was implemented is shown in Figure 5.13. The data types used in the beta system are shown in Figure 5.14. The sub-system architecture for the beta sub blocks are the same as the alpha sub blocks, see Figure 5.11 and Figure 5.12. For full coding details see Appendix C.

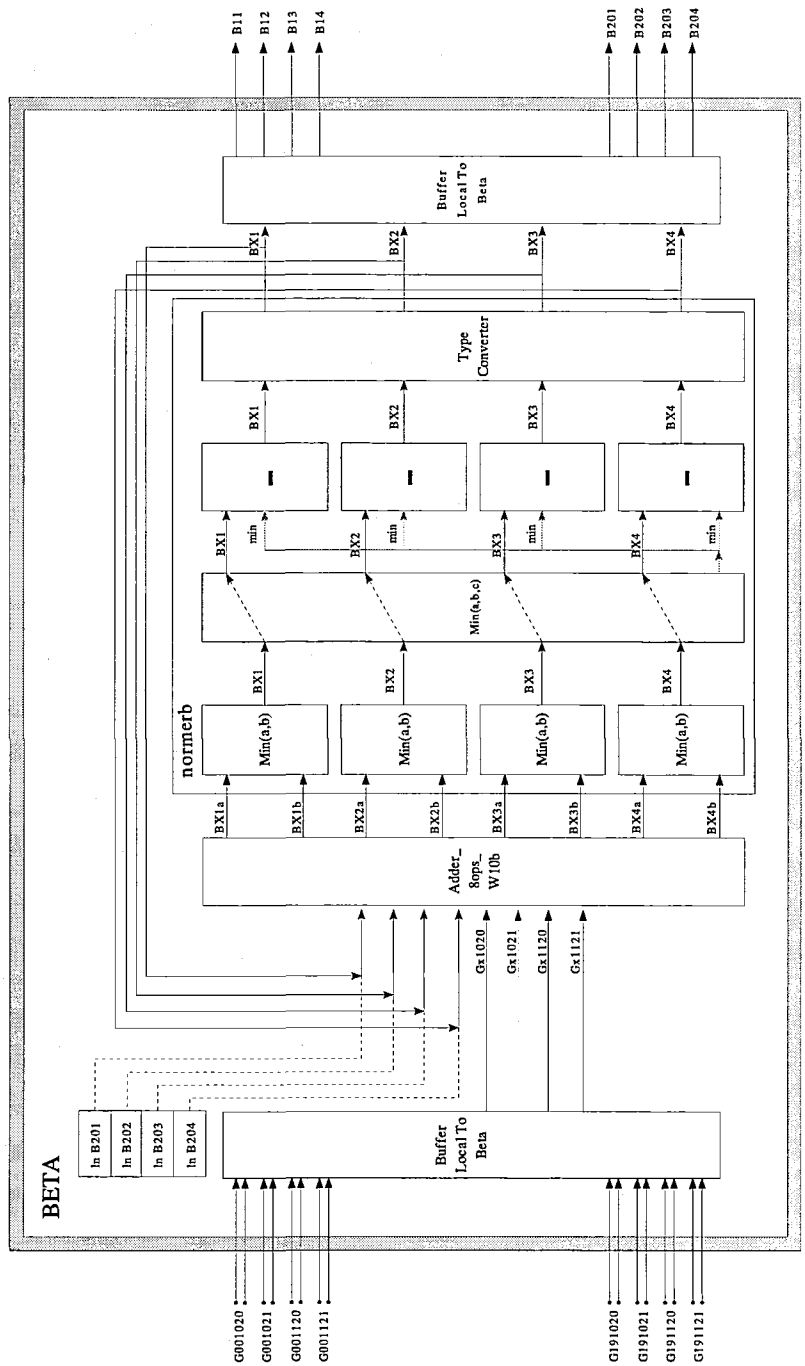



Figure 5.13: Beta Architecture

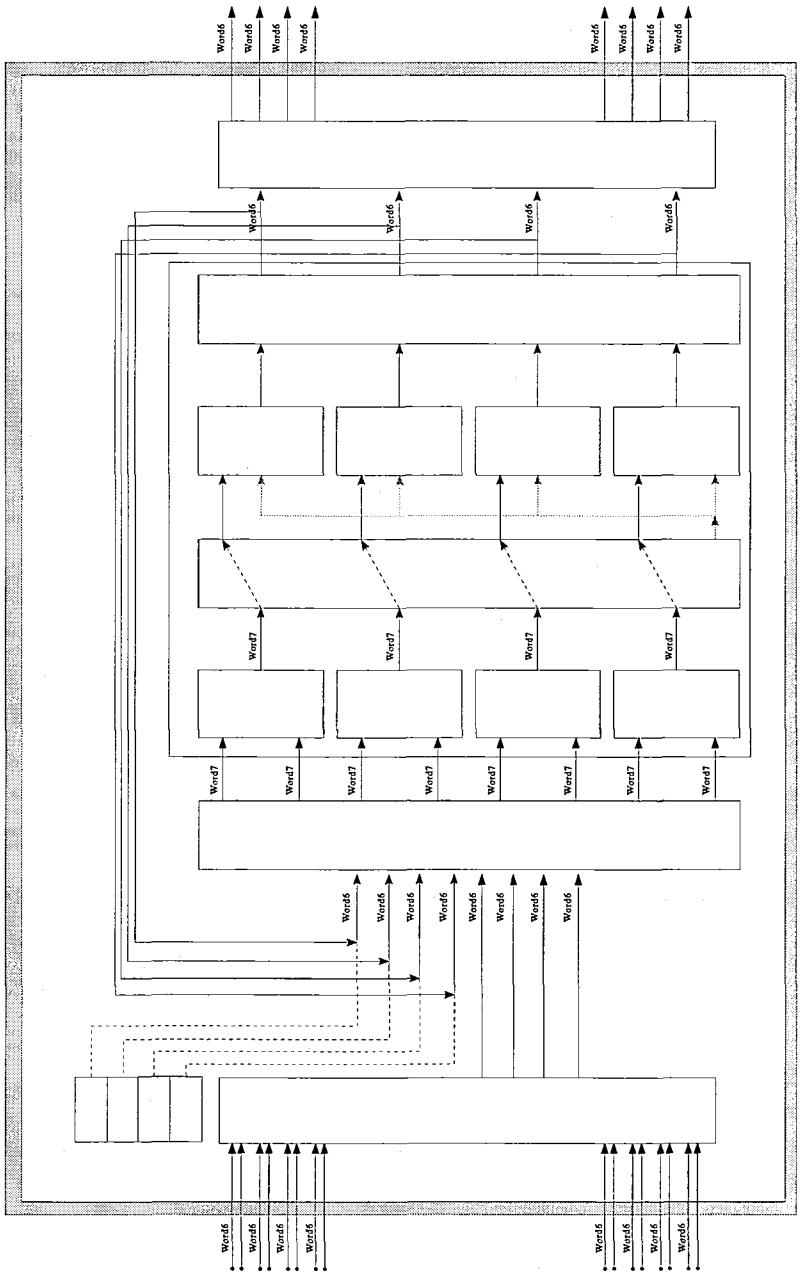



Figure 5.14: Alpha Architecture - Data Type Structures

Regarding Figure 5.13, there is local logic that controls the sequence of iterative events, i.e., logic that initially feeds in locally stored variables (top left of schematic) and then feeds the input the iterative output of the system (counter). $X = x$, where initially x=19 and decrements to 0.

All sub-blocks in alpha, function the same as the blocks in Beta.

Recall, equation 3.27, excluding the normalization correction. The normalization correction will be discussed in Section 5.3.7.

$$\beta_t(m) = \sum_{m'=0}^{N-1} \beta_{t+1}(m') \cdot \Gamma_{t+1}(m, m')$$

Using the same procedure in Section 5.3.4, in the logarithmic domain the equation becomes,

$$\ln \beta_t(m) = \ln \sum_{m'}^{N-1} e^{\ln(\beta_{t+1}(m')) + \ln(\Gamma_{t+1}(m', m))} \tag{5.10}$$

Let,

$$\begin{aligned} \delta_1 &= \ln \beta_{t+1}(m') \\ \delta_2 &= \ln \Gamma_{t+1}(m', m) \end{aligned}$$

and by using equation 5.1, the previous equation 5.10 becomes,

$$\ln \beta_t(m) = MAX^*_{m'}(\delta_1, \delta_2) \tag{5.11}$$

Rewriting, this equation becomes,

$$\ln \beta_t(m) = MAX_{m'}(\delta_1, \delta_2) + \ln\left(1 + e^{-|\delta_1 - \delta_2|}\right) \tag{5.12}$$

Since the Max-Log-MAP algorithm will be used for this design, the equation can be rewritten minus the correction factor,

$$\ln \beta_t(m) = MAX_{m'}(\delta_1, \delta_2) \tag{5.13}$$

### 5.3.6 LLR Architecture

The LLR architecture that was implemented is shown in Figure 5.15. The data types used in the LLR system are shown in Figure 5.16. The sub-system architecture for the LLR sub blocks are shown in Figure 5.17 and Figure 5.18 . For full coding details see Appendix C.

Since a NRC Encoder is being used for this study, $\lambda_t(m)$ will be used to find the APPs of the symbols. Recall equation 3.31,

$$\lambda_t(m) = \alpha_t(m) \cdot \beta_t(m)$$

Using the same procedure in Section 5.3.4 and 5.3.5, in the logarithmic domain the equation becomes,

$$\ln \lambda_t(m) = MAX(\delta_1, \delta_2) \tag{5.14}$$

where

$$\begin{aligned} \delta_1 &= \ln \alpha_t(m) \\ \delta_2 &= \ln \beta_t(m) \end{aligned}$$

Recalling equation 3.33, we can substitute in the previous equation,

$$Pr_{APPlog}[u_t = 1] = \frac{\sum_{m \in A} MAX(\delta_1, \delta_2)}{\sum_{m \in All\ states} MAX(\delta_1, \delta_2)} \tag{5.15}$$

Recall equation 5.16,

$$Pr_{APPlog}[u_t = -1] = 1 - Pr_{APPlog}[u_t = 1] \tag{5.16}$$

Therefore, the decision rule for the hard output remains the same and is as follows,

$$\hat{u}_t = \begin{cases} 1 & \text{if } Pr_{APPlog}[u_t = 1] \geq Pr_{APPlog}[u_t = -1] \\ -1 & \text{if } Pr_{APPlog}[u_t = 1] < Pr_{APPlog}[u_t = -1] \end{cases} \tag{5.17}$$

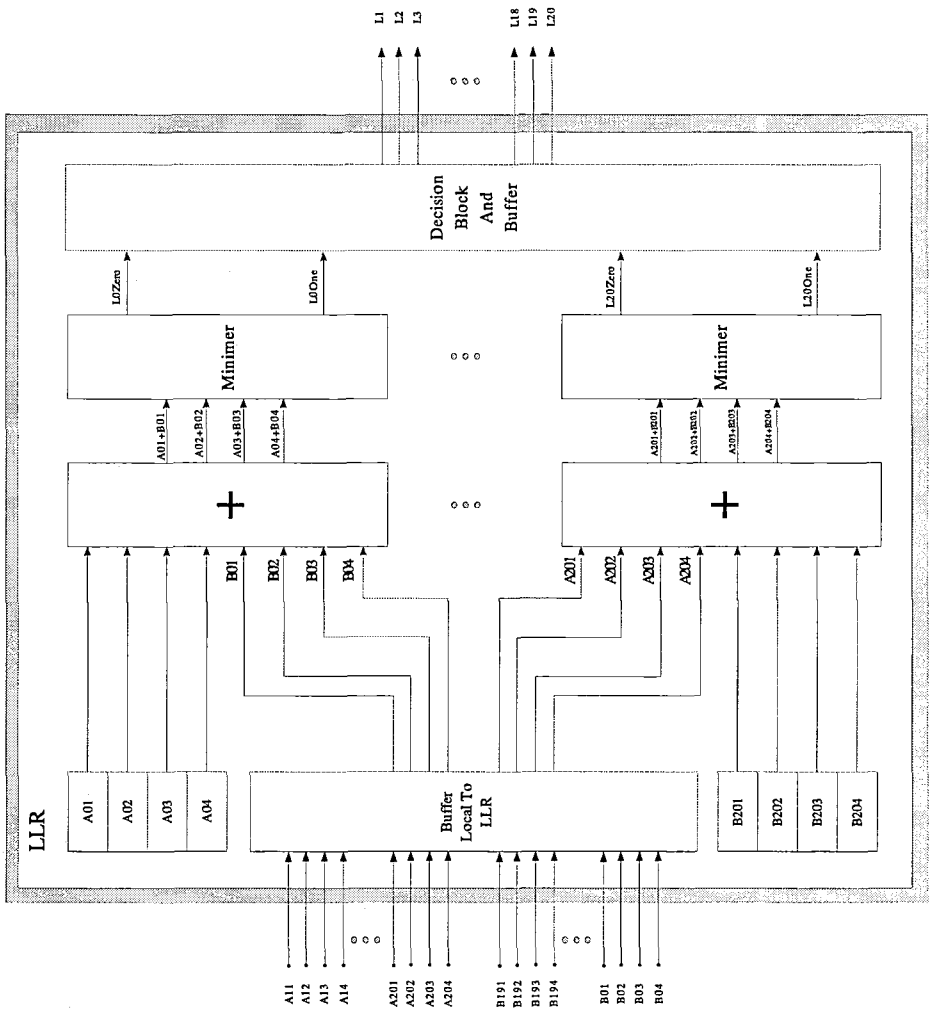



Figure 5.15: LLR Architecture

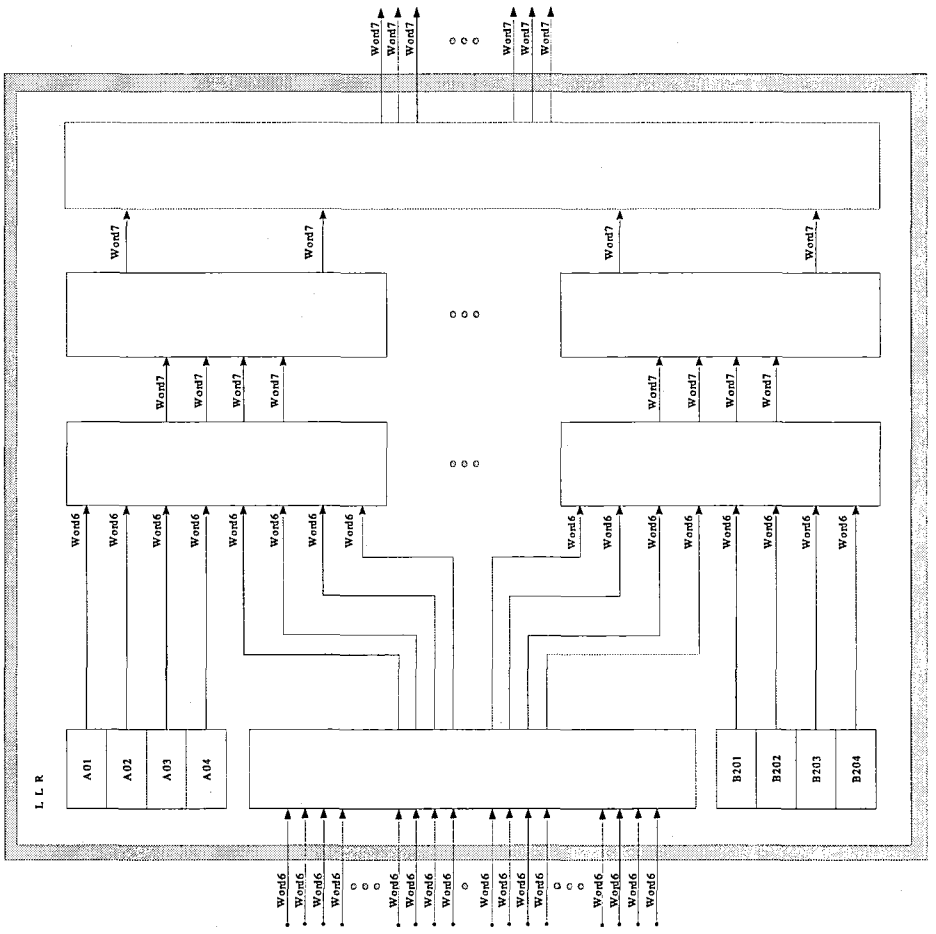



Figure 5.16: LLR Architecture - Data Type Structures

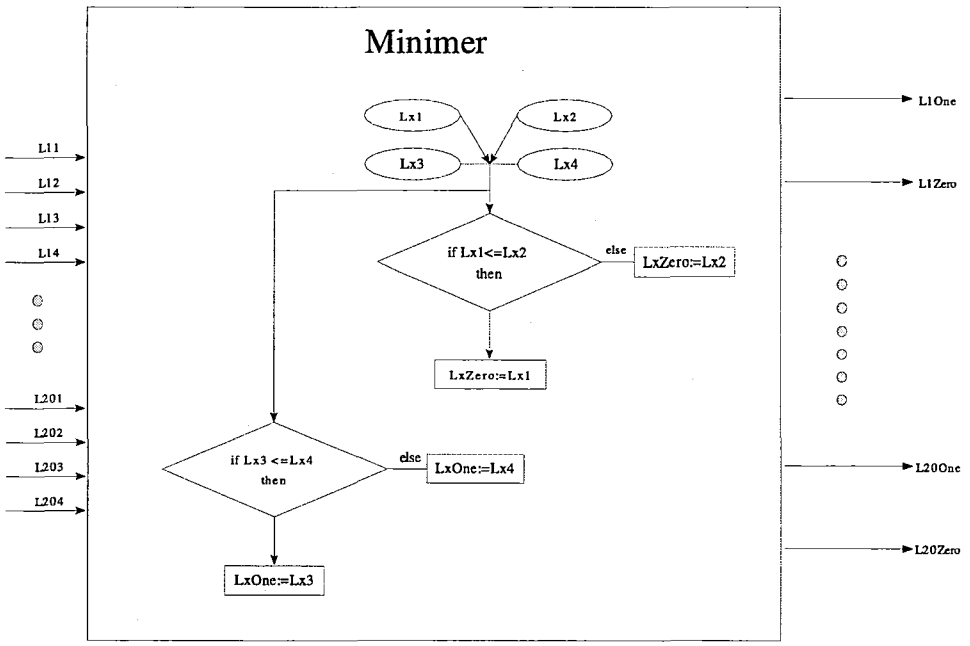



Figure 5.17: LLR Sub-System Architecture - Minimer

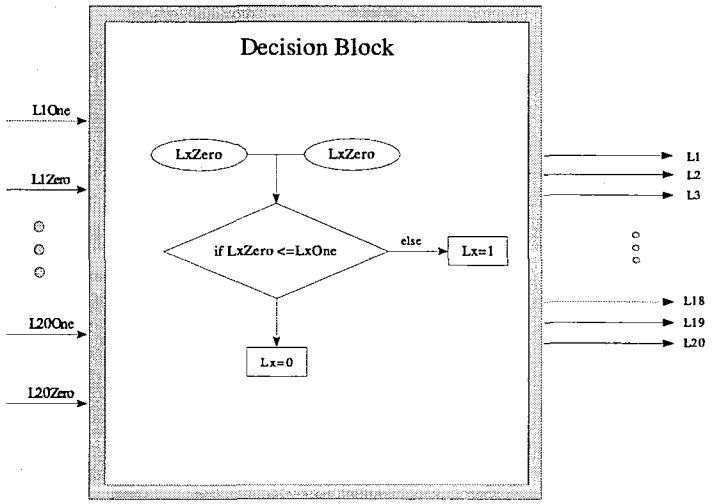



Figure 5.18: Alpha Sub-System Architecture - Decision Block

### 5.3.7 Normalization And The Positive Domain

Normalization was discussed previously in Section 3.7.1 and 3.7.2. One of the best techniques for normalization, for instance to prevent overflow, is as follows,

Let,

$$X = \{x_1, x_2, \ldots, x_{n-1}, x_n\} \tag{5.18}$$

be a set of numbers that need to be normalized.

Then the normalization of each term, denoted by, $\bar{x}$, is as follows,

$$\bar{x}_1 = \frac{x_1}{\sum_{k=1}^{k} x_k} \tag{5.19}$$

$$\vdots = \vdots$$

$$\bar{x}_n = \frac{x_n}{\sum_{k=1}^{k} x_k} \tag{5.20}$$

where $k = \{1, 2, \ldots, K-1, K\}$ and $K$ is the number of elements in the set $X$.

For VLSI implementations of normalization, a division, which is basically a multiplication, is large in terms of area. Different normalization techniques are needed. For numbers between $0, 1, 2, \ldots, \infty$, the following could be used.

Let,

$$X = \{x_1, x_2, \ldots, x_{n-1}, x_n\} \tag{5.21}$$

be a set of numbers that need to be normalized.

Then the normalization of each term, denoted by, $\bar{x}$, is as follows,

$$\bar{x}_1 = x_1 - MAX\,(X) \tag{5.22}$$

$$\vdots = \vdots$$

$$\bar{x}_n = x_n - MAX\,(X) \tag{5.23}$$

For a set of numbers in the negative domain, i.e., number between $-\infty, \ldots, 2, 1, 0$, the same rule applies. For the normalization of the $\alpha$ and $\beta$ values, equations 5.22 need to be used.

The values coming from the Gamma($\Gamma$) system block are always negative. Therefore, to save area and speed, all subsequent sub systems, i.e., alpha, beta and the LLR, can use the values all converted into the positive domain. However, since the values were originally negative, the normalization must take this into account. Therefore, for normalization, instead of maximum, the minimum must be taken. This also applies for all maximum comparisons in the alpha and beta calculations. The decision rule for hard outputs, i.e: equation 5.17 has to also be altered to accommodate this conversion to positive values.

To make a comment on area and speed savings, the system will now not need to employ two's complement arithmetic. Since the alpha,beta and LLR calculations do require additions. Using an extra signed bit and two's complement arithmetic throughtout the calculations needed will yield extra area, slower speeds and complexity that can be avoided using the simple conversion to the positive domain.

# Chapter 6

# *Simulation Architecture and Simulation Results*

## 6.1 Software Tools - Verilog, Synopsys and MatLab

The simulation step in design is most crucial for determining functionality, performance and feasibility. Different software tools were used including MatLab, Balsa, Verilog and Synopsys.

MatLab is "a high-level technical computing language and interactive development environment" [26]. It was used for functionality and best case performance evaluation.

Balsa, the asynchronous HDL was converted to gate-level Verilog code during one of the asynchronous design flow steps. Verilog emerged in 1983 at Gateway Design Automation. Verilog is a "general-purpose" HDL which uses syntax similar to C programming. It utilizes different levels of abstraction in the same model. A designer can code a circuit in terms of switches, gates, Register Transfer Level (RTL) or behavioural code. One language needs to be learned to create 'hierarchical' design and a stimulus harness [27].

Once the mapping from Balsa to Verilog occurred, the synthesis of Verilog code to a gate-level netlist was needed so that area, power and timing could be analyzed. Synopsys

performs synthesis and area, power and timing reports. Synopsys is a full set of tools that not only performs HDL simulation but full RTL coding solutions [28].

## 6.2 MatLab Simulation Architecture

The most important step in the design of the asynchronous BCJR/MAP decoder was that of functionality testing of the Balsa designed decoder. A method of automation was needed and MatLab was a perfect fit due to the ability to execute UNIX commands.

Balsa has a built in test harness Graphical User Interface (GUI) that allows the user to give inputs and outputs to the Balsa designed system. The Balsa design suite can also be executed without the GUI. By combining the results of the Balsa BCJR/MAP decoder, which used the test harness program, with that of other systems implemented in MatLab, the design was thoroughly tested for functionality and best case performance. The performance was measured in BER against SNR.

Figure 6.1 demonstrates an early design where the performance of the system was either not met or the system still had incorrect code that needed to be altered. The axes of the figure are BER vs. SNR.

Figure 6.2 demonstrates the software simulation architecture for functionality and performance testing. Within MatLab, random numbers were generated and then encoded. These encoded values were then put through a AWGN channel using BPSK modulation. These values were then passed onto previously designed MatLab BCJR/MAP decoders. These same values were then written to file. Using UNIX commands within MatLab, the Balsa simulator used the values from file and gave an output that was also written to file. The values produced from Balsa and the previously designed decoders were then used to calculate the BER values against the SNR values of the channel. The different decoders names are listed in the following figures as:

1. **uncoded** - This is the only system where the values are not encoded or decoded. This represents the worst possible scenario. Any type of encoding and decoding should improve the performance.

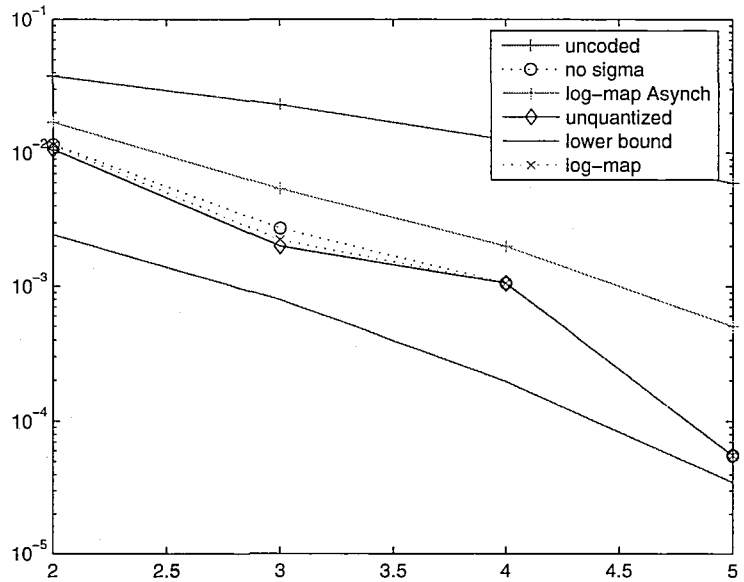



Figure 6.1: 1000 Blocks Transmitted Per dB Level - BER Vs. SNR - Invalid Balsa Design

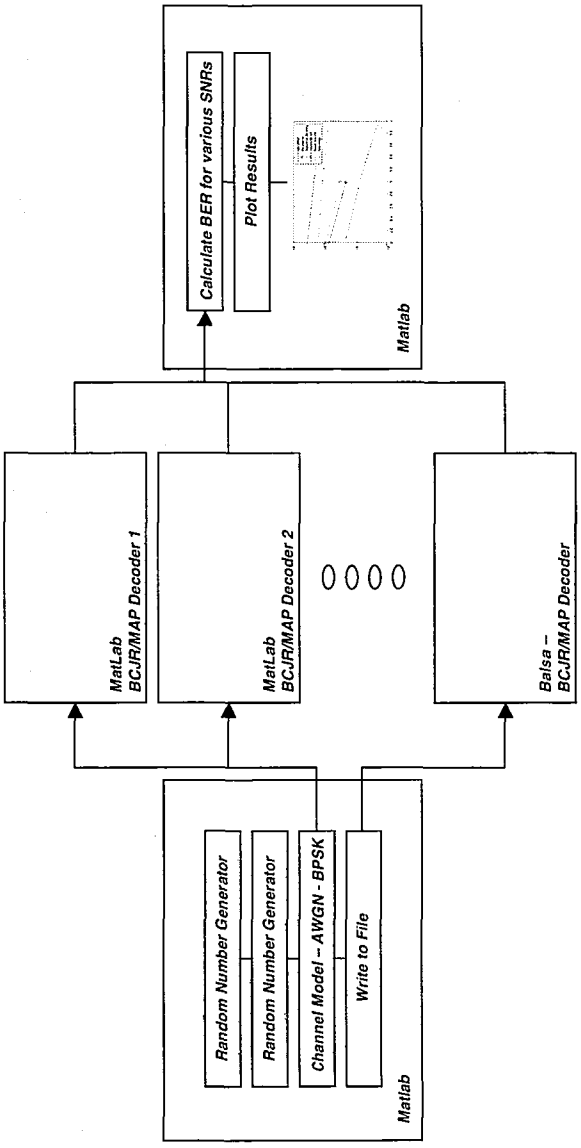





Figure 6.2: MatLab System Simulation Architecture

2. **unquantized** - This BCJR/MAP system does not use quantized values. This system represents the best possible encoding-decoding scheme realizable.

3. **log-map** - This BCJR/MAP system uses 6-bit quantized values. This system represents a more realistic decoder, the best possible realization of a hardware design.

4. **no sigma** - This is the same BCJR/MAP system as the 'log-map', however, the system has been simplified. It achieves the same performance as the 'log-map' system.

5. **log-map Asynch** - These are Balsa simulator test results of the design under test.

6. **lower bound** - This is a system that contains no errors after decoding. It is the best possible scenario but unrealistic.

The optimal performance was obtained through the design named trunc_5bit. The 5-bit truncation refers to the number of bits represented after the LUT in the Gamma block, see Appendix C. Figure 6.3 demonstrates that 'log-map Asynch' achieves approximately the same level of performance as 'log-map' and 'no sigma' systems. The LUT in trunc_5bit is fed a 6-bit word and then is coverted to a 5-bit word.

The design named trunc_4bit is a decoder that uses a 6-bit to 4-bit LUT. At the expense of performance, the trunc_4bit design, see Figure 6.4, would most likely decrease the area and power metrics. This was roughly estimated with the Balsa tool called breeze-cost. This tool estimates the area cost of the circuit. These are rough estimates, a more thorough gate-level Synopsys area value would be more effective. The values produced by breeze-cost are shown in Table 6.1. This estimate shows a possible reduction in area of 10.2%. This reduction is due to the 'new' data type truncation being carried through the remainder of the design.

## 6.3 Synopsys Synthesis Simulation Results

The step from Balsa to Verilog code was not supported automatically by the creators at the University of Manchester. This was partly because the gate-level netlist contained technology specific cells and they did not support the same technology that the RCIM

Table 6.1: breeze-cost Estimates For Area (Units Are Normalized)

| | trunc_6bit | trunc_5bit | trunc_4bit |
|---|---|---|---|
| Full System BJR/MAP | 233597.5 | 212015.5 | 190433.5 |

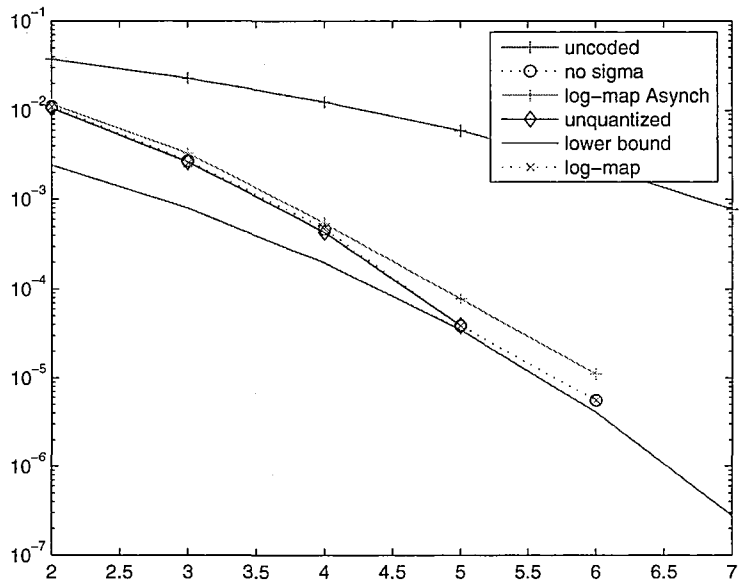



Figure 6.3: 10000 Blocks Transmitted Per dB Level - BER Vs. SNR - trunc_5bit Design

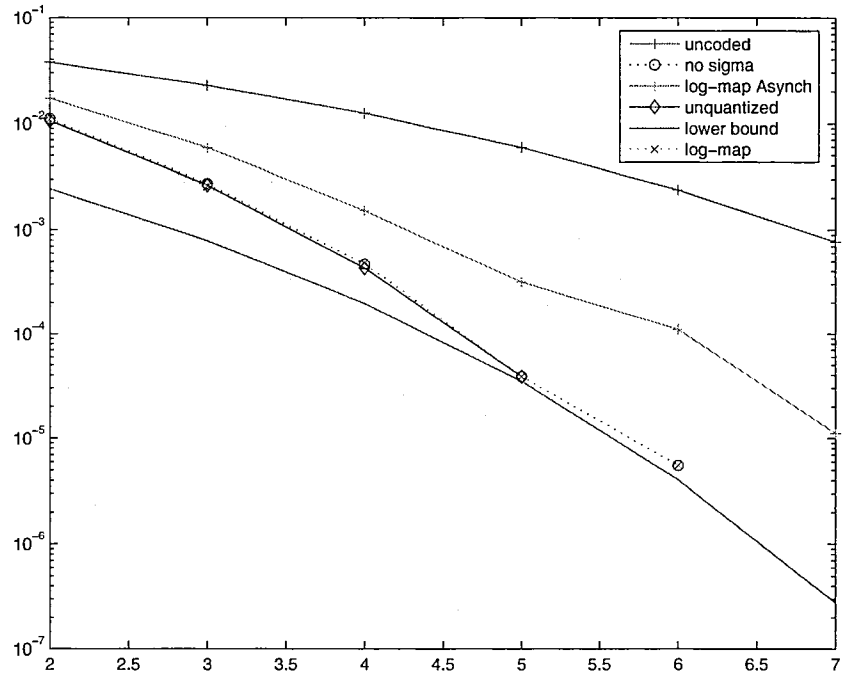



Figure 6.4: 10000 Blocks Transmitted Per dB Level - BER Vs. SNR - trunc_4bit Design

Table 6.2: Area, Power And Timing Values

| | Area ($\mu m^2$) | Power ($mW$) | Delay ($ns$) |
|---|---|---|---|
| Overall | 6552259.5000 | 218.0502 | 254.35 |
| Gamma | 463780.0000 | - | - |
| Alpha | 2041387.3750 | - | - |
| Beta | 2461549.2500 | - | - |
| LLR | 796769.2500 | - | - |
| Other | 788773.6250 | - | - |

group uses. Therefore, adding the TSCM 01.8 micron technology was needed to use the Synopsys tools available to the RCIM group. The files needed for this mapping are included in Appendix D. The process is discussed in further detail in [19].

The process of obtaining gate-level Verilog code was known as an 'implementation'. The implementation that was created used the 4PBDP. The 4PBDP can also have variations as previously discussed. The variation of 4PBDP that was used was the broad variation, see Figure 2.7.

Once the design was mapped to a gate-level netlist, the gate-level netlist could be used to generate area, power and timing results. The software tool that derived these reports is Synopsys' 'Design Analyzer'. These results are highlighted in Table 6.2. The complete reports for area, power and timing are shown in Appendix F.

The area values reported showed an overall 'gate' size of 6552259.5 $\mu m^2$. The area of improvement should be dedicated to the LLR and Balsa tools which generate all the 'Other' circuitry required. The chip gate dimension was 2.560 mm X 2.560 mm.

The power value shown is referred to as the 'total dynamic power'. This is the total power consumed by the chip and it is the sum of 'cell internal power' and 'net switching power'. The 'cell internal power' was 95.9450 $mW$ (44%) and the 'net switching power' was 122.1051 $mW$ (56%). The 'cell leakage power' was 43.9936 $\mu W$ and thus is considered negligible. The power of the submodules of the BCJR/MAP decoder were not applicable

Table 6.3: Comparison To Synchronous MAP Decoder Designs

| Authors | Rate | Algorithm | Block Length | Tech. ($\mu m^2$) | Area ($mm^2$) | Power ($\mu W$) |
|---|---|---|---|---|---|---|
| Vogt | $\frac{1}{2}$ | ML-MAP | 668 | 0.35 | 2.8 | - |
| Vogt | $\frac{1}{3}$ | SOVA-OU | 668 | 0.35 | 2.5 | - |
| Bicherstaff | $\frac{1}{3}$ | L-MAP | 2568 | 0.35 | 9 | - |
| Sabeti | $\frac{1}{2}$ | ML-MAP | 512 | 0.18 | 1.16 | 59.66 |
| Perta | $\frac{1}{2}$ | ML-MAP | 20 | 0.18 | 6.55 | 218.05 |

since the design is constrained at the top level. If the values of the submodules were to be found, the sum of their parts would exceed the total power of the whole system. This has to do with the way the simulator derives the power value.

The worst case path delay of the BCJR/MAP decoder is 254.35 ns. The values obtained are 'conservative' (default) constraints placed on the system. Constrained area and timing outcomes may result in better values.

In [29], a comparison of different MAP decoders were presented. These findings are presented alongside the proposed BCJR/MAP decoder in Figure 6.3.

These results show that the asynchronous digital design technology and tools are not yet up to par with the standard clocked tools. The tools optimize clock networks and this has serious ramifications on mapped systems that exhibit an asynchronous topology.

The large area value can be explained because non-optimized Random Access Memory (RAM) is being synthesized. These RAM blocks are being implemented by using small primitive cells. If there was an asynchronous optimized cells used in the mapping from Balsa to Verilog, this would improve the area and power consumption. Since the BCJR/MAP decoder for hardware implementation required large amounts of RAM, this was reflected in the larger than normal area values. In [29], optimized RAM blocks were used and this did comparatively reduce area.

### 6.3.1 Problems Encountered And Possible Remedies

The Balsa to Verilog mapping was troublesome. When the conversion was finally completed, Synopsys detected cells that were placed without loads. These cells were removed, see the fully generated Verilog code in Appendix E. The commented code reflects the cells that needed to be removed to obtain area, power and timing. Since the cells were removed, functionality at a gate-level becomes a concern. Balsa simulation is not sufficient in terms of functionality testing until these concerns are addressed. Gate-level testing needs to be employed to fully test the functionality of the system if the mapping behaves erratically.

The Balsa suite offers two other protocol implementations: a delay-insensitive dual-rail encoding and a delay-insensitive one-of-four encoding. These other protocols require serious file manipulation (or 'hacking') in order that they map the technology properly. The sheer amount of work that is required is daunting due to the 'trial and error' required methodology to fix the files.

The immense amount of small cells generated by the Balsa to Verilog netlist created memory problems with the well equipped Sun Workstations. The Sun stations often times ran out of memory. Adressing this with inventive constraint parameters will make the generation of results less volatile.

### 6.3.2 Future Work

Some future studies could include the already addressed issues, as well as the following. The Balsa language is a computer scientist's solution to the asynchronous digital circuit automation for VLSI design. The separate language and system for simulation and synthesis is satisfactory, however, existing electrical engineers that are fluent in digital design should not have to learn a brand new language. Verilog can be used to design asynchronous systems, however, they lack several features that the CSP languages offer. A solution should be somewhere in between.

From a practical standpoint, training engineers to learn a new language requires time and money. However, if a system could be designed that uses existing synchronous RTL code (Verilog or VHDL) which is mapped to asynchronous gate-level code, the solution

would seem a little more practical. The biggest problem is the unsupported tools for the asynchronous domain.

Most digital asynchronous designers are physically laying-out (transistor level) their systems. This becomes analog design and is only plausible for companies and institutions with many engineers and large sums of money. When the technology changes, i.e.: feature size of transistor shrinks, these designs need to be redesigned. The automation of VLSI asynchronous digital design is the sector that needs to be researched, not analog 'digital' physical layout because this will yield more efficient systems and designs.

An automated way of taking existing synchronous code (Verilog or VHDL) and converting it to gate-level asynchronous code is a needed area of research. This should also include the option of giving the designer different asynchronous design styles and the choice of algorithms to achieve low-power and smaller area systems.

Further exploration is needed past the gate-level step to uncover other advantages that the asynchronous paradigm offers. Since this thesis only examined area and power, a larger study needs to be conducted in the other beneficial areas of asynchronous systems. This can include lower noise systems and speed.

Once asynchronous design is recognized as a viable option, it will slowly amalgamate with synchronous systems offering solutions to existing unsolvable problems, e.g., clock skew and interconnect scaling issues. This will then give the required time for academia and industry to address low power and other algorithmic remedies needed in digital automated asynchronous design.

# Chapter 7

# *Summary Of Contributions and Conclusion*

This study has provided a foundation into the investigation of asynchronous digital design using a BCJR/MAP channel decoder as the design example. The results were encouraging. The following sections will summarize the results.

## 7.1 Asynchronous VLSI

An asynchronous digital design flow has been investigated. A BCJR/MAP channel decoder has been designed in an asynchronous HDL (Balsa). This was converted to Verilog gate-level code for analysis with Synopsys. Most asynchronous designs employ a physical layout of the circuit. This level of design is expensive and not for small institutions and businesses. Employing a strategy that automates digital asynchronous design, such as Balsa, yields metrics that will eventually improve designs that are heavily constrained.

## 7.2 Simulation Architecture

The MatLab / Balsa simulation architecture that was implemented was a quick and efficient method for functionality and best case performance evaluation. The BCJR/MAP decoder was fully tested for functionality and best case performance and met the standard. For future designs, this simulation architecture will improve the quickness needed to determine feasibility of an asynchronous design.

## 7.3 BCJR/MAP Channel Decoding

The BCJR/MAP channel decoder was implemented and the results were encouraging. The level of refinement that asynchronous tools need to achieve is a daunting and difficult task. The application of these new tools, however, need to be tested on useful technologies, e.g., Turbo Codes, to improve all areas of design.

Although the BCJR/MAP decoder meets reasonable area standards, other benefits including lowered noise could catapult asynchronous decoding into the forefront of space communications. Space systems require data sensitive communication exchange that would highly benefit from the use of a lower noise and power system. Furthering the design flow beyond the gate-level layer will aid in revealing the benefits of asynchronous design.

## 7.4 Conclusion

The BCJR/MAP decoder has been synthesized to the gate-level layer of the digital design flow. The results for gate area were 2.560 mm X 2.560 mm. The power consumption of 218.0502 $\mu$W was a promising result.

This asynchronous BCJR/MAP decoder will hopefully inspire others to examine the asynchronous paradigm as a viable alternative to clocked systems.

# References


[1] Erico Guizo,"Closing In On The Perfect Code",IEEE Spectrum Magazine,March 2004

[2] C.H. Van Berkel, M.B. Josephs, and S.M. Nowick, Scanning the Technology: Applications of Asynchronous Circuits", Proceedings of the IEEE, Volume 87, Issue 2, pp. 223-233, 1999

[3] J. Sparso, S. Furber, "Principles of Asynchronous Circuit Design: A System Perspective", Kluwer Academic Publishers, 2001

[4] C. J. Myers, "Asynchronous Circuit Design", John Wiley and Sons, Inc., 2001

[5] S. Furber, "The Return of Asynchronous Logic", http://www.cs.man.ac.uk/async/background/return_async.html

[6] "Keynoter Sees Asynchronous Future For Digital Designs", EE Times, http://www.eetimes.com/story/OEG20021204S0018, December 4, 2002

[7] I. Sutherland and J. Ebergen, "Computers Without Clocks", http://www.sciam.com/article.cfm?articleID=00013F47-37CF-1D2A-97CA809EC588EEDF

[8] L. E. M. Brackenbury, M. Cumpstey and S.B. Furber, "Applying Asynchronous Techniques to a Viterbi Decoder Design", IEEE Seminar on Low Power IC Design, 2001

[9] K. E. Tepe, "Iterative Decoding Techniques for Correlated Rayleigh Fading and Diversity Channels", Communication, Information and Voice Processing Report Series, Report TR-2001-1, University of Lund - Information Technology Department and Rensselaer Polytechnic Institute - Electrical, Computer and System Engineering Department, 2001

[10] R. B. Wells, "Applied Coding and Information Theory for Engineers", Prentice Hall, 1999

[11] S. Shahzad Shah, S. Yaqub and F.l Suleman, "Self-Correcting Codes Conquer Noise, Part One: Viterbi Codecs", EDN Magazine, http://www.reed-electronics.com/ednmag/contents/images/75255.pdf, 2001

[12] P.A. Riocreux, L.E.M. Brackenbury, M. Cumpstey and S.B. Furber, "Low-Power Self-Timed Viterbi Decoder", Seventh International Symposium on Asynchronous Circuits and Systems, 2001

[13] I. Sutherland, "Micropipelines: Turing Award Lecture", Communications of the ACM, Volume 32, no. 6, pp. 720-738, 1989

[14] D.W. Lloyd and J.D. Garside, "A Practical Comparison of Asynchronous Design Styles", Seventh International Symposium on Asynchronous Circuits and Systems, pp. 36-45, 2001

[15] D. Muller and W. Bartky, "A Theory of Asynchronous Circuits", Proceedings of an International Symposium on the Theory of Switching,pp. 204-243, 1959

[16] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo Codes", Proceedings 1993 IEEE International Conference on Communications, pp. 1064-1070, 1993

[17] John G. Proakis and Masoud Salehi, "Communication Systems Engineering, 2nd Edition", Prentice Hall, 2002

[18] R. Goering,"Keynoter Sees Asynchronous Future For Digital Designs", EE Times, December 4, 2002

[19] Doug Edwards, Andrew Bardsley, Lilian Janin and Will Toms, "Balsa: A Tutorial Guide", Version V3.4.1, 2004

[20] "APT Website", http://www.cs.man.ac.uk/apt/index.html

[21] N. G. Kingsbury and P. J. W. Rayner, "Digital Filtering Unsing Logarithmic Arithmetic", Electronic Letters, Volume 7, Number 2, pp. 56-58, 1971

[22] J. A. Erfanian and S. Pasupathy, "Low-Complexitity Parallel-Structure Symbol-by-Symbol Detection for ISI Channels", IEEE Transaction on Information Theory, Volume 41, Number 3, pp. 704-713, 1995

[23] Emmanuel Boutillon, Warren J. Gross and Glenn Gulak, "VLSI Architectures for the Forward-Backward Algorithm", http://lester.univ-ubs.fr:8080/ boutillon/sanchezt/ForBack_article.ps

[24] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimixzing Symbol Error Rate", International Symposium on Information Theory, pp. 284-287, 1972

[25] Patrick Robertson, Emmanuelle Villebrun and Peter Hocher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain", Proceedings of the IEEE ICC, pp. 1009-1013, 1995

[26] "The MathWorks - MATLAB - The Language of Technical Computing", http://www.mathworks.com/products/matlab

[27] Samir Palnitkar, "Verilog HDL - A Guide to Digital Design and Synthesis", Sun Soft Press - Prentice Hall, 1996

[28] "Synopsys", http://www.synopsys.com

[29] Leila Sabeti, University of Windsor - Masters Thesis : New VLSI Design Of A Max-Log MAP Decoder", 2004

# Appendix A

## *List of Abbreviations*

| | |
|---|---|
| **2PBDP** | 2 Phase Bundle Data Protocol |
| **2PDRP** | 2 Phase Dual Rail Protocol |
| **4PBDP** | 4 Phase Bundled Data Protocol |
| **4PDRP** | 4 Phase Dual Rail Protocol |
| **ack** | Acknowledge |
| **APP** | A Posterior Probabilities |
| **APT** | Advanced Processor Technologies |
| **AS** | Asynchronous Systems |
| **AWGN** | Additive White Gaussian Noise |
| **BCJR** | Bahl, Cocke, Jelinek, and Raviv |
| **BEC** | Binary Erasure Channel BEC |
| **BER** | Bit Error Rate |
| **BD** | Bundled Data |
| **BDP** | Bundled Data Protocol |
| **BMU** | Branch Metric Unit |
| **BPSK** | Binary Phase Shift Keying |
| **BSC** | Binary Symmetric Channel |

| | |
|---|---|
| **CAD** | Computer Aided Design |
| **CC** | Convolutional Coding or Codes |
| **CPU** | Central Processing Unit |
| **CSP** | Communicating Sequential Processes |
| **CMOS** | Complementary Metal Oxide Semiconductor |
| **DMC** | Discrete Memoryless Channel |
| **DR** | Dual Rail |
| **DRP** | Dual Rail Protocol |
| **EDA** | Electronic Design and Automation |
| **EM** | Electro Magnetic |
| **FB** | Forward and Backward |
| **FPGA** | Field Programmable Gate Arrays |
| **FIR** | Finite Impulse Response |
| **FSM** | Finite State Machine |
| **GUI** | Graphical User Interface |
| **HC** | Handshake Component |
| **HDL** | Hardware Definition Language |
| **HU** | History Unit |
| **IIR** | Infinite Impulse Response |
| **LLR** | Log Likelihood Ratio |
| **LUT** | Look Up Table |
| **MAP** | Maximum A Posteriori |
| **MP** | Muller Pipeline |
| **NRC** | Non Recursive Convolutional |
| **NRTZ** | Non Return To Zero |
| **NRTZBDP** | Non Return To Zero Bundled Data Protocol |
| **PMU** | Path Metric Unit |
| **PREST** | Power Reduction for System Technologies |
| **RAM** | Random Access Memory |

| | |
|---|---|
| **RCIM** | Research Center for Integrated Microsystems |
| **req** | Request |
| **RSC** | Recursive Systematic Convolutional |
| **RTL** | Register Transfer Level |
| **RTZBDP** | Return To Zero Bundled Data Protocol |
| **RTZ** | Return To Zero |
| *Si* | The Chemical Element Silicon |
| **SMU** | Survival Metric Unit |
| **SNR** | Signal To Noise Ratio |
| **SS** | Synchronous Systems |
| **TSCM** | Taiwan Semiconductor Manufacturing Company |
| **UNIX** | UNiplexed Information and Computing System |
| **VA** | Viterbi Algorithm |
| **VHDL** | Very High Speed Integrated Circuit Hardware Description Language |
| **VLSI** | Very Large Scale Integration |
| **XOR** | Exclusive 'OR' |

# Appendix B

*Matlab Code - see enclosed CD*

# Appendix C

*Balsa Code - see enclosed CD*

# Appendix D

## *Balsa To Verilog Netlist Mapping Files: For TSCM 0.18 micron, Single Poly, Six Metal, Salicide CMOS Process - see enclosed CD*

# Appendix E

*Verilog Code - see enclosed CD*

# Appendix F

## *Synopsys Area, Power And Timing Report Files - see enclosed CD*

# *VITA AUCTORIS*

Kristofer Perta received his B.A.Sc. degree in Electrical Engineering from the University of Windsor and graduated 2001. His undergraduate co-op degree gave him the opportunity to take placements outside of his hometown to Toronto and Ottawa with Nortel Networks and JDS Uniphase.

In the fall of 2002, Kris began his graduate degree under the supervision of Dr. Tepe in the fields of digital VLSI design and convolutional channel coding.

Kris plans on pursuing a career in Hardware Design in the Telecommunications sector after the completion of his graduate degree.