A VITERBI DECODER USING SYSTEM C FOR AREA EFFICIENT VLSI IMPLEMENTATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERKAN SÖZEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

_____

Prof. Dr. İsmet Erkmen

Head of the Department

This is to certify that we have read this thesis and in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Murat Aşkar

Supervisor

Examining Committee Members

Prof. Dr. Tayfun Akın	(METU, EE)	_____
Prof. Dr. Murat Aşkar	(METU, EE)	_____
Assoc. Prof. Dr. Aydın Alatan	(METU, EE)	_____
Assist. Prof. Dr. Çağatay Candan	(METU, EE)	_____
Hayrettin Kesim	(ASELSAN A.S.)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Serkan Sözen

Signature :

ABSTRACT

A VITERBI DECODER USING SYSTEM C FOR AREA EFFICIENT VLSI IMPLEMENTATION

Sözen, Serkan

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Murat Aşkar

September 2006, 153 pages

In this thesis, the VLSI implementation of Viterbi decoder using a design and simulation platform called SystemC is studied. For this purpose, the architecture of Viterbi decoder is tried to be optimized for VLSI implementations. Consequently, two novel area efficient structures for reconfigurable Viterbi decoders have been suggested.

The traditional and SystemC design cycles are compared to show the advantages of SystemC, and the C++ platforms supporting SystemC are listed, installation issues and examples are discussed.

The Viterbi decoder is widely used to estimate the message encoded by Convolutional encoder. For the implementations in the literature, it can be found that special structures called trellis have been formed to decrease the complexity and the area.

In this thesis, two new area efficient reconfigurable Viterbi decoder approaches are suggested depending on the rearrangement of the states of the trellis structures to eliminate the switching and memory addressing complexity.

The first suggested architecture based on reconfigurable Viterbi decoder reduces switching and memory addressing complexity. In the architectures, the states are reorganized and the trellis structures are realized by the usage of the same structures in subsequent instances. As the result, the area is minimized and power consumption is reduced. Since the addressing complexity is reduced, the speed is expected to increase.

The second area efficient Viterbi decoder is an improved version of the first one and has the ability to configure the parameters of constraint length, code rate, transition probabilities, trace-back depth and generator polynomials.

Keywords: Viterbi Algorithm, Reconfigurable Viterbi Decoder, SystemC, Convolutional Encoder, Maximum Likelihood Method.

ÖZ

SYSTEM C KULLANILARAK BİR VITERBI KOD ÇÖZÜCÜSÜNÜN ALANI VERİMLİ TÜMDEVRE OLARAK GERÇEKLENMESİ

Sözen, Serkan

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Murat Aşkar

Eylül 2006, 153 sayfa

Bu tez çalışmasında, SystemC olarak bilinen tasarım ve simülasyon ortamı kullanılarak Viterbi Kod Çözücüsünün tümdevre gerçeklenmesi üzerine çalışılmıştır. Bu amaçla, tümdevre gerçeklemesine yönelik Viterbi Kod Çözücüsünün mimarisi iyileştirilmeye çalışılmıştır. Sonuç olarak, iki yeni alan bakımından verimli biçimlendirilebilir Viterbi kod çözücü yapı önerilmiştir.

SystemC'nin avantajlarını sergilemek için geleneksel ve SystemC tabanlı tasarım aşamaları karşılaştırılmış ve SystemC'yi destekleyen C++ ortamları listelenmiş, örneklerle kurulumundan bahsedilmiştir.

Viterbi kod çözücü sıklıkla Evrişimsel şifrelenmiş mesajların yakınsanmasında kullanılmaktadır. Kaynaklardaki gerçeklemelerde özel kafes yapılar biçimlendirilerek karmaşıklığın ve alanın azaltılmasına çalışıldığı görülmektedir.

Bu tezde, anahtarlama ve hafıza adreslemesindeki karmaşayı yok etmek için kafes yapısının statülerinin yeniden düzenlenmesine dayanan iki yeni alan bakımından verimli biçimlendirilebilir Viterbi kod çözücü yaklaşımı önerilmektedir.

Biçimlendirilebilir Viterbi Kod çözücüsü için önerilen ilk mimari, anahtarlama ve hafıza adreslemesindeki karmaşayı azaltmaktadır. Önerilen mimaride statüler yeniden organize edilmekte ve ardışık zaman aralıklarında aynı yapıların tekrar kullanılmasıyla kafes yapıları gerçekleştirilmektedir. Sonuçta, alan küçültülmekte ve güç tüketimi azaltılmaktadır. Adresleme karmaşası azaltıldığı için de hızın artması beklenmektedir.

İkinci alanı verimli Viterbi kod çözücü ise birinci yapının geliştirilmiş bir versiyonudur ve kod oranı, kısıt uzunluğu, değişim olasılığı, geriye iz sürüm derinliği ve üreteç polinomu gibi parametrelerin biçimlendirilmesine olanak sağlamaktadır.

Anahtar Kelimeler: Viterbi Algoritması, Biçimlendirilebilir Viterbi Kod Çözücüsü, SystemC, Evrişimsel Kodlayıcı, En Büyük Olabilirlik Yöntemi.

ACKNOWLEDGEMENTS

I would like to express my special thanks to Prof. Dr. Murat Aşkar for his guidance and supervision throughout this thesis work.

I am also grateful to my colleagues in ASELSAN for their encouragements towards the realization of this thesis work.

Finally, I would like to thank my family for their great support during this thesis work.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The communication deals with the transportation of information from one place to another. In a typical communication scheme, a group of symbols are generated by a source. Then, the source data passes through some encoding and modulation processes in the transmitter side to increase the noise immunity of source data and the modulated data is transmitted to the channel. But due to noise and interference in the channel corruption occurs in the received data. In the receiver side, after inverse processing of received data by demodulation and decoding stage, the original data is aimed to be reached.

To decrease the error in the data transmission several methods have been considered like Maximum Likelihood Decoding. In Maximum Likelihood Decoding technique, some correlation bits are added to the transmitted data so that any errors introduced in the communication channel can be corrected at the receiver end.

One of the most popular Maximum Likelihood error correction methods is the Viterbi Algorithm [1] [2]. The Viterbi Algorithm is used in many applications including speech recognition, digital sequence detection for magnetic storage devices and wireless communication. For the wireless communication, the Viterbi Algorithm generally decodes the convolutionally coded data to purify the original message from received data transferred in a noisy channel. Depending on the applications different implementation issues has been studied by several researches, including high speed Viterbi decoders [12] [22], reconfigurable constraint length Viterbi decoders [14] [15] [18] [21] [23] [25], several trellis structures [5], shared ACS units between separate states in the trellis [4], the block-based decoding approach [6], bit-serial approaches [8] [10], path metric computations [8], hardware

size reduction [4] [24] [25], radix4 based architectures [3] [19] and low power consumption [7] [9] [10].

The objective of this thesis is the VLSI implementation of an area efficient reconfigurable Viterbi decoder using SystemC tool. For the Viterbi Decoding usually two methods are used namely Register-Exchange and the trace-back. The Register-Exchange method is suitable for only a small number of states containing trellis', however, the trace-back approach is acceptable for trellis' with a large number of states. In the thesis, the configurable constraint length ranges from 4 to 7, so the maximum number of states is 64. For this reason, the trace-back method is preferred and used in the implementations.

The design environment SystemC [20] [11] [13] [17] is indeed a C++ class library, with added hardware modelling structures, that increases the power of C++ language to meet the needs of next generation designs containing analog signal computations and embedded software simulations like in System-on-Chip (SoC). The selection of SystemC as the development platform requires some preliminary studies to configure C++ compiler as SystemC compiler. The platform called OpenSystemC regulates to the SystemC specific syntaxes and debug techniques [16].

The Hard and Soft Decision Viterbi decoders have been implemented to divide the decoder into sub-blocks. Upon detailed study on the states in the trellis structure, two novel area efficient reconfigurable Viterbi decoder architecture are suggested. The major improvements in these area efficient structures are the new trellis structures which give ability to configure the constraint length by the usage of the same structure in subsequent time instances. Depending on the rearrangement of the states of the trellis structures, the switching and memory addressing complexity are reduced.

For reducing the switching and hardware complexity, in the first suggested reconfigurable Viterbi decoder architecture, vertically rotated complemented state rearrangement is used in subsequent iterations. As the result, the area is minimized and power consumption is decreased. Since the addressing complexity is reduced

the speed is expected to increase. In the suggested architectures, the trellis structures are realized by the usage of the same structures in subsequent instances to reduce the number of Add-Compare-Select units, Branch Metric units, DPRAMs and interconnections between these units. To decrease the number of memory locations for state metric storage, the in-place path metric updating technique has been used.

The second area efficient Viterbi decoder is an improved version of the first one and has the ability to configure the parameters of constraint length, code rate, transition probabilities, trace-back depth and generator polynomials. This improved method implemented in SystemC. Also, in the implementation some special efforts was carried out to optimize the operation. For the synchronization, the demodulator output is connected to the implemented architecture through an input FIFO stage. To increase the decoding speed to real time operation, the decoder is designed to include a LIFO memory stage between trellis and the traceback. The trellis is not implemented directly with the bare state identifiers. The state identifiers are also mapped to hidden state identifiers with a complement operation. So the addressing of the memories becomes consistent with the iteration counter. Then, the direct connection of the iteration counter to the memory address lines without any extra circuitry is possible.

In Chapter 2, the basics and the advantages of the SystemC beyond the other Hardware Description Languages are discussed. Then, step by step instructions to setup the SystemC design and development platform on Microsoft Visual Studio is described on an example.

The source of the message to be decoded in Viterbi decoder is created by convolutional coding. Therefore, the Chapter 3 starts with the explanation of the convolutional coding. Then the theoretical explanation with probabilistic analysis of the Viterbi algorithm is presented. Finally, in Chapter 3, the practical operational considerations of the Viterbi decoder is discussed and decoding of a sample data set is given in details.

In Chapter 4, to acquire the experience of the SystemC for the hardware modelling with the connections between sub-modules implementation of the simplest type of Viterbi decoders called hard decision Viterbi decoder is carried out. The modules in hard decision Viterbi decoder are modified to obtain implementation of soft decision Viterbi decoder.

Then, theoretical study was carried out about reconfigurable Viterbi decoding and a new area efficient approach of the reconfigurable Viterbi decoder is suggested in the first part of Chapter 5. In the second part of the Chapter 5, the second area efficient approach is explained and the SystemC implementation details are described for this further developed architecture. Then the simulation results of this decoder are given for several different parameters.

Finally, in Chapter 6, some concluding remarks and proposed future works are declared. At the end of Chapter 6, references are also presented for further reading and understanding.

CHAPTER 2

HARDWARE DESCRIPTION LANGUAGES AND SYSTEM C

2.1 General

Even a NOR gate implementation takes up more than a day to work on schematic, layout and simulation. With the necessity of producing higher density chips the engineer groups turned towards the search of new methodologies to implement the designs connecting millions of gates with the timing and die area constraints. So, some decades ago the decision were made to change the design topology from schematic based to Hardware Description Language (HDL) based design. The HDLs met the expectations of digital circuit implementations for a long time, but with System On Chip (SoC) concept once again the hardware implementation suffers from complexity. The new design challenges on System-on-Chip (SoC) grows up in the eras of analog signals processing, embedded software usage representing over the half of the functionality and reduction in time-to-market.

On the other hand, the traditional design methodology in project life cycle commences with the model creations in a software platform (generally in C or C++ language) to verify the algorithms at the system level, then, continues on the division of the implementation into sub-blocks of hardware and embedded software. After the confirmation of operation, parts to be implemented as hardware are manually converted to HDLs, like VHDL or Verilog, performing the same functionality as in the C/C++ model. But the model created in C/C++ was verified in a software platform neglecting the timing issues. So, the manual conversion approach is a time consuming and error-prone job including the statement conversion to the hardware obeying a master clock and also again new test suite is needed to be setup in the HDL environment. Thus, apart from HDL, another implementation and simulation tool is required to cope with the complexity of SoC and the processes in the project life cycle.

Because of the commonly acceptance of C++ language at many abstraction levels in the industry, this object oriented programming language was selected to be the baseline for the building of the new implementation and simulation tool called SystemC. SystemC is a C++ class library with added hardware modelling structure that increases the power of C++ language to meet the needs of next generation hardware design.

With the SystemC approach, instead of the design conversion from a C level description to a HDL in one large effort, the design can be slowly refined in small sections to add the necessary hardware and timing structures to produce a good design. Within this refinement methodology, the designer can easily modify the design for further changes and detect bugs during refinement. Using this approach, the designer does not need to be an expert in multiple languages. Because SystemC allows modelling from the system level to RTL. The SystemC approach provides higher productivity because the designer can model at a higher level. Writing at a higher level can result in smaller code, that is easier to write and faster to simulate than traditional modelling environments. Also test benches can be reused from the system level model to the RTL model saving conversion time. Using the same test bench also gives the designer a higher confidence that the system level and the RTL level implementations have the same functionality.

Because of the C++ features, the SystemC is also a naturally object oriented tool that uses powerful data types based on the class of C++ language. These major data types of the SystemC version 2.0 can be defined as below.

- Modules: are the hardware entities used to perform operations (process) and connections of the sub-modules.
- Processes: performs calculations and decisions of the hardware.
- Ports: are the connection points of modules and can be used either uni-directional or bi-directional.
- Signals: are special data types to direct the information inside and in between modules.

- Clocks: are the timekeepers for the modules to synchronise the state machines. In SystemC, the multi clock operations with arbitrary phase shifts are available.

On the other hand, designer can use standard C data types to model analog signals and control loops like small functions in their test benches.

Because of the advantages of SystemC, the firms listed below are in function on the integration of SystemC in several abstraction levels.

Alcatel, Altera, Aptix, Arcadia Design Systems, ARC Cores, ARM, Billions of Operations Per Second, Chameleon Systems, Inc., Co-Design Automation, CoWare, CSELT, Cygnus Solutions, Denali, Ericsson, Frequency Technology, Frontier Design, Fujitsu Microelectronics, IKOS Systems, I-Logix, Infineon Technologies, Integrated Silicon Systems, Intellectual Property Inc., Internet CAD, LogicVision, Lucent Technologies, Magma Design Automation, MIPS Technologies, Monterey Design Systems, Motorola, Inc., Seva Technologies, Sican Microelectronics Corp., Snaketech, Sony Corporation, STMicroelectronics, Sun Microsystems, Synchronicity, Synopsys, Tensilica, Texas Instruments, TransModeling, Ultima, Verplex, and Xilinx.

The SystemC language is free and can be downloaded from the Open SystemC Initiative web site. Currently, the release of SystemC V2.0.1 is supported on the following platforms:

- Sun Solaris 2.7 and 2.8 with GNU C++ compiler versions gcc-2.95.2 and gcc-2.95.3
- Sun Solaris 2.7 and 2.8 with SUN C++ compiler versions SC6.1 and SC6.2
- Linux (Redhat 6.2) with GNU C++ compiler versions gcc-2.95.2 and gcc-2.95.3
- Linux (Redhat 7.2) with GNU C++ compiler version gcc-2.95.3
- HP-UX 11.00 with HP C++ compiler versions A.03.15 and A.03.33

- Windows NT 4.0 (SP6a) with VC++ 6.0 (SP5)
- Borland C++ Builder V5.0 (SP1)

For further information references [1], [2], [3], [4], [5], [6] can be applied.

2.2 Requirements to Establish SystemC Compiler In This Thesis

The development process of this thesis was performed on the softwares listed in Table 2-1.

Table 2-1 Requirements

Software	**Definition**	**For more information**
SystemC-2.0.1	SystemC library	www.SystemC.org
Microsoft Visual Studio	C++ Compiler	Microsoft
Winbeta	Simulation Software	SystemC_win@yahoo.com
Windows XP	Operating System	Microsoft

2.3 Example Usage of SystemC

In this section to configure Microsoft Visual Studio 6.0 as SystemC compiler and to implement an example hardware will be described with the main steps. To perform error free design the designer can perform the instructions listed below.

1. Download SystemC-2.0.1 files from “www.SystemC.org” and unzip the contents into the folder “D:\SystemC\SystemC-2.0.1\”.
2. Execute Microsoft Visual C++ 6.0. Fig. 2-1 shows the main window of Visual Studio.

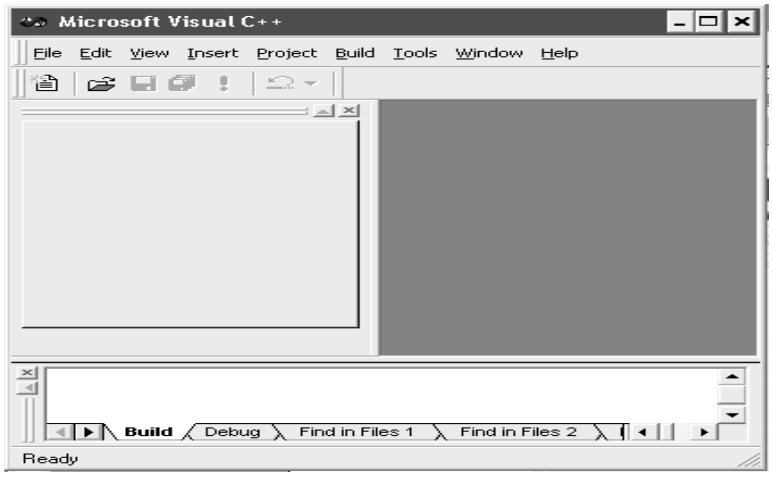


Fig. 2-1 Microsoft Visual C++ 6.0 Main Window

3. To create a new project click on "File->New" button. Then the "New" window will be opened. On the "Projects" tap of the "New" window, select "Win32 Console Application", fill the project name and the location to where the project will be created.(**Fig. 2-2**) Then, press "OK" button.

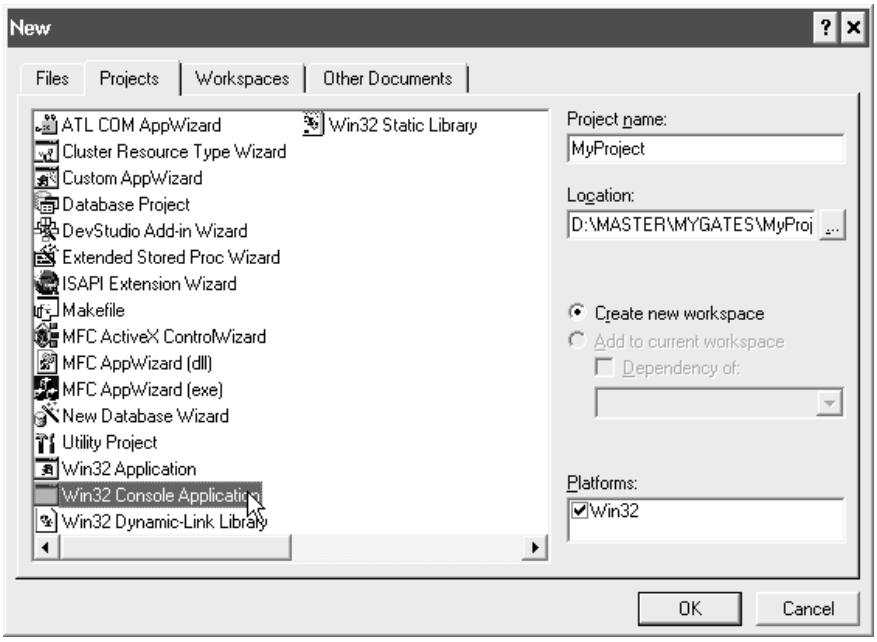


Fig. 2-2 "New" Window

4. A new window called “Win32 Console Application – Step 1 of 1” will appear. In this new window select “An empty project” option and click on “Finish” button. At this stage, the new application would be created as shown in **Fig. 2-3**.

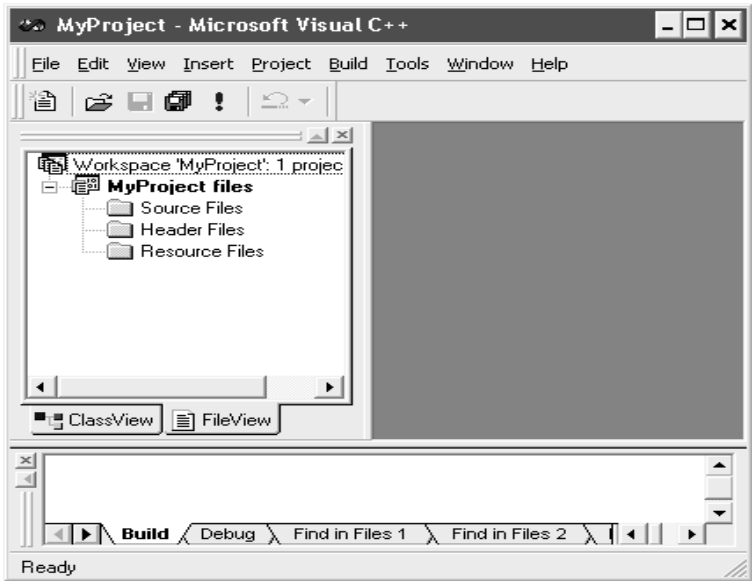


Fig. 2-3 New Project

5. SystemC library should be embedded into the project. For this reason right click on the “Resource Files” and select “Add Files to Folder...” button on pop-up menu. Select “SystemC.lib” file located in “D:\SystemC\SystemC-2.0.1\msvc60\SystemC\Debug” folder. Then, into the “Header Files” section import “SystemC.h” file located in the same folder.

6. In the main window click “Project->Settings...” button. On the “C/C++” Tap select “C++ Language” in the category drop-down list. And, check the “Enable Run-Time Type Information(RTTI)” check box.(**Fig. 2-4**)

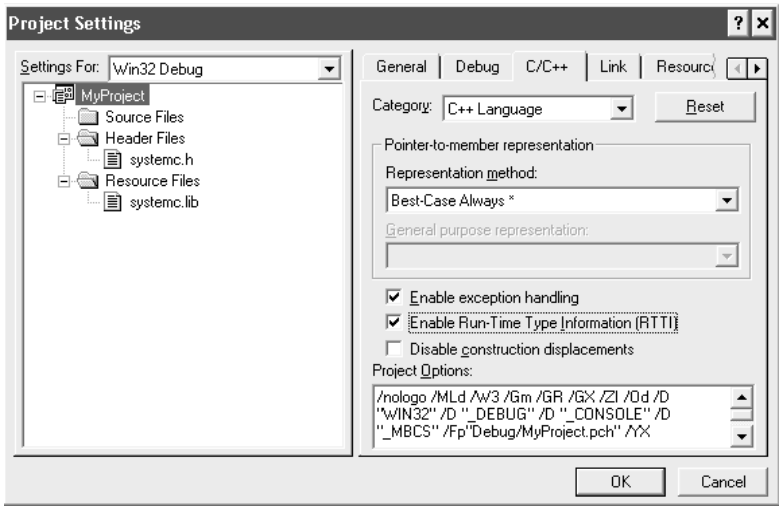


Fig. 2-4 Project Settings "C++ Language" Category

7. Select "Processor" in the category drop-down list and fill the "Additional include directories" section with "D:\SystemC\SystemC-2.0.1\src\".(**Fig. 2-5**)

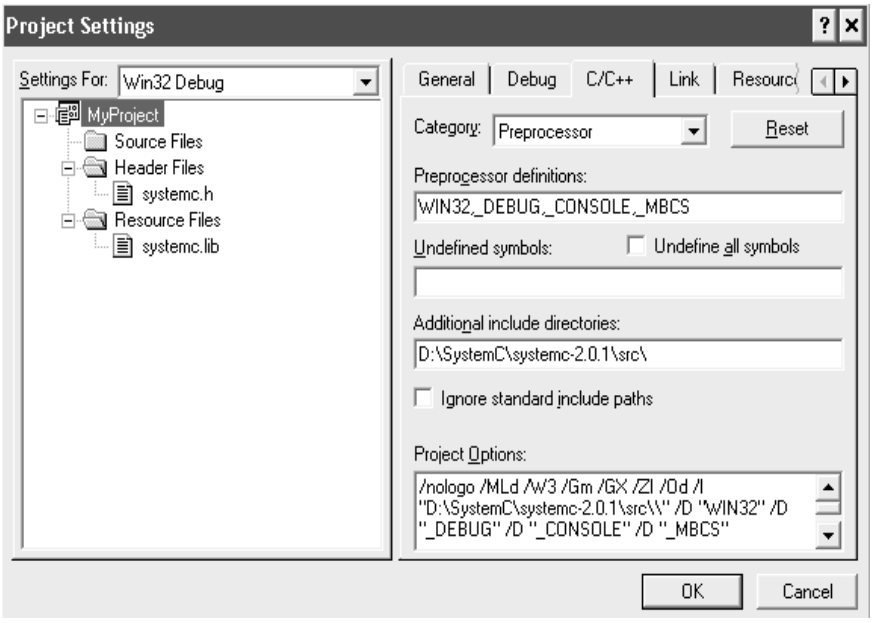


Fig. 2-5 Project Settings "Processor" Category

8. Up to this point, the compiler has been configured for SystemC simulations. From now on, an example project will be compiled. For this reason a source

code for a hardware called “ACS” was written in “ACS.h” file and also two extra files were written for monitoring the ports (mon.h) and creating the stimulation pattern (stim.h). These header files should be imported into the “Header Files” section.

9. Now “main.cpp” file is added into the “Source Files” section in order to make hardware connections between source, simulation and monitor files.

10. Finally, the project will be compiled by clicking “Build->Rebuild All” button and will be executed by clicking “Build->Execute” button. This will create a command prompt to show simulation results determined in monitor file. (Fig. 2-6)

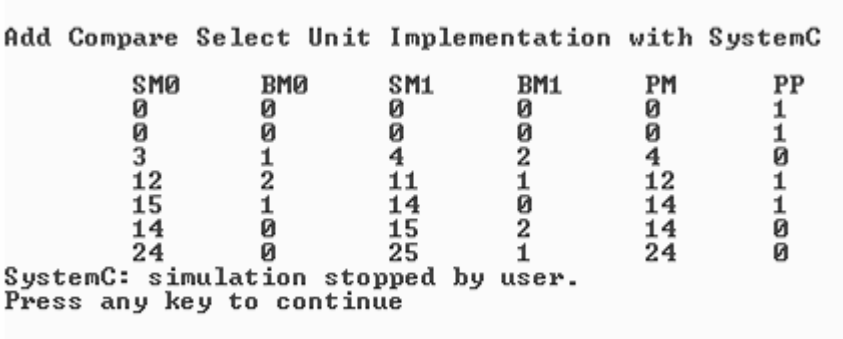


Fig. 2-6 Command Prompt Simulation Result

11. Depending on the statements in “main.cpp” a SystemC vcd trace file “*.vcd” can be created. In “Winbeta” program the “*.vcd” file can be visualized. In main window of Winbeta (**Fig. 2-7**) click on “File->Open Vcd Files” button and select the “*.vcd” file in the open dialog box.

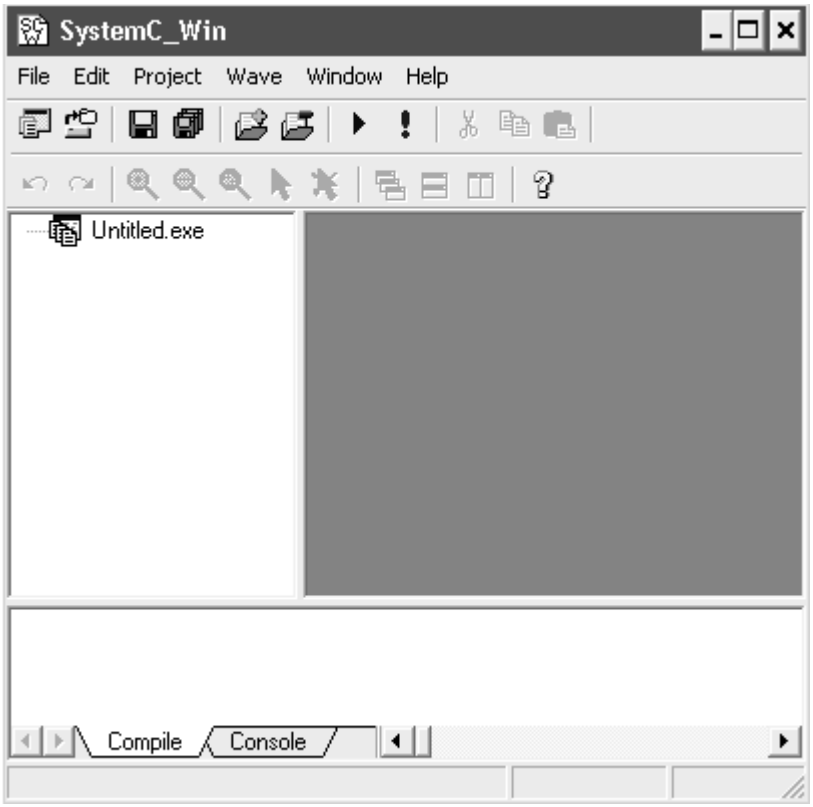


Fig. 2-7 Winbeta main window

12. The simulation will be seen as in **Fig. 2-8**

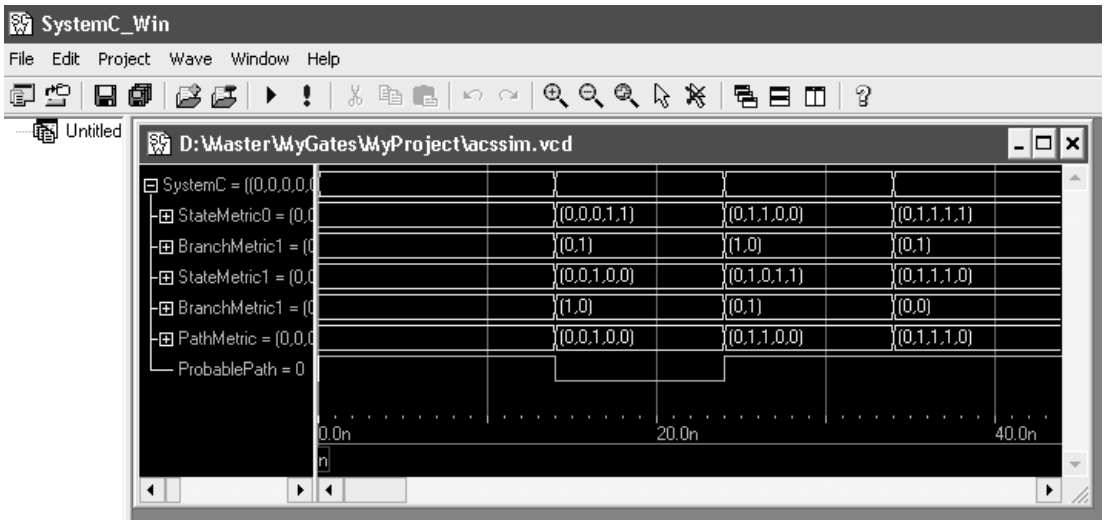


Fig. 2-8 Vcd file

CHAPTER 3

CONVOLUTIONAL CODER AND VITERBI DECODER

3.1 General

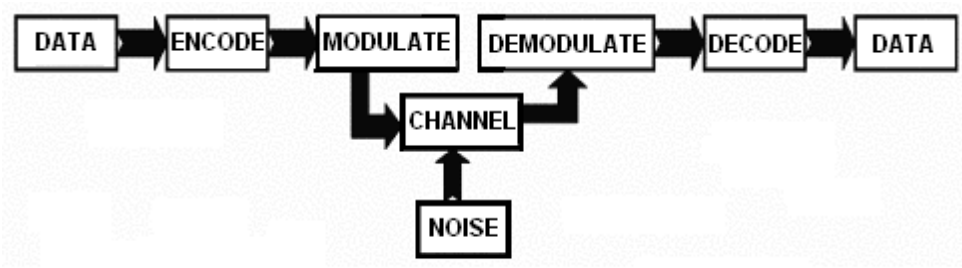


Fig. 3-1 Typical Communication Scheme

The error correction methods have been studied extensively in digital communication researches to overcome the data transmission error problem. One of the well known error correction method is Viterbi Algorithm and generally used in decoding the convolutionally coded data to purify the original message from received data in a noisy channel. The Viterbi Algorithm is a maximum likelihood decoding method to estimate the message embedded in noisy data by using the maximum a-posterior probability.

The Viterbi algorithm is used in many applications including speech recognition, digital sequence detection for magnetic storage devices and wireless communication. Depending on the applications different implementation issues has been studied by several researches, including high speed Viterbi decoders,

reconfigurable constraint length Viterbi decoders, path metric computations, hardware size reduction and low power consumption.

As indicated, the Viterbi decoder is used mainly to decode the convolutional encoded data. So in the first part of this chapter, an overview of convolutional encoding is stated. Then theoretical meaning and basic terminologies of the Viterbi decoding is described. In the subsequent part of this chapter, basic computations including trellis, butterfly and branch metrics are provided within the Viterbi Decoding example.

3.2 Convolutional Encoder

A convolutional encoder accepts an input stream of message and generates encoded output streams to be transmitted. In this process for one input bit the encoder generates more than one output bits and these redundant symbols in output bit pattern makes the transmitted data more immune to the noise in the channel. As will be seen later in Viterbi Decoding section, the redundant bits help to decide and correct the errors in received pattern.

For the standardization among the engineers in convolutional encoding and Viterbi decoding concept a terminology was generated as summarized below:

M : Length of the shift register stage in the encoder

Constraint Length (K) = M+1 : This number represents the number of input bits required to generate a unique output pattern in the encoder. A constraint length of K=7 means that each output symbol depends on the current input symbol and the six previous input symbols.

Number of States = $2^{(K-1)}$: Defines the maximum number of states that is possible to be mapped by the combinations of the K number of input bits for the convolutional encoder.

L : Length of Input Message

R : Convolutional Code Rate

$$R = \frac{\text{Number of input bits to create a symbol at the output}}{\text{Number of output bits in a symbol at the output}} = \frac{m}{n}$$

For example, 1/2 code rate means each bit entering the encoder results in 2 bits leaving the encoder.

Generator polynomial: A generator polynomial specifies the encoder connections. In another words, the generator polynomial can be deduced as the mathematical description of the convolutional encoder. Each polynomial forming the generator polynomial should be at most K degree and specifies the connections between the shift registers and the modulo-2 adders. In the generator polynomial representation the variable D corresponds to clock delay

$$G^{(i)}(D) = G0^{(i)} + G1^{(i)}D + G2^{(i)}D^2 + \ldots + GM^{(i)}D^{(K-1)}$$

$$\begin{bmatrix} \text{Output} \\ \text{symbol} \\ \text{matrix} \end{bmatrix}_{1xn} = \text{Mod2}\left(\begin{bmatrix} \text{Input bit} \\ \text{pattern of} \\ \text{length K} \end{bmatrix}_{1xK} \text{x} \begin{bmatrix} \text{Generator} \\ \text{polynomial} \\ \text{matrix} \end{bmatrix}_{Kxn} \right)$$

$$\text{Generator polynomial matrix} = \begin{bmatrix} GM^{(i)} & . & . & G0^0 \\ GM^{(i)} & . & . & G1^0 \\ . & & & . \\ . & & & . \\ GM^{(i)} & . & . & GM^0 \end{bmatrix}_{3x2}$$

For input pattern of 1, 0, 1 the generic convolutional coder creates 2 bits of symbol at a time.

$$\text{mod } 2\left(\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}_{1x3} \bullet \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}_{3x2} \right) = \begin{bmatrix} 0 & 0 \end{bmatrix}_{1x2}$$

As declared in terminology part the constraint length is linearly related to the M in the coder side. On the other hand, the number of states are the main parameter to decide the computational complexity of Viterbi decoder and increases exponentially as constraint length increases. However, the immunity of the encoded data increases with the increase of the constraint length. So, for an adequate noise immunity and the cheapest solution, an optimum constraint length should be selected.

3.2.1 Convolutional Encoder Examples

To clarify the terminology some further examples are required for convolutional coders. So this section is dedicated to the convolutional coder examples.

In the convolutional coders below, ports labelled with "I" corresponds to input and other ports labelled with "Q" corresponds to output ports. The convolutional encoders consist of M stages shift registers and one or more modulo-2 adders, represented with the shape $\oplus$. Also for each encoder, the related generator polynomials are given at the bottom of the specified figures.

➢ **Convolutional Coder K=2 r=1/2**

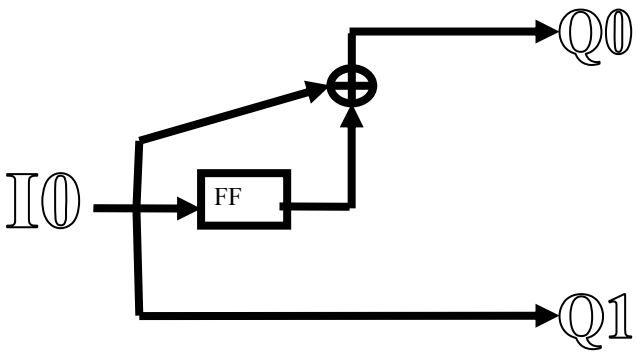


Fig. 3-2 Convolutional Coder K=2 r=1/2

Q0➔ $G^{(0)}(D)= 1 + D$

Q1➔ $G^{(1)}(D)= 1$

➢**Convolutional Coder K=3 r=1/2**

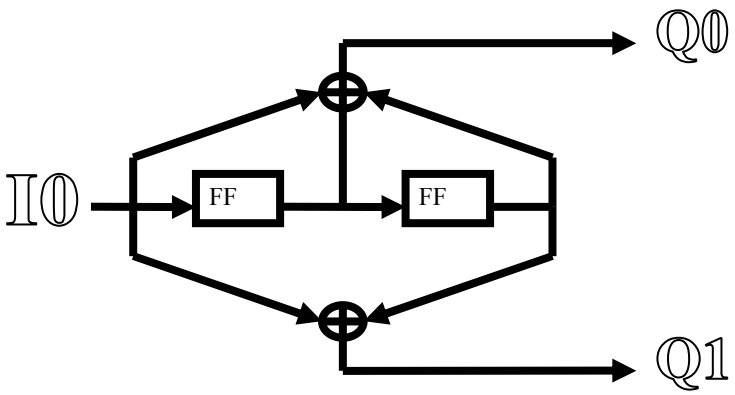


Fig. 3-3 Convolutional Coder K=3 r=1/2

Q0➔ $G^{(0)}(D)= 1 + D + D^2$

Q1➔ $G^{(1)}(D)= 1 + D^2$

➢**Convolutional Coder K=5 r=1/2**

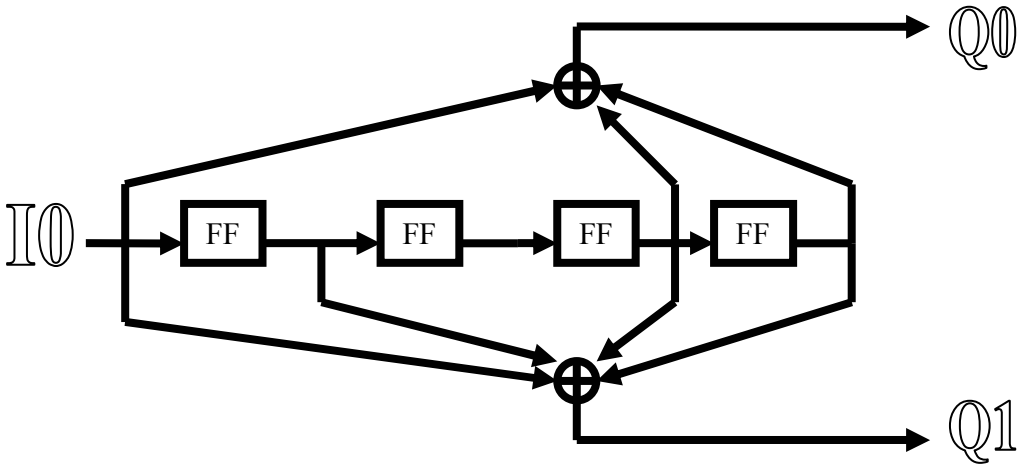


Fig. 3-4 Convolutional Coder K=5 r=1/2

Q0➔ $G^{(0)}(D)= 1 + D^3 + D^4$

Q1➔ $G^{(1)}(D)= 1 + D + D^3 + D^4$

➢**Convolutional Coder K=3 r=2/3**

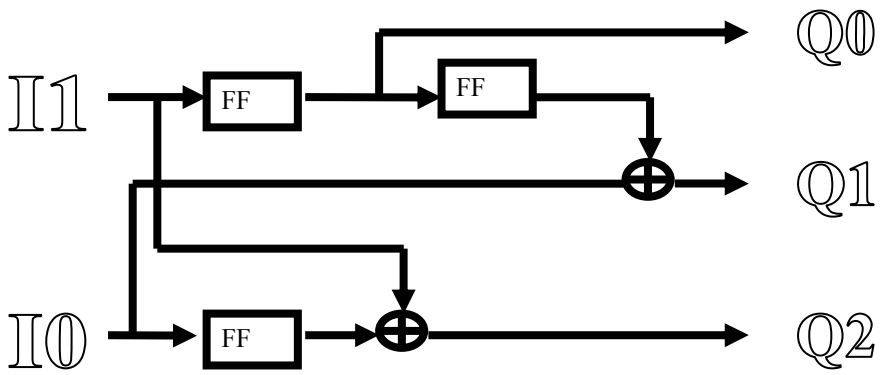


Fig. 3-5 Convolutional Coder K=3 r=2/3

3.2.2 Operation of the K=3 r=1/2 Convolutional Encoder

In this section the operation of the generic convolutional encoder of K=3 and R=1/2 (Fig. 3-7) will be stated. In the figures "I0" is the input port of encoder from where the original message stream is applied. "Q0" and "Q1" are the output ports where encoded message comes out. The two boxes in the middle are serial shift registers and circles corresponds to modulo-2 adders. In the hardware, the modulo-2 adder is implemented with the exclusive-or gate whose truth table is the same as the modulo-2 adder, as illustrated below.

A	B	C	O
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

Truth Table of Modulo-2 Adder

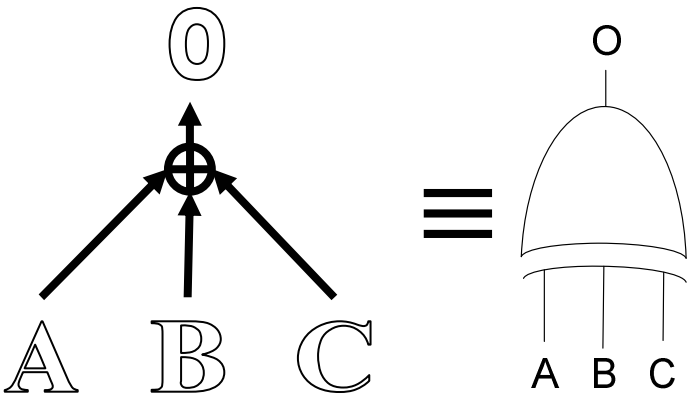


Fig. 3-6 Modulo Adder

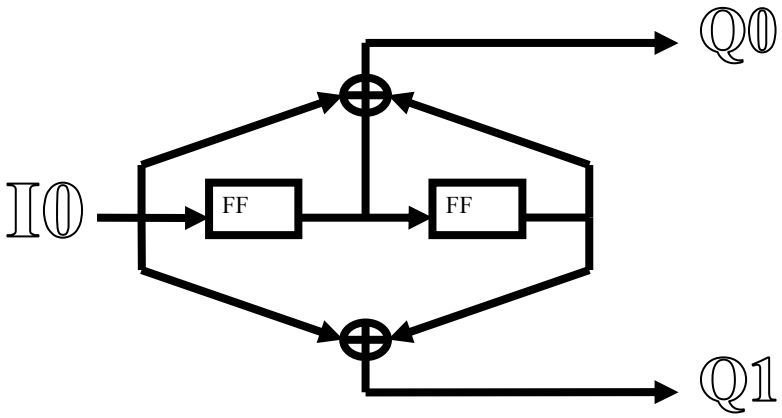


Fig. 3-7 Generic Convolutional Coder K=3 r=1/2

At the initial point the shift registers in the convolutional encoder are at reset position which is all-zero content. As illustrated in Fig. 3-7, whenever a data bit enters from the input port, encoder produces two encoded bits and both of the encoded bits are correlated with instant and K-1 number of previous input bits, where K refers to the constraint length of the convolutional encoder. In the encoding process of K=3 and r=1/2 coder the input symbol enters the shift registers stage one bit at a time on the left terminals. After the modulo-2 addition of the selected shift register stages and input data, the two output bits are generated out of the encoder.

Now “01110010” input stream will be applied into the encoder.

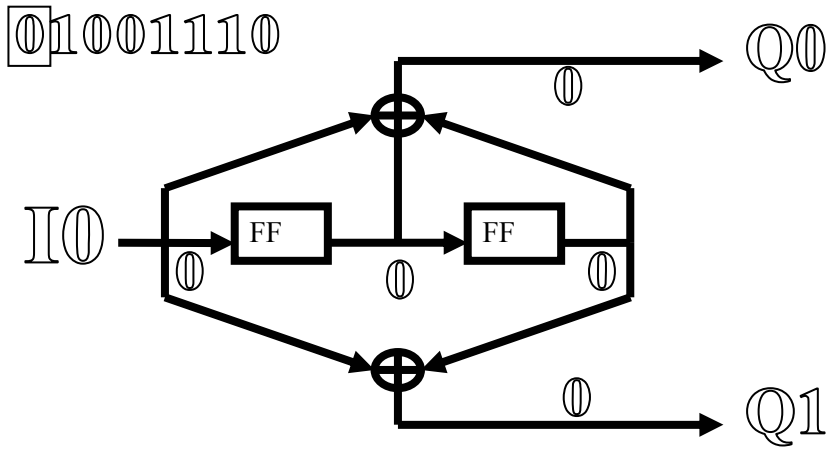


Fig. 3-8 Convolutional Encoding Example with 1st bit

At the beginning, the convolutional encoder is in initial state (“00” state). So the flip flops are loaded with logic “0”. When the first bit of input message is applied which is logic “0” the output will be “00” (“Q1Q0”) and the next state will be again “00”.

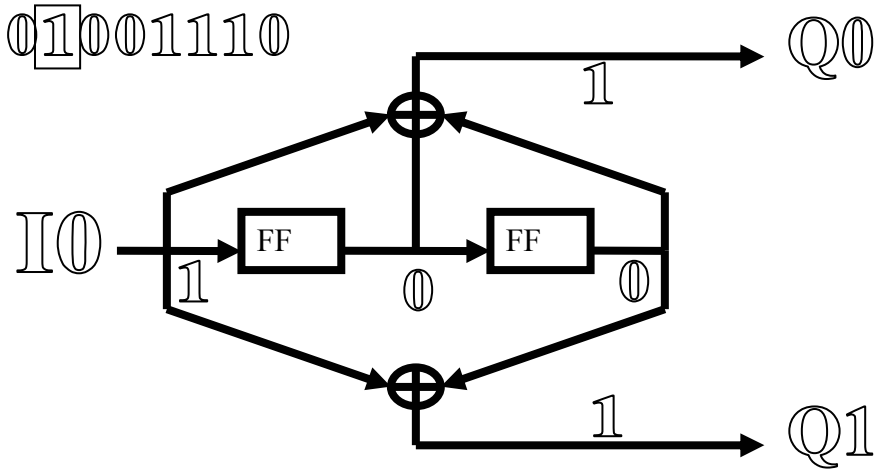


Fig. 3-9 Convolutional Encoding Example with 2nd bit

Now the second input bit will go inside the encoder which is logic 1,then successive output will be “11” and the next state will be “10”.

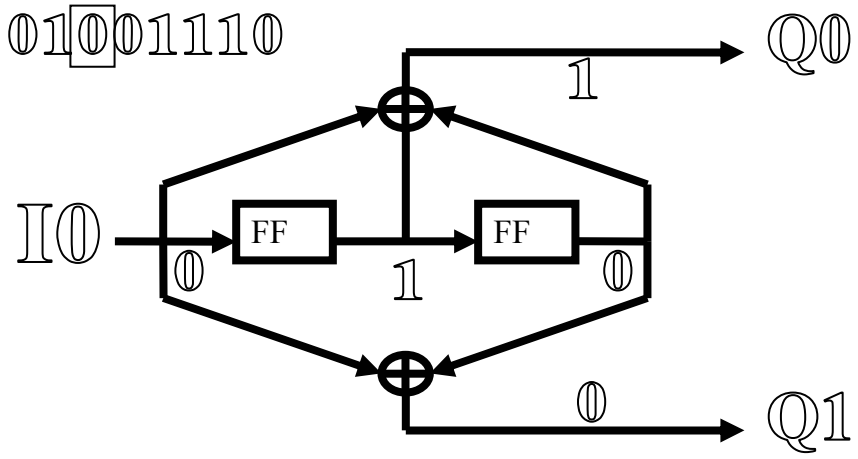


Fig. 3-10 Convolutional Encoding Example with 3rd bit

Then the third input bit will go inside the encoder which is logic 0. Present state is “10”. Output will be “11” and the next state will be “01”. At the fourth input Input=”0” Present State=”01” Output=”11” Next State=”00”

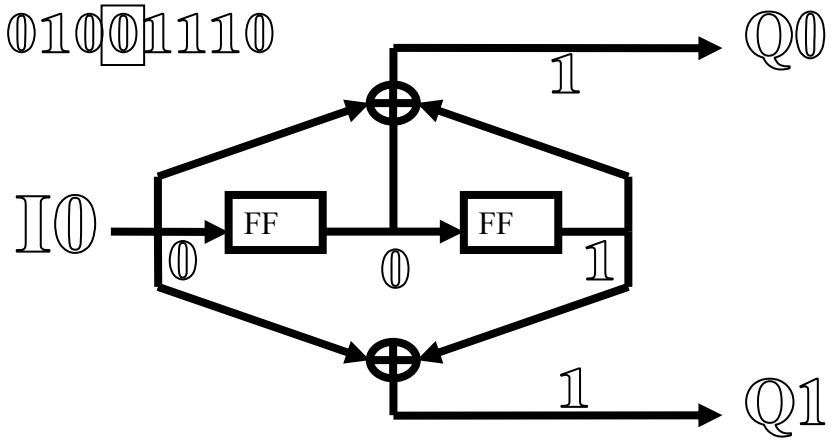
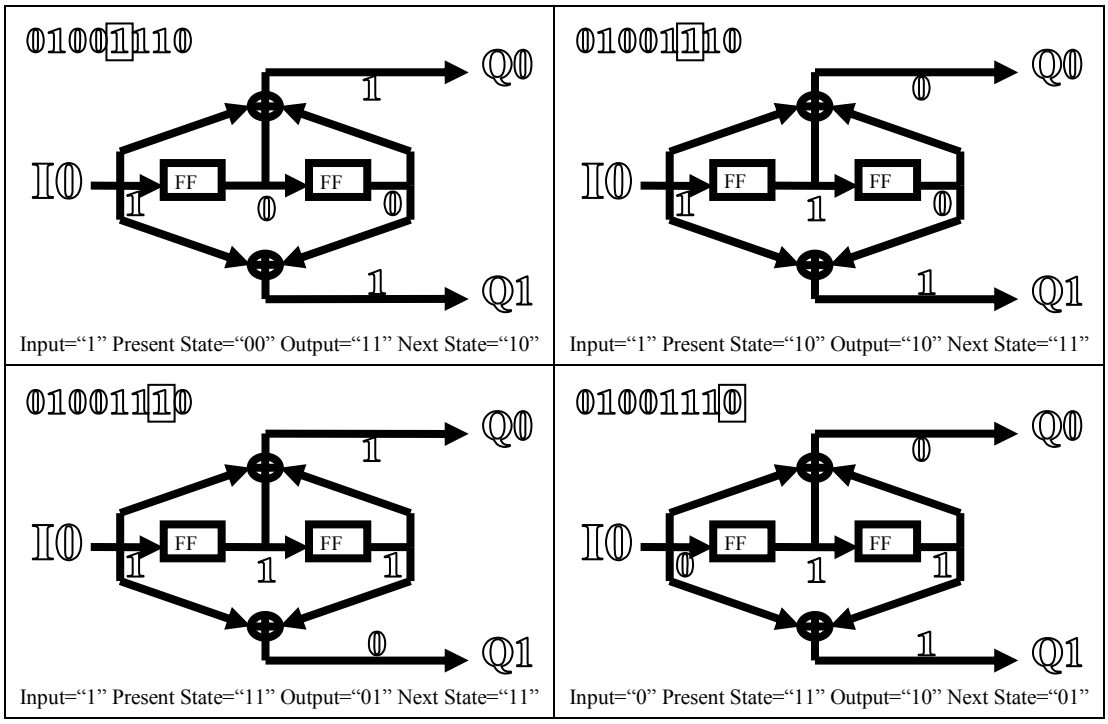


Fig. 3-11 Convolutional Encoding Example with 4th bit



Output Pattern: 0011101111011001

Fig. 3-12 Convolutional Encoding Example Continued

From the example the conclusions below are extracted:

- State information is directly related to the content of serial registers. Present content of registers forms present state information and next content of registers after the current input insertion forms the next state information.
- The output of the encoder is not only dependent to input but also dependent to the present state of the encoder, so the register values of the encoder.

3.2.3 Representations of Convolutional Encoders

In this section major representations for the state transitions of the convolutional encoder will be listed.

➢**State Table**

State table is the easiest way to determine the state information and output relation of the encoder as shown below.

Table 3-1 State Table Representation of Convolutional Encoder

INPUT BIT	PRESENT STATE	NEXT STATE	OUTPUT CODEWORD
0	00	00	00
1	00	10	11
0	01	00	11
1	01	10	00
0	10	01	10
1	10	11	01
0	11	01	01
1	11	11	10

➢**State Diagram**

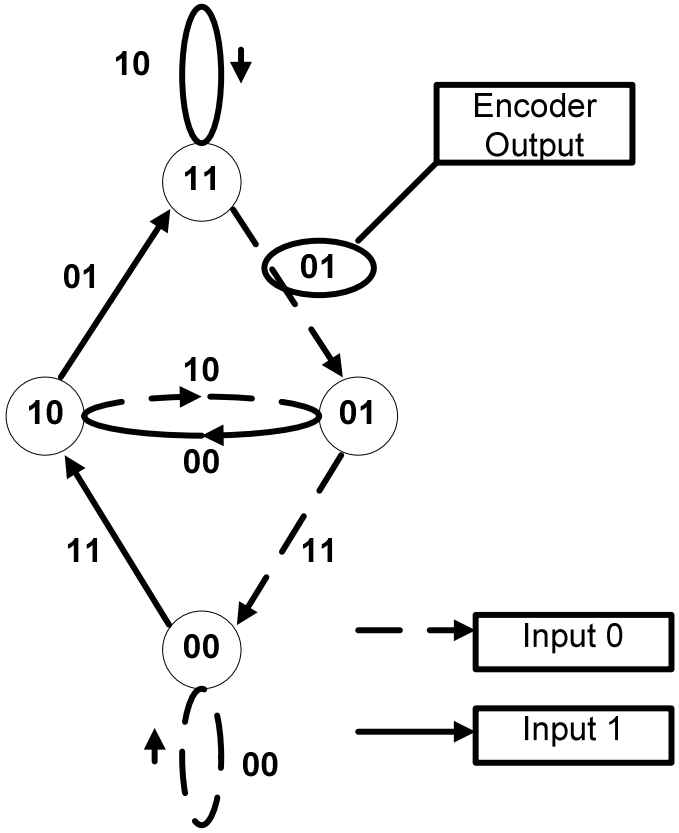


Fig. 3-13 State Diagram Representation

State diagram is the graphical way to show state table. In the state diagram bubbles are the states and the indicators of the states are written inside the bubbles. Arrows are the state transitions according to input values. The values on the arrows corresponds to the output of the encoder while the transition takes place with respect to the input value.

➢**Code Tree K=3 r=1/2**

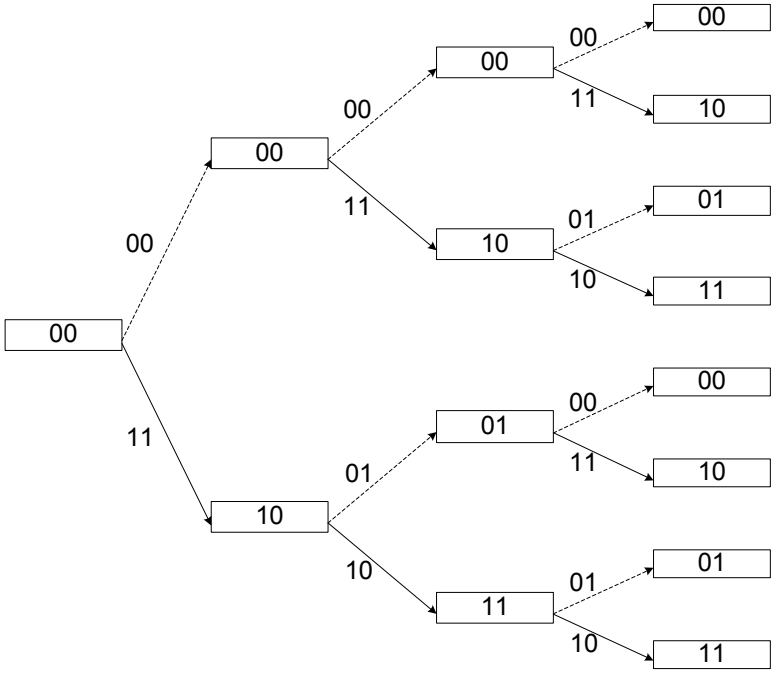


Fig. 3-14 Code Tree Representation 1

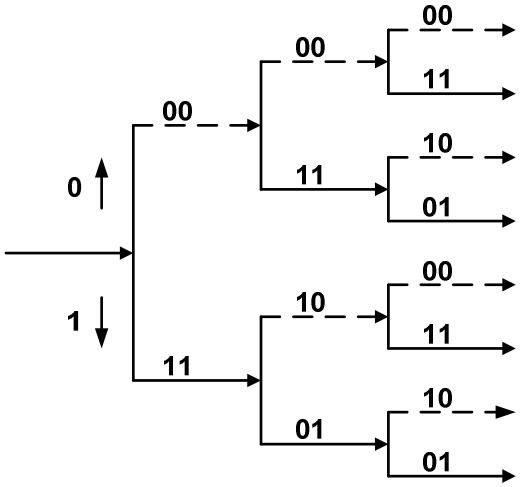


Fig. 3-15 Code Tree Representation 2

In the code tree representation small boxes are called state boxes (Fig. 3-14). And arrows represents the state transitions The values on the arrows are encoder output values. In code tree representations sometimes state boxes are not shown as Fig. 3-15.

3.3 Viterbi Decoder

3.3.1 Viterbi Algorithm

As indicated in the introduction, the Viterbi Algorithm (VA) is a Maximum Likelihood Decision Algorithm. The Algorithm tries to find the message stream m_i whose likelihood function is larger or equal to the likelihood functions of other probable messages m_j, due to the observed data.

$$P(m_i \text{ sent}\backslash z) \geq P(m_j \text{ sent}\backslash z) \text{ for all } j \neq i \qquad \text{(Eq. 1)}$$

The convention to represent the observed stream is $z=\{z_0, z_1, \ldots, z_T\}$ where subscript numbers indicate the sampling time.

$P(m_i \text{ sent}\backslash z)$ is a likelihood function and indicates the probability of message m_i sent given that the observed stream is z.

In the decoding process, VA searches for the best path and so the messages corresponding to states, that are the building blocks of the path. The best path passes through the state x^i_k which is the state at sampling instant k in the state sequence i. The transition in the state sequence i between states x^i_k and x^i_{k+1} is shown as ξ^i_k. According to the state sequence $x^i=\{x^i_0, x^i_1, \ldots, x^i_T\}$ the Maximum Likelihood Equation can be written as;

$$P(x^i \backslash z) \geq P(x^j \backslash z) \text{ For all } j \neq i \qquad \text{(Eq. 2)}$$

The new aim is to maximize the $P(x \backslash z)$. As known, the unconditional probability of P(z) is positive value and is independent of the transmitted function so the state sequence. Thus, scaling the both sides of the Maximum Likelihood Equation in Eq. 2 has no effect on the result of inequality.

$$P(x^i \setminus z)P(z) \geq P(x^j \setminus z)P(z) \text{ For all } j \neq i \qquad \text{(Eq. 3)}$$

The joint probability $P(x^i, z)$, that is the probability of the simultaneous occurrence of the state sequence x^i and the observation sequence z, is equal to;

$$P(x^i, z) = P(x^i \setminus z)P(z) \qquad \text{(Eq. 4)}$$

The Eq. 3 can be rewritten as

$$P(x^i, z) \geq P(x^j, z) \text{ For all } j \neq i \qquad \text{(Eq. 5)}$$

The joint probability $P(x^i, z)$ is also equals to;

$$P(x^i, z) = P(x^i)P(z \setminus x^i) \qquad \text{(Eq. 6)}$$

In a Markov process the state transition to x_{k+1} only depends on the previous state x_k.

$$P(x^i{}_{k+1} \setminus x^i{}_0, x^i{}_1, \ldots, x^i{}_k) = P(x^i{}_{k+1} \setminus x^i{}_k) \qquad \text{(Eq. 7)}$$

Both the state x_k and the state transition $\xi_k = (x^i{}_{k+1}, x^i{}_k)$ represent the same path in different ways. Thus,

$$P(x^i) = P(\xi^i) = \prod_{k=0}^{K-1} P(x^i{}_{k+1} \setminus x^i{}_k) \qquad \text{(Eq. 8)}$$

Thus the conditional probability in Eq. 6 can be rewritten as

$$P(z \setminus x^i) = P(z \setminus \xi^i) = \prod_{k=0}^{K-1} P(z_k \setminus \xi^i{}_k) \qquad \text{(Eq. 9)}$$

The reimplementation of Eq. 6 using Eq. 7, Eq. 8 and Eq. 9

$$P(x^i, z) = \prod_{k=0}^{K-1} P(x^i{}_{k+1} \setminus x^i{}_k) \prod_{k=0}^{K-1} P(z_k \setminus x^i{}_{k+1}, x^i{}_k) \qquad \text{(Eq. 10)}$$

With the help of natural logarithm the time consuming and complicated multiplication calculations can be converted to summation.

$$-\ln\left(P\left(x^i,z\right)\right)=-\ln\left(\prod_{k=0}^{K-1}P\left(x^i{}_{k+1}\setminus x^i{}_k\right)\right)-\ln\left(\prod_{k=0}^{K-1}P\left(z_k\setminus x^i{}_{k+1},x^i{}_k\right)\right) \quad \text{(Eq. 11)}$$

$$-\ln\left(P\left(x^i,z\right)\right)=-\sum_{k=0}^{K-1}\ln\left(P\left(x^i{}_{k+1}\setminus x^i{}_k\right)\right)-\sum_{k=0}^{K-1}\ln\left(P\left(z_k\setminus x^i{}_{k+1},x^i{}_k\right)\right) \quad \text{(Eq. 12)}$$

$$-\ln\left(P\left(x^i,z\right)\right)=\sum_{k=0}^{K-1}\left(-\ln\left(P\left(x^i{}_{k+1}\setminus x^i{}_k\right)\right)-\ln\left(P\left(z_k\setminus x^i{}_{k+1},x^i{}_k\right)\right)\right) \quad \text{(Eq. 13)}$$

Each element of summation in the right side of the Eq. 13 is called branch metric $\lambda(\xi)$ and calculated as;

$$\lambda(\xi^i{}_k)=\lambda_{k+1}^{(x^i{}_k,x^i{}_{k+1})}=-\ln\left(P\left(x^i{}_{k+1}\setminus x^i{}_k\right)\right)-\ln\left(P\left(z_k\setminus x^i{}_{k+1},x^i{}_k\right)\right) \quad \text{(Eq. 14)}$$

$\lambda(\xi^i{}_k)$: The branch metric for the state transition of sequence i from time instant k to instant k+1 ($x^i{}_k,x^i{}_{k+1}$).

Then, the equation Eq. 13 is given in another form.

$$-\ln\left(P\left(x^i,z\right)\right)=\sum_{k=0}^{K-1}\lambda(\xi^i{}_k) \quad \text{(Eq. 14)}$$

Finally VA aims to find the state transition path for which cumulative branch metric calculation is minimum.

$$\sum_{k=0}^{K-1}\lambda(\xi^i{}_k)\le\sum_{k=0}^{K-1}\lambda(\xi^j{}_k) \quad \text{For all } j\neq i \quad \text{(Eq. 15)}$$

The variable called Path Metric which is defined as $\Gamma^{x^i{}_k}$.

$$\Gamma^{x^i{}_k}=\sum_{l=0}^{k-1}\lambda(\xi^i{}_l)=\sum_{l=0}^{k-1}\lambda^{\left(x^i{}_l,x^i{}_{l+1}\right)} \quad \text{(Eq. 16)}$$

Finally, the overall Maximum likelihood decision turns to

$$\Gamma^{x^i}{}_k \leq \Gamma^{x^j}{}_k$$

For all j≠i (Eq. 17)

This is the bare structure leading to the Viterbi decoding through code tree representation which is described in details under the "Decision Through Code Tree K=5" heading.

From now on, the recursive behaviour of the VA is stated to decrease the implementation difficulty. The recursive behaviour of the algorithm creates a new structure called trellis, which is described in details under the "Decision Through Trellis" heading.

For the initialization of VA the initial state x_0 of the decoder is needed to be known for time k=0 and path metric of this state is $\Gamma^{x_0} = 0$.

Then for all states of the decoder the metrics related to the two competing transitions from states x_{k-1} and x'_{k-1} leading to the same next state x_k are calculated by the equations Eq. 18 and Eq. 19.

$$\gamma(x_k, x_{k-1}) = \Gamma^{x_{k-1}} + \lambda^{(x_k, x_{k-1})} \quad \text{(Eq. 18)}$$

$$\gamma(x_k, x'_{k-1}) = \Gamma^{x_{k-1}} + \lambda^{(x_k, x'_{k-1})} \quad \text{(Eq. 19)}$$

Then the minimum of these two calculations is stored as the path metric of next state x_k.

$$\Gamma^{x_k} = \min\left(\gamma(x_k, x_{k-1}), \gamma(x_k, x'_{k-1})\right) \quad \text{(Eq. 20)}$$

The graphical representation of the recursive calculation for one next state is shown in

Fig. 3-16

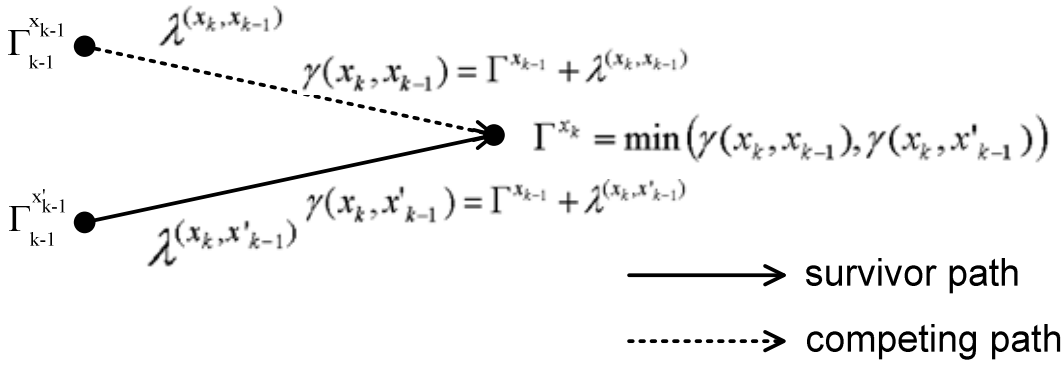


Fig. 3-16 Graphical Representation of The Recursive Calculation

Finally, after traceback depth length of input stream the VA compares the path metrics of the final states and selects the path ending with minimum state metric as the most probable path.

3.3.2 Viterbi Decoding

Viterbi decoder is the implementation of the Viterbi Algorithm, which is a maximum likelihood method to find the most probable input pattern coming from transmitter and through channel. In the operation, Viterbi decoder, independent of the input data, calculates every probable state transition from present time instant to the next, and relevant expected encoder output related to the state transition. Due to the distance between the expected encoder outputs and the received pattern, the Viterbi decoder also computes the branch transition probabilities for every state transition. Then, from the summation of the present state metric and the branch metric the next state path metric is calculated for every transition. After a specific number of observed data the decoder decides the most probable path and starts traceback from this state.

- **Decision Through Code Tree K=5**

In code tree decision scheme the boxes are the states and arrows are the state transitions with probable input values shown in Fig. 3-17. The upper arrows directing out of each state denotes the transitions as if logic 0 input was applied and

lower arrows denotes as if logic 1 input was applied to the encoder. Also there are two informations on the transitions arrows which are expected encoder output and branch metric.

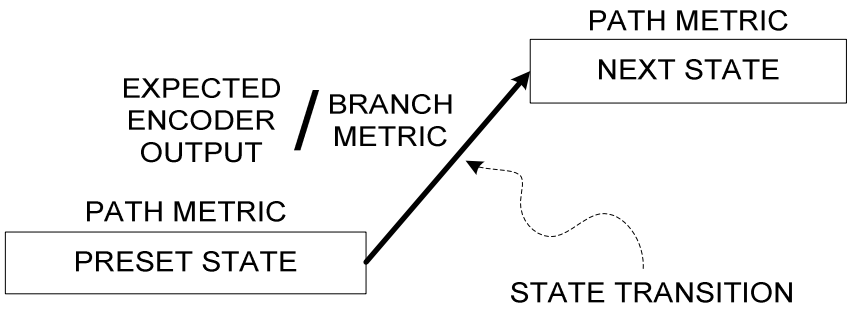
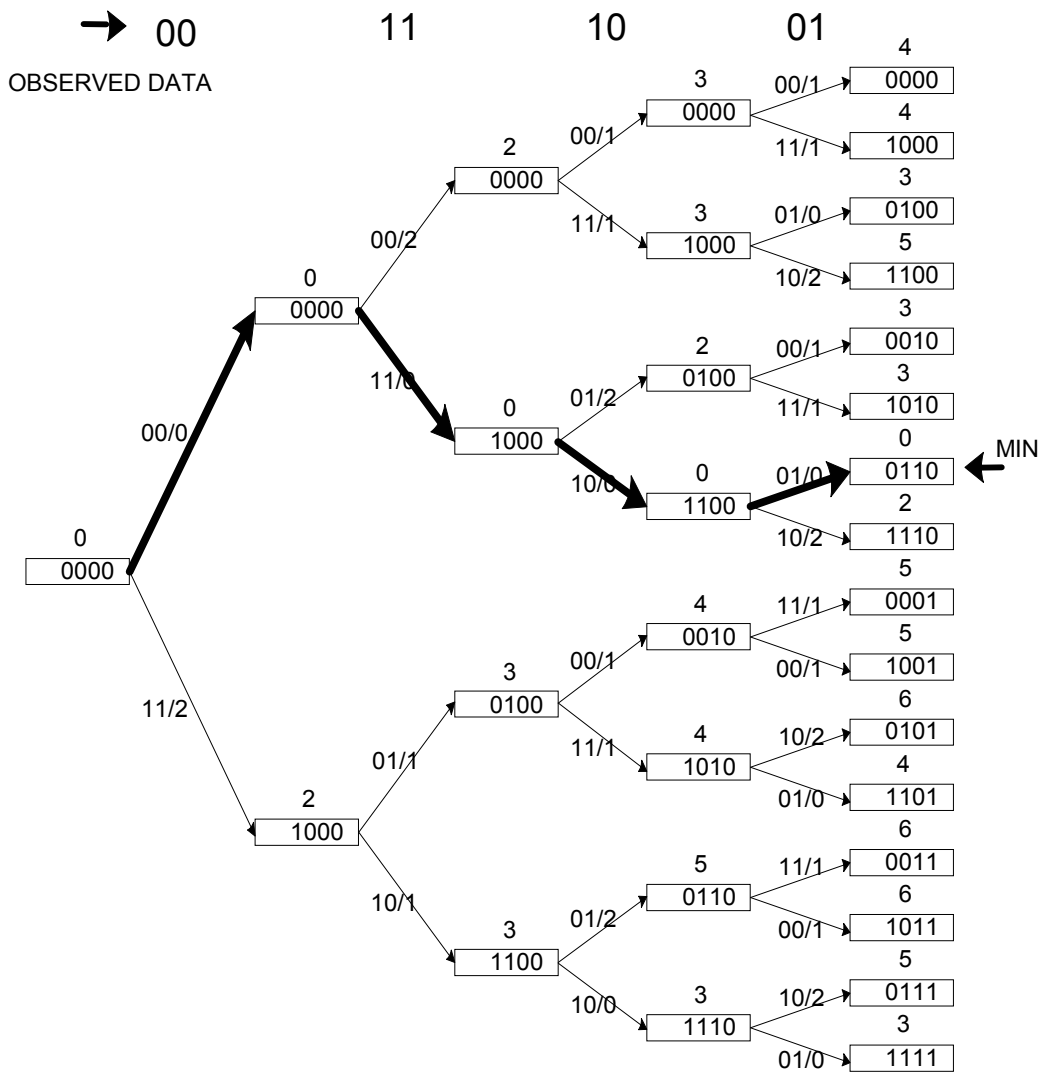


Fig. 3-17 Central part of decision tree

The expected encoder output is the output symbol of the convolutional encoder when the transitions between the specified states occurs due to the assumed input. The branch metric is dependent on a function of expected encoder output and received data input to show the transition probability. For a generic hard decision Viterbi decoder, branch metric is calculated to find the number of different bits between observed data and expected encoder output. For this purpose bitwise xor operation will be carried out between received data and expected encoder output to find the different bits which are logic 1 in the result of xor operation. Then summation of the bits in the result gives the total number of different bits.

Also all states have their own scale, called path metric. The path metric is a measure to indicate the probability of the input pattern leading the decoder to related state. The calculation of path metric is obtained by the cumulation of branch metric values of the path containing the state.



Message stream = 0110

Fig. 3-18 Decision Through Code Tree K=5

In Fig. 3-18 an example is given for K=5 Viterbi Decoding. The generator polynomials of the source encoder are "G(0)(D)= 1 + D^3 + D4" and "G(1)(D)= 1 + D + D^3 + D^4". As indicated in Viterbi decoder section the decoding starts from all zero initial state ("0000" state) with the path metric value of 0. From this state two branches exist for "0" and "1" input bits assumptions. The expected encoder output of these transitions are found by

$$\begin{bmatrix}\text{Output}\\ \text{symbol}\\ \text{matrix}\end{bmatrix}_{1xn} = \text{Mod2}\left(\begin{bmatrix}\text{Input bit}\\ \text{pattern of}\\ \text{length K}\end{bmatrix}_{1xK} \times \begin{bmatrix}\text{Generator}\\ \text{polynomial}\\ \text{matrix}\end{bmatrix}_{Kxn}\right)$$

Input bit pattern of length K = [assumed input bit, encoder state bits]$_{1x5}$

$$\text{Generator polynomial matrix} = \begin{bmatrix}1 & 1\\ 1 & 0\\ 0 & 0\\ 1 & 1\\ 1 & 1\end{bmatrix}_{5x2}$$

Output symbol matrix=Expected Encoder Output bits

For the transition with "0" input assumption branch goes to "0000" state and expected encoder output is calculated by

$$\begin{bmatrix}0 & 0\end{bmatrix}_{1x2} = \text{mod}\,2\left(\begin{bmatrix}0 & 0 & 0 & 0 & 0\end{bmatrix}_{1x5} \bullet \begin{bmatrix}1 & 1\\ 1 & 0\\ 0 & 0\\ 1 & 1\\ 1 & 1\end{bmatrix}_{5x2}\right)$$

For the transition with "1" input assumption branch goes to "1000" state and expected encoder output is calculated by

$$\begin{bmatrix}1 & 1\end{bmatrix}_{1x2} = \text{mod}\,2\left(\begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix}_{1x5} \bullet \begin{bmatrix}1 & 1\\ 1 & 0\\ 0 & 0\\ 1 & 1\\ 1 & 1\end{bmatrix}_{5x2}\right)$$

At this stage the decoder decides the branch metrics from the number of different bits between the observed data and the expected encoder output data. The observed symbol is "00" in this first stage. For "0" input assumption transition the observed data is the same as the expected encoder output so the branch metric of the

transition is 0. For “1” input assumption transition, where the expected encoder output is “11”, both of the two bits are different between observed and the expected data so the branch metric of the transition is 2.

After these transitions the path metric of the next states are calculated by addition of present state metric and branch metrics. Thus, the path metric of next states “0000” and “1000” are 0 and 2, respectively.

The traceback depth of this example is 4, so after the calculations stated in above statements for the 4 stage the decoder compares the path metrics of the final states and select the minimum valued state for traceback. Then the most probable message is decided from the assumed inputs of the path leading to the minimum valued state.

The code tree method is very useful for description of Viterbi decoder but not suitable for the IC implementation. Because of the major disadvantage of the code tree representation, which is the exponential increase in the number of states caused by the increase in the received data, a new method called trellis has been proposed to restrict the number of states.

- **Decision Through Trellis**

As stated in previous part because of the increasing number of states in the code tree method (Fig. 3-19), the trellis method was proposed (Fig. 3-20) in which the total number of states is restricted to the 2^{K-1}. In the code tree method both states labelled with “a” and “e” goes to the state “00” in different places of state “b” and state “f” with logic 0 inputs meaning that the repetition of the state machine started from “00” state. The observed data and state identifiers so the branch metrics in transitions b-c and f-g are the same. Also the branch metric of b-d and f-h are the same.

$$\lambda^{(c,b)} = \lambda^{(g,f)}$$

$$\lambda^{(d,b)} = \lambda^{(h,f)}$$

The path metric of states c and g are calculated by the equations below

$$\Gamma^{c} = \Gamma^{b} + \lambda^{(c,b)}$$

$$\Gamma^{g} = \Gamma^{f} + \lambda^{(g,f)}$$

Now if the traceback is started in this state, the better path in between the repetition states would be found by

$$betterpath = \min\left(\Gamma^{c}, \Gamma^{g}\right) = \min\left(\Gamma^{b}, \Gamma^{f}\right)$$

$$betterpath = \min\left(\Gamma^{d}, \Gamma^{h}\right) = \min\left(\Gamma^{b}, \Gamma^{f}\right)$$

In both cases the winner path is only related to the path metric of the b and f states. The successor states of the greater path metric state are useless in the computations because the path metric of the successors is directly related to the path metric of the predecessor and the greater valued predecessor will always direct the successor at the greater path metric values. So, the repetitive transitions are merged and the smaller path metric valued state is selected for computation of the path metrics.

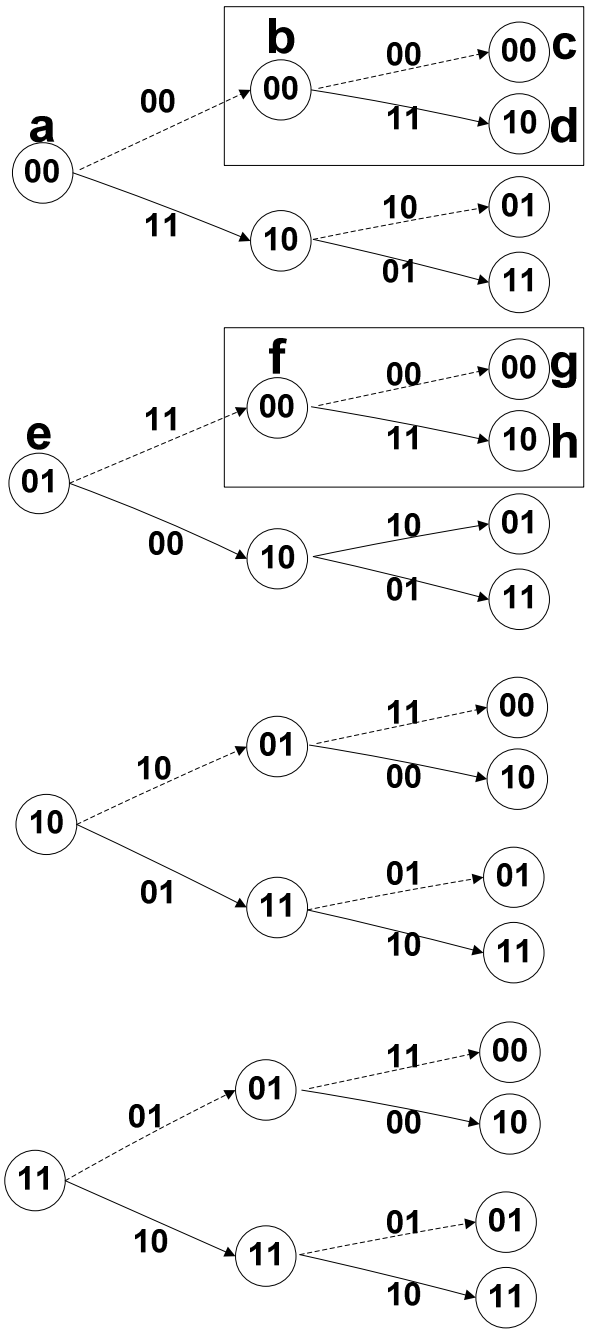


Fig. 3-19 Code Tree for K=3 r=1/2

After, merging all the repetitive states for every transitions the trellis structure of Fig. 3-20 is obtained.

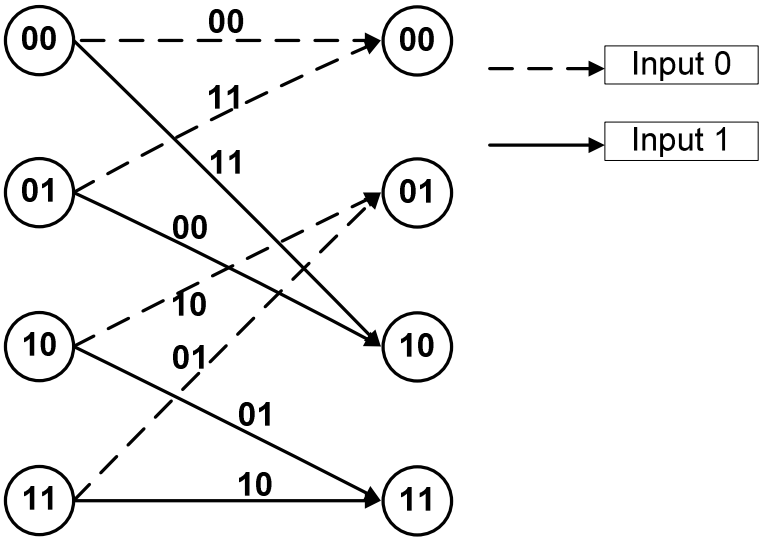


Fig. 3-20 Central Part of Trellis K=3 r=1/2

For the hard decision Viterbi decoder, all the transitions including the initialization instant are figured out in Fig. 3-21. In the trellis, as expected, after the K-1 number of observed data the trellis structure in Fig. 3-20 is reached for all stages.

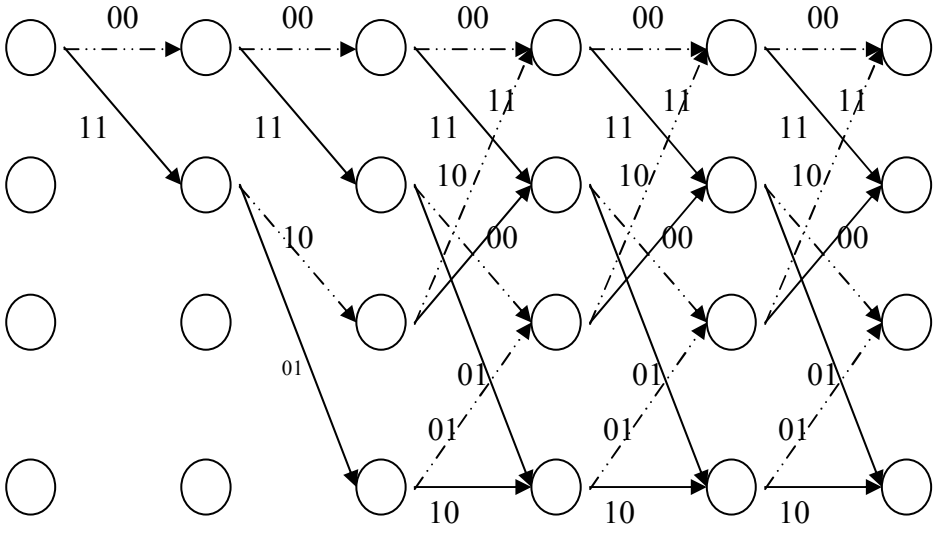


Fig. 3-21 Trellis K=3 r=1/2

3.3.3 Generic Viterbi Decoding Examples

In this section, cycle by cycle the Viterbi decoding process for hard decision is illustrated for two observed streams. In the first example the error correcting capability and in the second stream the malfunctioning of the decoding are exampled.

➢ **Decoding of All Zero Input Message with 2 Transmission Errors**

In this part, the all zero input message encoded and transmitted from the K=3 r=1/2 basic convolutional encoder is decoded. But the observed message contains two bits transmission errors.

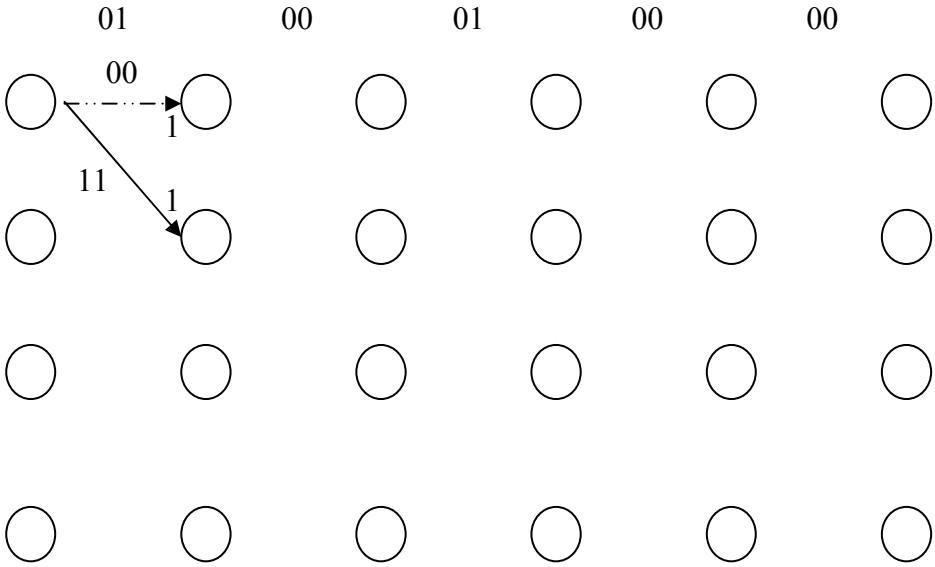


Fig. 3-22 Trellis Cycle 1 with Decoder Input "01"

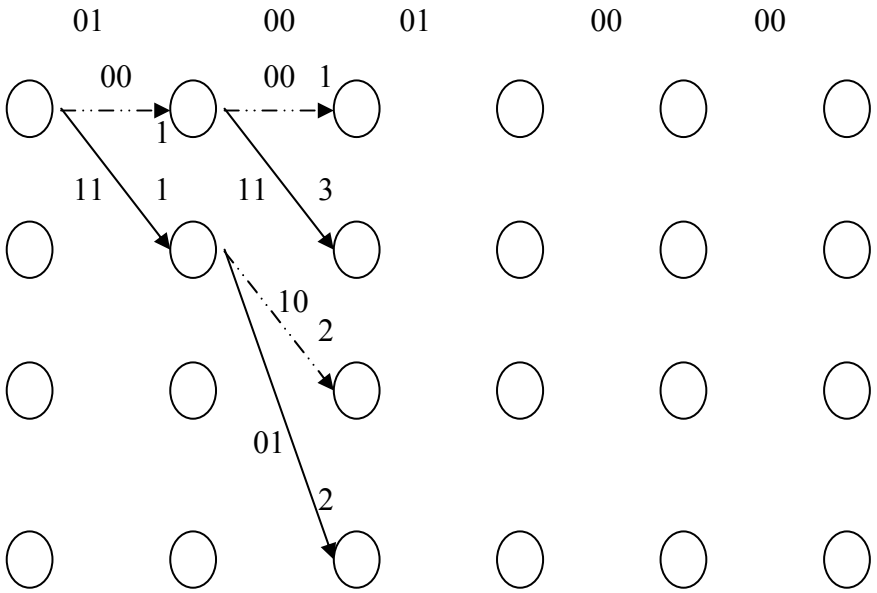


Fig. 3-23 Trellis Cycle 2 with Decoder Input "00"

The decoding process is same as the code tree method in initial 2 cycles (Fig. 3-22 and Fig. 3-23). In these two cycles, comparison of the better transition for the next state doesn't occur. Because only one defined branch exists directing to each next state. However, the 3rd and the rest cycles are made up of the same trellis structure containing the comparison phases.

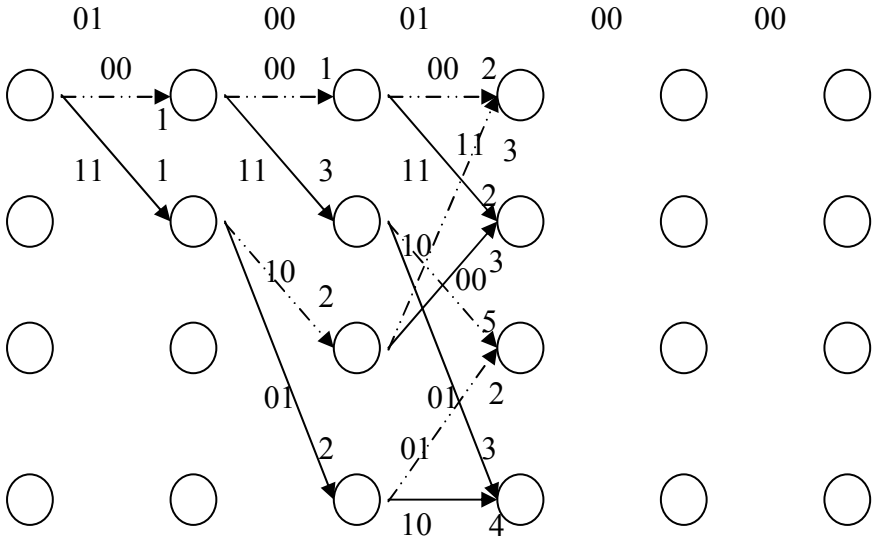


Fig. 3-24 Trellis Cycle 3 with Decoder Input "01"

For each next state the smaller metric containing path is selected.

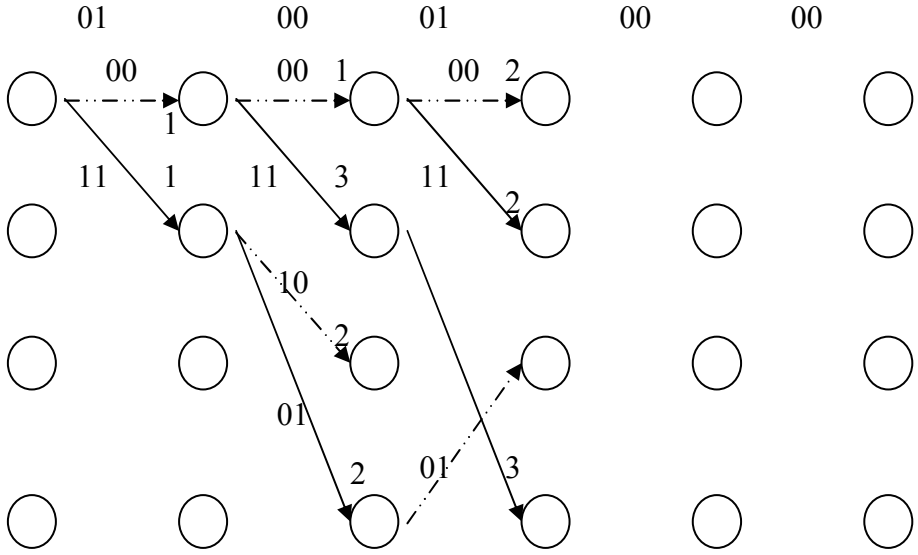


Fig. 3-25 Trellis Cycle 3 after Comparison of Minimum Branch

After the comparison phase, the improbable paths lost the connections to the next states.

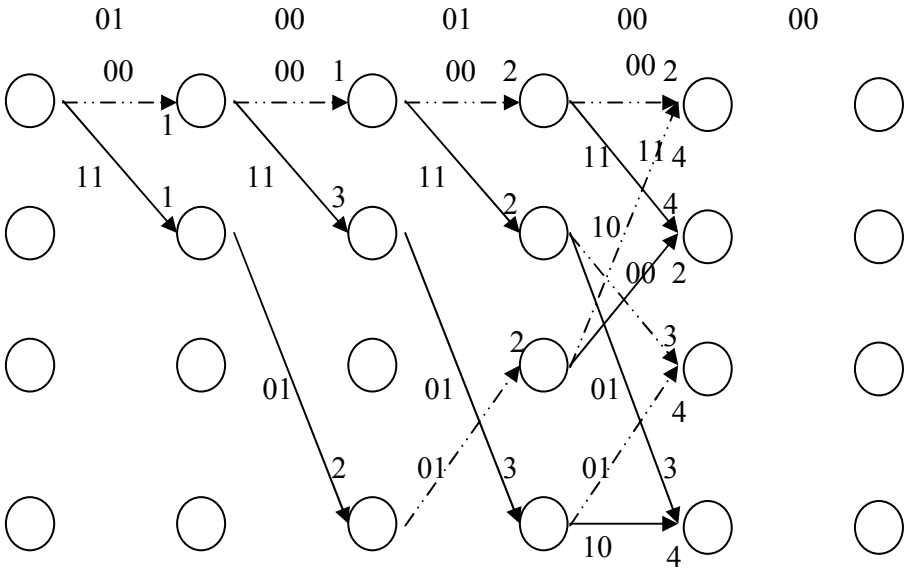


Fig. 3-26 Trellis Cycle 4 with Decoder Input "00"

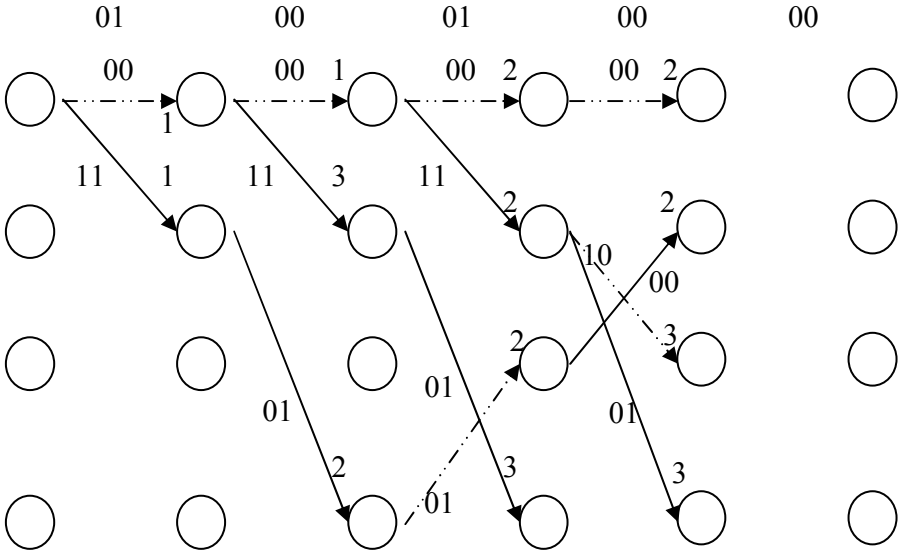


Fig. 3-27 Trellis Cycle 4 after Comparison of Minimum Branch

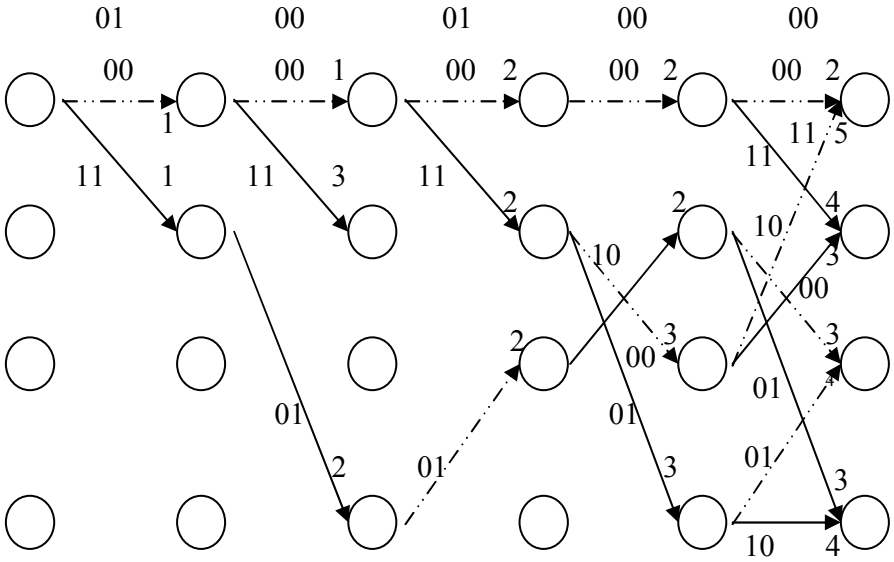


Fig. 3-28 Trellis Cycle 5 with Decoder Input "00"

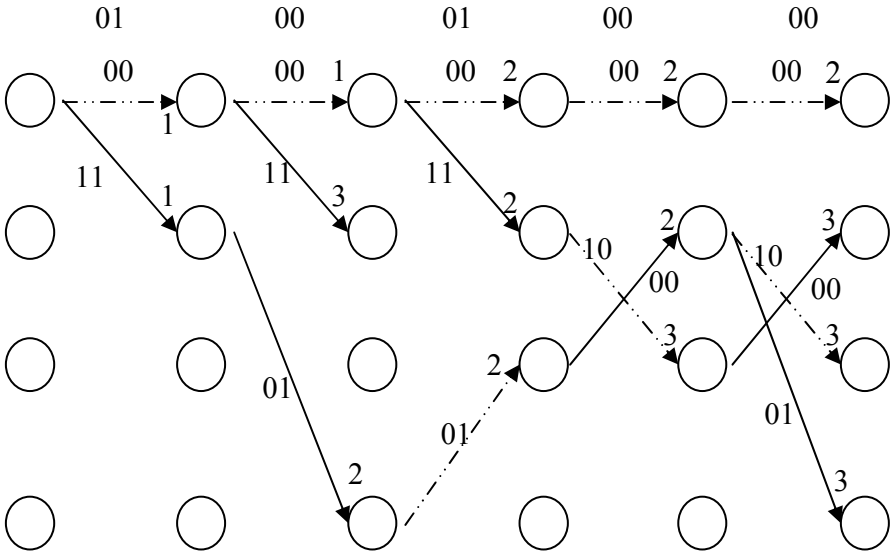


Fig. 3-29 Trellis Cycle 5 after Comparison of Minimum Branch

Finally, the state containing the minimum path metric between the final states is selected to start the traceback and the states leading to this final state is drawn as the best path (Fig. 3-30).

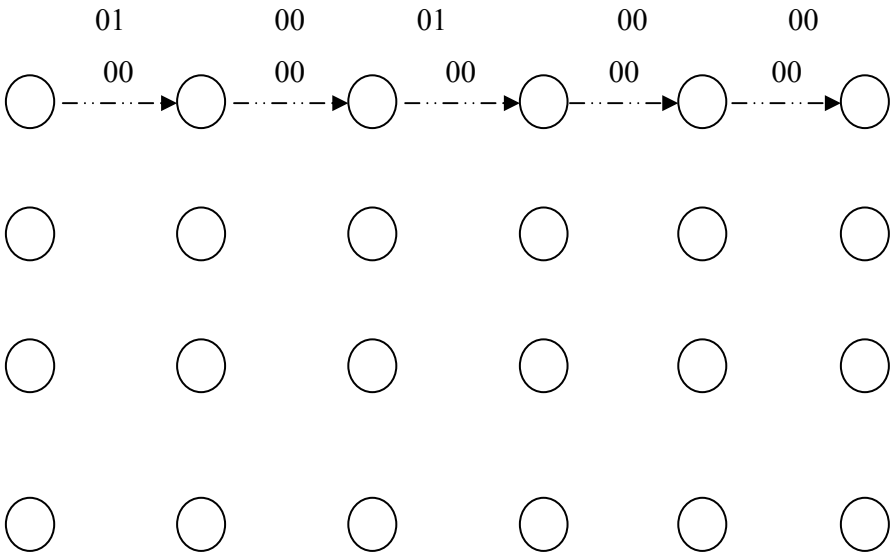


Fig. 3-30 Trace Back Path

➢**Decoding of All Zero Input Message with 3 Transmission Errors**

Now the same encoded message is aimed to be reached. But this time the received data contains three transmission errors.

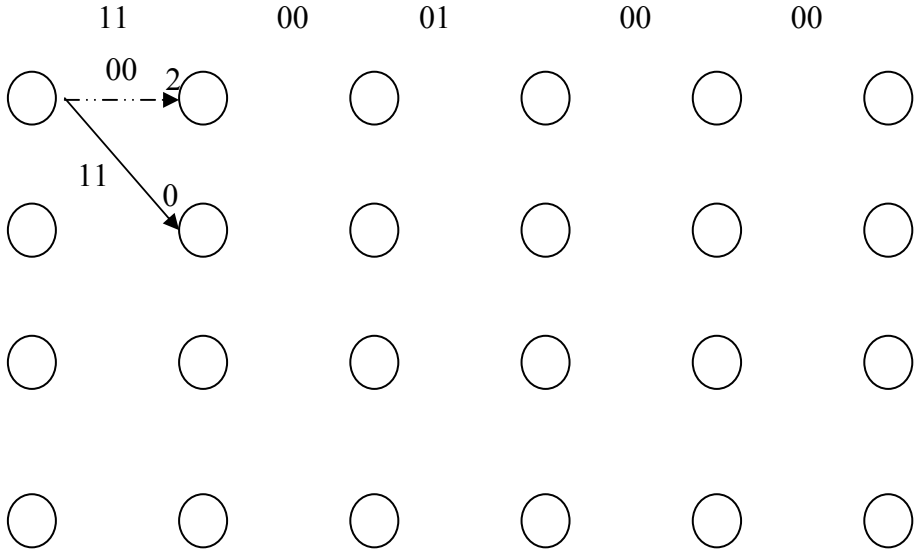


Fig. 3-31 Trellis Cycle 1 with Decoder Input "11"

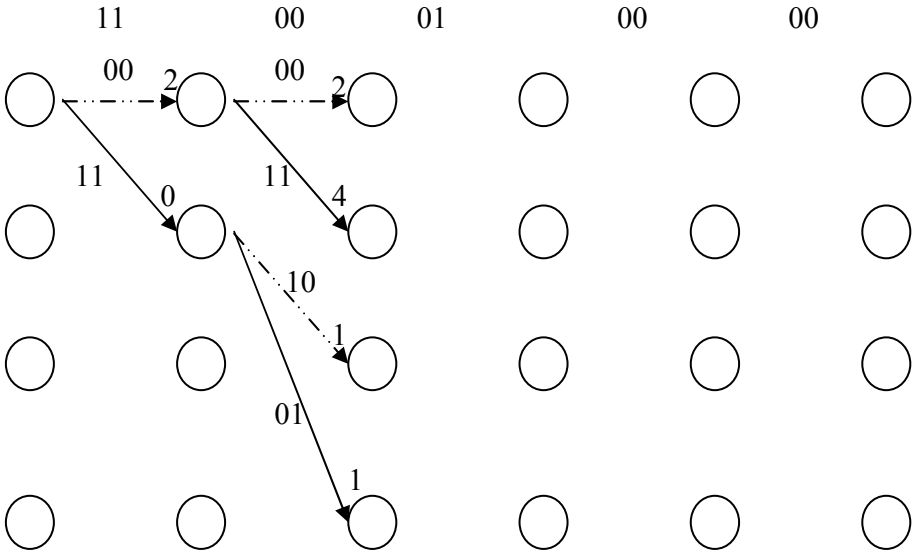


Fig. 3-32 Trellis Cycle 2 with Decoder Input "00"

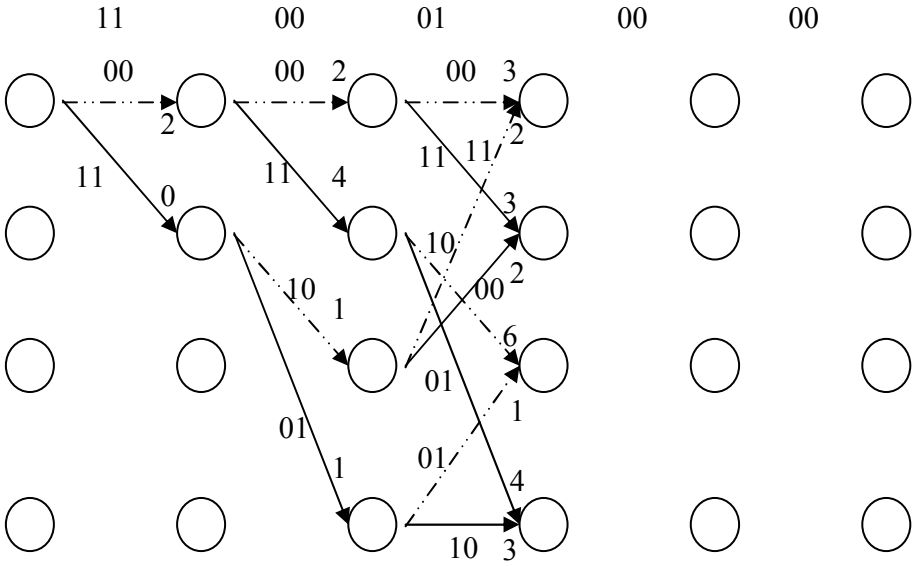


Fig. 3-33 Trellis Cycle 3 with Decoder Input "01"

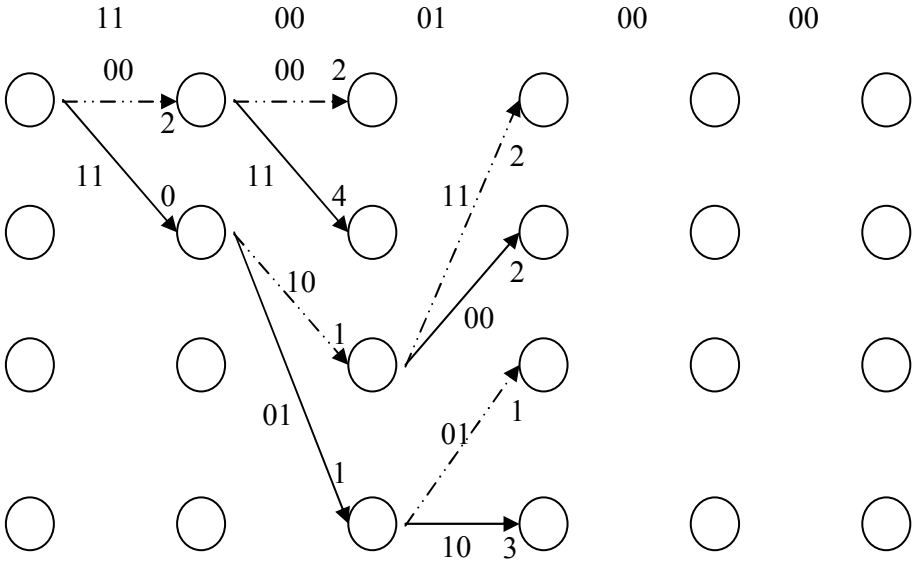


Fig. 3-34 Trellis Cycle 3 after Comparison of Minimum Branch

For the observed pattern containing 3 transmission error the Viterbi decoder discarded the path of all zero input transitions. So the probable traceback paths are shaped as in Fig. 3-35.

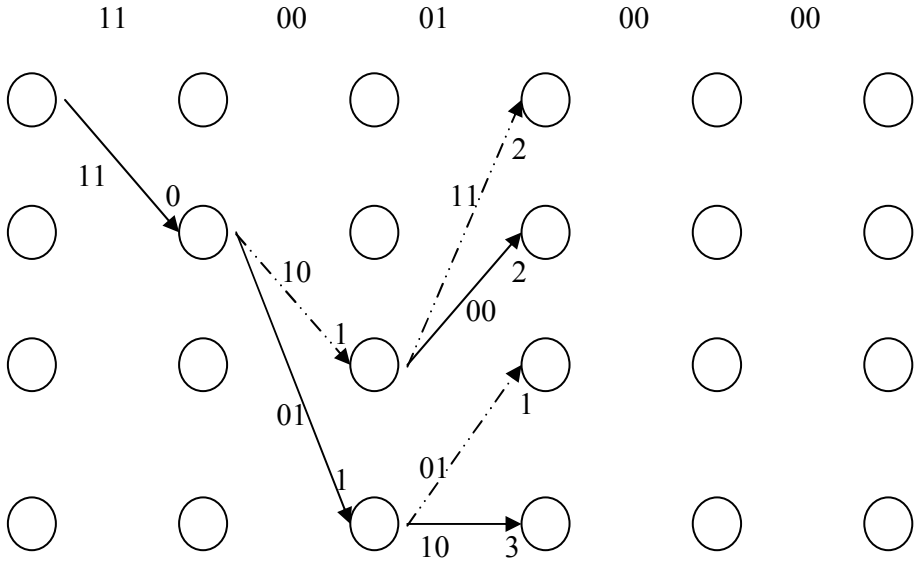


Fig. 3-35 Probable Trace Back Paths

CHAPTER 4

IMPLEMENTATION OF BASIC BUILDING BLOCKS OF VITERBI DECODER

4.1 General

In this chapter implementation of hard and soft decision Viterbi decoders are explained to get the decoder divided into sub modules, to gain the advantages of simulation simplification and debug period reduction. Creating the test platform of these components in the early stages of the thesis will decrease the overall development time because the specified modules will be used through out the thesis with some modifications to meet the design specific expectations.

4.2 Implementation of Hard Decision Viterbi Decoder

In this part, the implementation of the simplest Viterbi decoder called hard decision Viterbi decoder, to acquire the experience of the SystemC for the hardware modelling with the connections between ACS, BMU, DPRAM and TraceBack Unit, is stated. The operation of the implemented modules are explained below.

4.2.1 Dual Port RAM

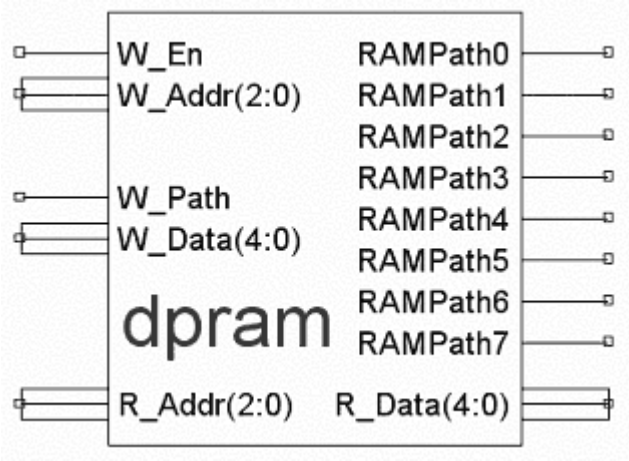


Fig. 4-1 DPRAM Symbol

Dual Port RAM (DPRAM) units are path metric keepers of the decoder. In each DPRAM to store the path metric only one register exists and several other registers exist to store probable transition information. So, the next state metric coming from W_Data is overwritten on the present state metric of previous cycle. In the next cycle without any addressing the trellis calculation is performed on the same registers as previous cycle. Using one register for the path metric storage to get rid of switching mechanism in trellis calculations is called in-place path metric updating.

On the contrary of the synchronous write operations controlled by W_En, to increase the speed of the trellis calculation, the read operations are performed asynchronously.

Before the Viterbi Decoding an initialization phase should be performed to assign initial path metrics of states. To initialize the path metrics on R_Data port, the port labelled as R_Addr is waited to be “000”. As figured in Chapter 3, in the initial K-1 cycles there are some states with undefined path metrics. To solve the implementation problem caused by the undefined states, the DPRAMs are created in two types. The only difference between these DPRAMs is the path metric values

in the initialization phase. The all zero state representing DPRAM supplies the path metric equal to 0 and the other DPRAMs supply very large path metrics in which the most significant bit is “1”. Assigning the undefined states in the initial cycle with a great path metric value makes these states improbable for traceback.

In the DPRAM symbol there are other ports named W_Addr and W_Path to store the probable transition information of each cycle. The RAMPathX ports reflect the probable transition leading to the next states in each cycle. In our implementation the traceback depth is selected as 8 so totally 8 RAMPath register so the 8 ports from RAMPath0 to RAMPath7 are implemented.

```
DPRAM Unit Implementation with SystemC
W_Addr  W_Data  W_Path  W_En  R_Addr  R_Data  RAMPath0  RAMPath1  RAMPath2  RAMPath3  RAMPath4  RAMPath5  RAMPath6  RAMPath7
000 00000 0 0 000 00000 0 0 0 0 0 0 0 0
000 00000 0 0 000 00000 0 0 0 0 0 0 0 0
001 00001 1 0 000 00000 0 0 0 0 0 0 0 0
001 00001 1 1 000 00000 0 1 0 0 0 0 0 0
010 00010 1 0 000 00000 0 1 0 0 0 0 0 0
010 00010 1 1 000 00000 0 1 1 0 0 0 0 0
011 00011 0 0 000 00000 0 1 1 0 0 0 0 0
011 00011 0 1 000 00000 0 1 1 0 0 0 0 0
100 00100 0 0 000 00000 0 1 1 0 0 0 0 0
100 00100 0 1 000 00000 0 1 1 0 0 0 0 0
101 00101 0 0 000 00000 0 1 1 0 0 0 0 0
101 00101 0 1 000 00000 0 1 1 0 0 0 0 0
110 00110 1 0 000 00000 0 1 1 0 0 0 0 0
110 00110 1 1 000 00000 0 1 1 0 0 0 1 0
111 00111 1 0 000 00000 0 1 1 0 0 0 1 0
111 00111 1 1 000 00000 0 1 1 0 0 0 1 1
111 00111 1 0 000 00000 0 1 1 0 0 0 1 1
111 00111 1 0 000 00000 0 1 1 0 0 0 1 1
111 00111 1 0 001 00001 0 1 1 0 0 0 1 1
111 00111 1 0 010 00010 0 1 1 0 0 0 1 1
111 00111 1 0 011 00011 0 1 1 0 0 0 1 1
111 00111 1 0 100 00100 0 1 1 0 0 0 1 1
111 00111 1 0 101 00101 0 1 1 0 0 0 1 1
111 00111 1 0 110 00110 0 1 1 0 0 0 1 1
111 00111 1 0 111 00111 0 1 1 0 0 0 1 1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 4-2 DPRAM Simulation Result

4.2.2 Add Compare Select Unit (ACS)

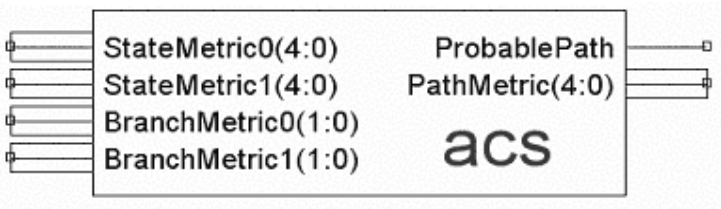


Fig. 4-3 ACS Symbol

The ACS units are the building blocks of the trellis structure. The ACS unit adds the branch metric and the path metric (StateMetric), then, determines the transitions causing the smaller path metric for the next state. The ACS units retrieve the state metric of previous cycle from the R_Data port of DPRAMs, after the calculations, the resultant path metric of the probable path is generated and written on the W_Data port of DPRAMs. The ACS units also compute and write an information into the DPRAM identifying the probable transition which will be used in traceback operation. The operation of ACS is summarized on the block diagram Fig. 4-4.

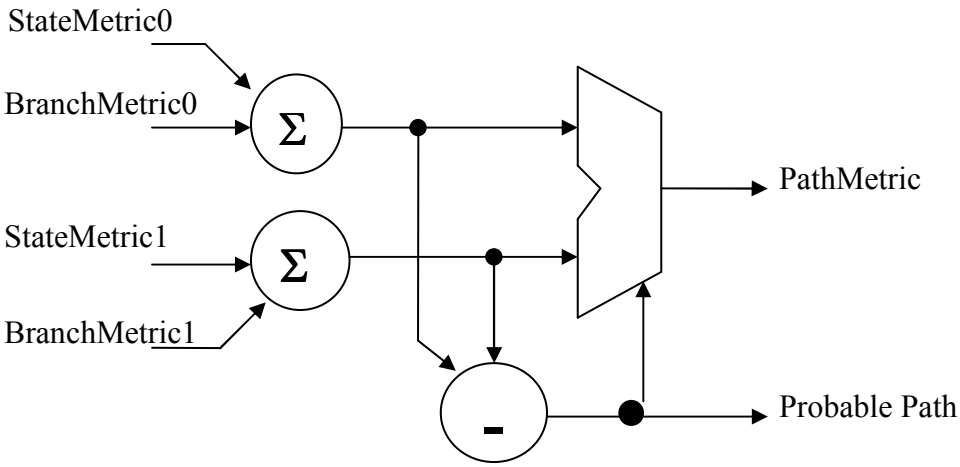


Fig. 4-4 ACS Block Diagram

```
Add Compare Select Unit Implementation with SystemC
        SM0     BM0     SM1     BM1     PM      PP
        0       0       0       0       0       1
        0       0       0       0       0       1
        3       1       4       2       4       0
        12      2       11      1       12      1
        15      1       14      0       14      1
        14      0       15      2       14      0
        24      0       25      1       24      0
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 4-5 ACS Monitor Based Simulation Result

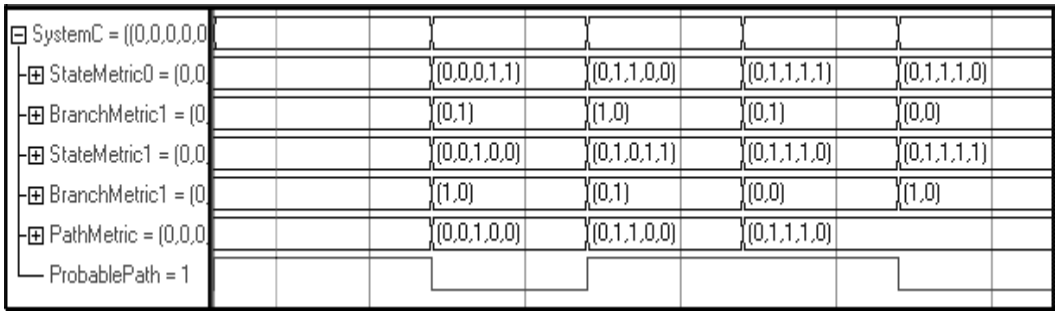


Fig. 4-6 ACS Winbeta Based Simulation Result

4.2.3 Branch Metric Unit (BMU)

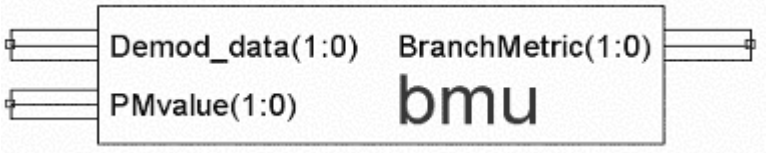


Fig. 4-7 BMU Symbol

Demod_data = Observed Data; PMvalue = Expected Encoder Output;

The BMU receives observed data from the channel (Demod_data) and expected encoder output (PMvalue) then computes the branch metric from the number of different bits between the observed data and expected encoder output. The branch metric can be seen

as the distance between the received data and expected encoder output in the state space.

Demod_data	PMvalue	BranchMetric
00	00	00
00	00	00
01	00	01
10	00	01
11	00	10
00	01	01
01	01	00
10	01	10
11	01	01
00	10	01
01	10	10
10	10	00
11	10	01
00	11	10
01	11	01
10	11	01
11	11	00

Fig. 4-8 BMU Monitor Based Simulation Result

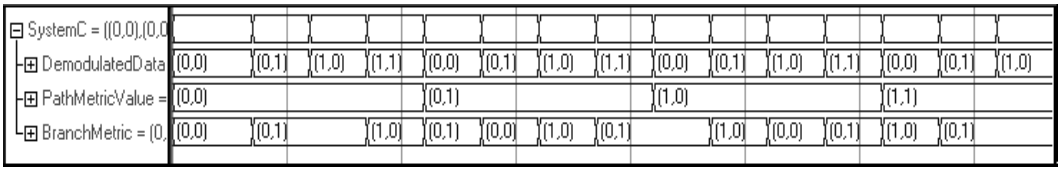


Fig. 4-9 BMU Winbeta Based Simulation Result

4.2.4 ACSDPRAM

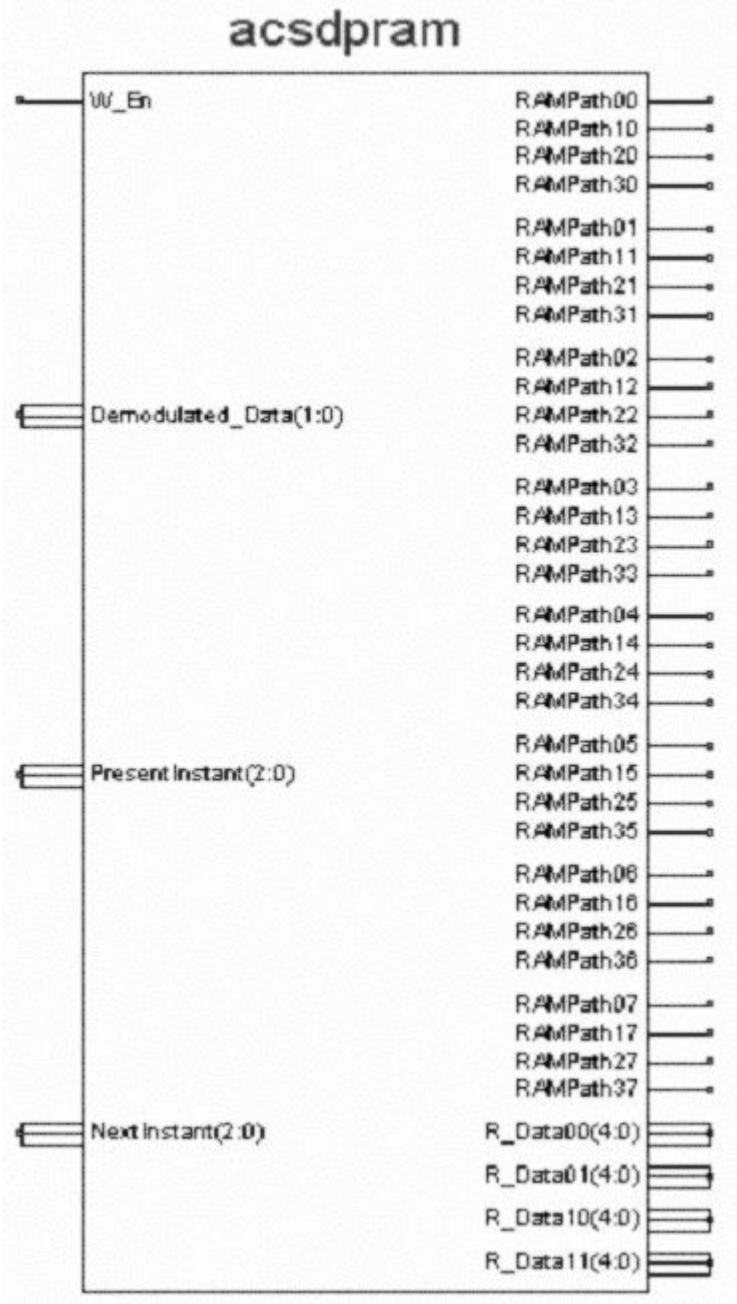


Fig. 4-10 ACSDPRAM Symbol

ACSDPRAM unit is the trellis structure composed of the BMU, DPRAM and ACS blocks.(Fig. 4-11 and Fig. 4-13) In the implementation port named as NextInstant

represents the trellis cycle number and is used to identify the address of RAMPath for probable transition storage. The PresentInstant is also the cycle representer and used to inform the DPRAMs for initialization at value of “000”. In any time the BMU units calculate the branchmetrics using observed data incoming from Demodulated_Data port and the ACS unit performs calculation of probable paths and the path metrics related to these paths asynchronously. After the W_En activation the path metrics are updated with the values calculated by ACS units. The Read_Data ports output the path metrics of the states to decide the probable path just before the traceback. The ports prefixed with RAMPath shows the probable transitions of each cycle to estimate the original message in traceback. For this purpose the least significant bit of the present state identifier of the probable transition is stored as RAMPath value.

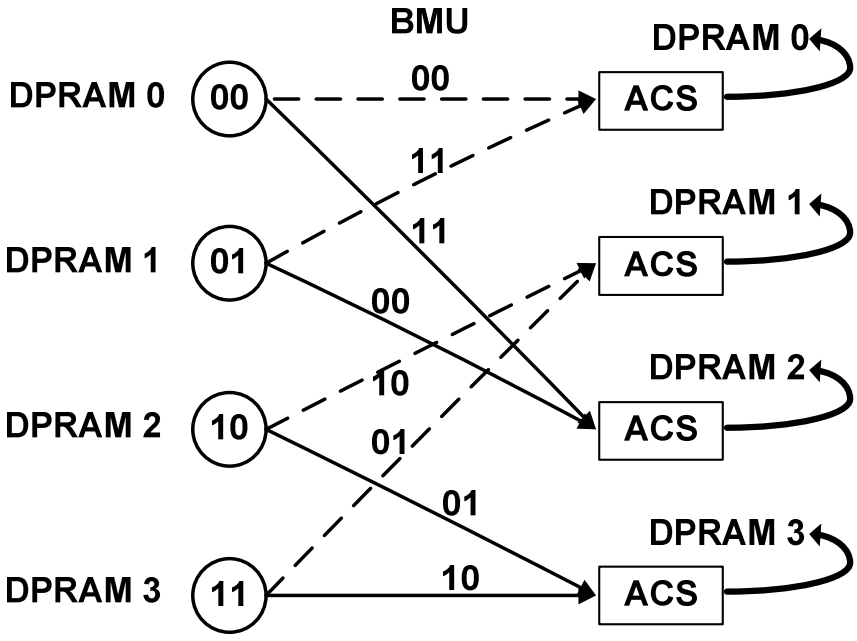


Fig. 4-11 ACSDPRAM Block Diagram

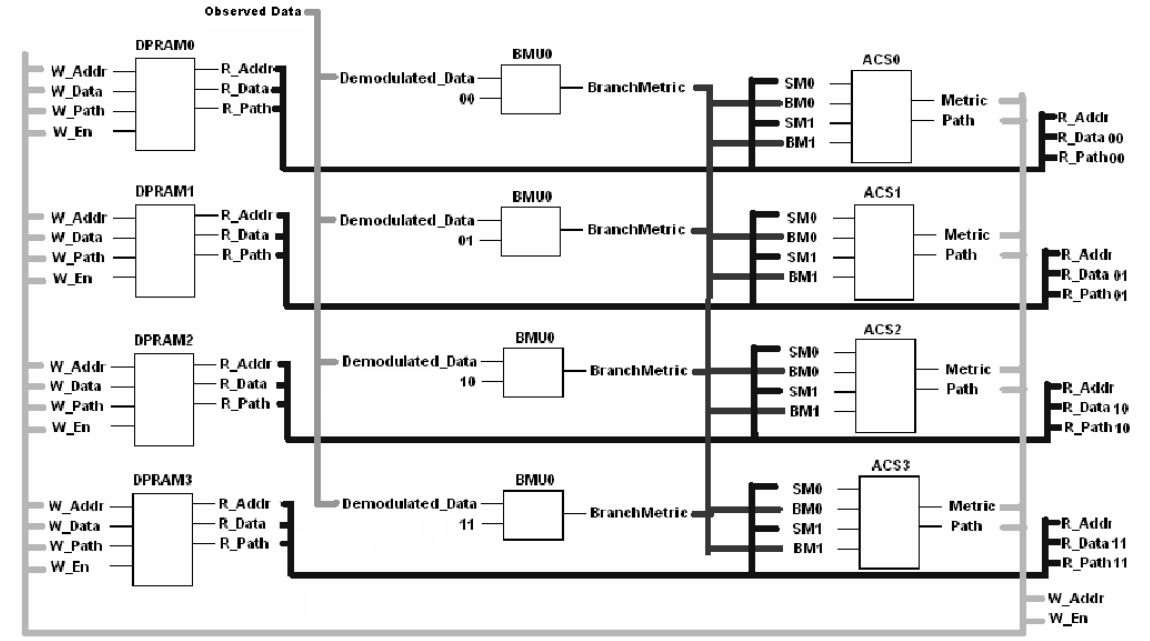


Fig. 4-12 ACSDPRAM Detailed Block Diagram

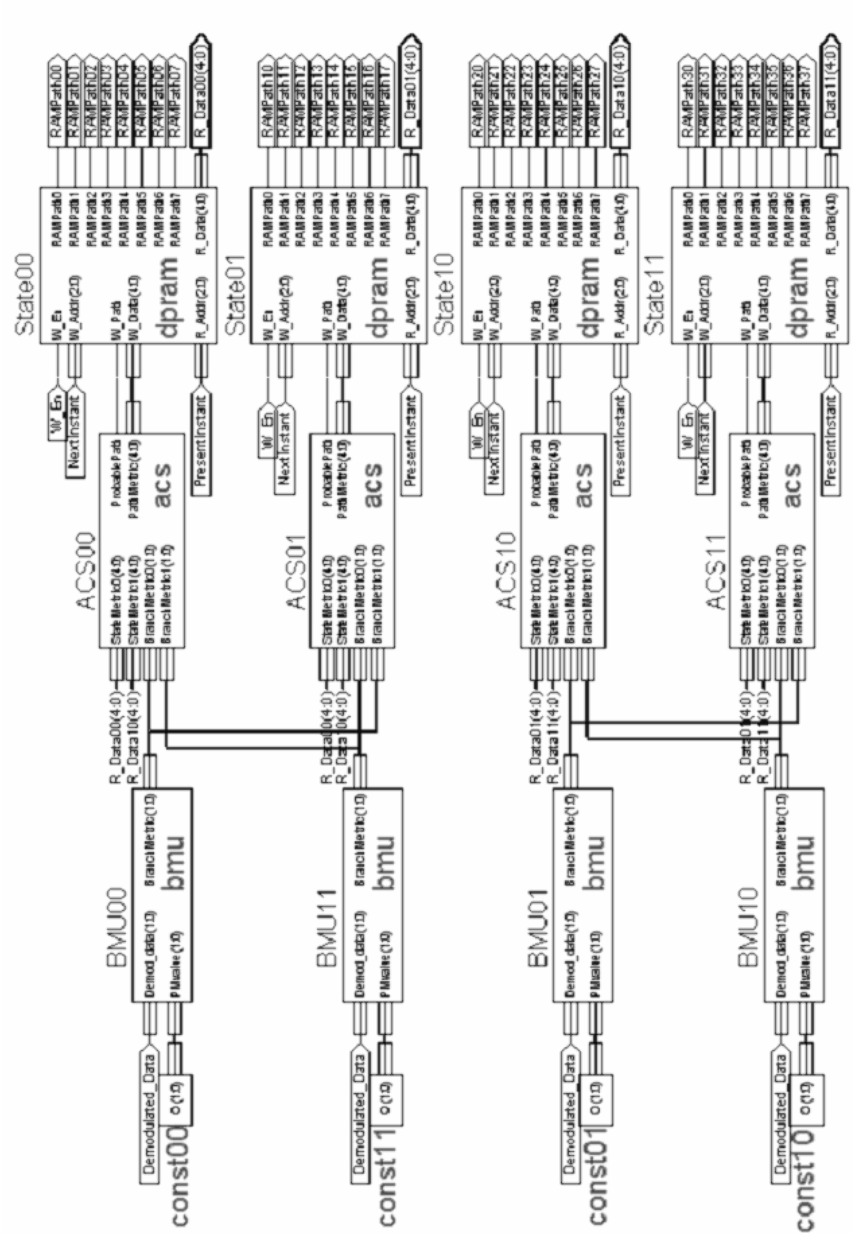


Fig. 4-13 ACSDPRAM Schematic

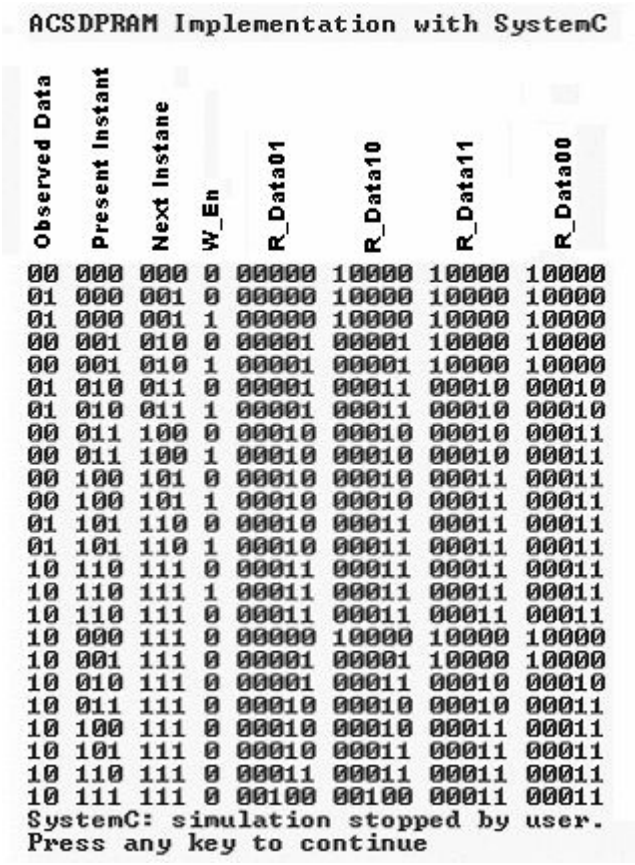


Fig. 4-14 ACSDPRAM Monitor Based Simulation Result

4.2.5 MinDetector

For the decision of the state containing minimum path metric, the mindetector module shown in Fig. 4-15 is implemented. The mindetector module inputs the path metrics of all states and outputs the identifier of the minimum path metric valued state.

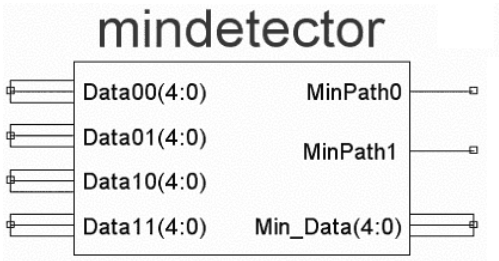


Fig. 4-15 Mindetector Symbol

4.2.6 Trace-Back Unit (TBU)

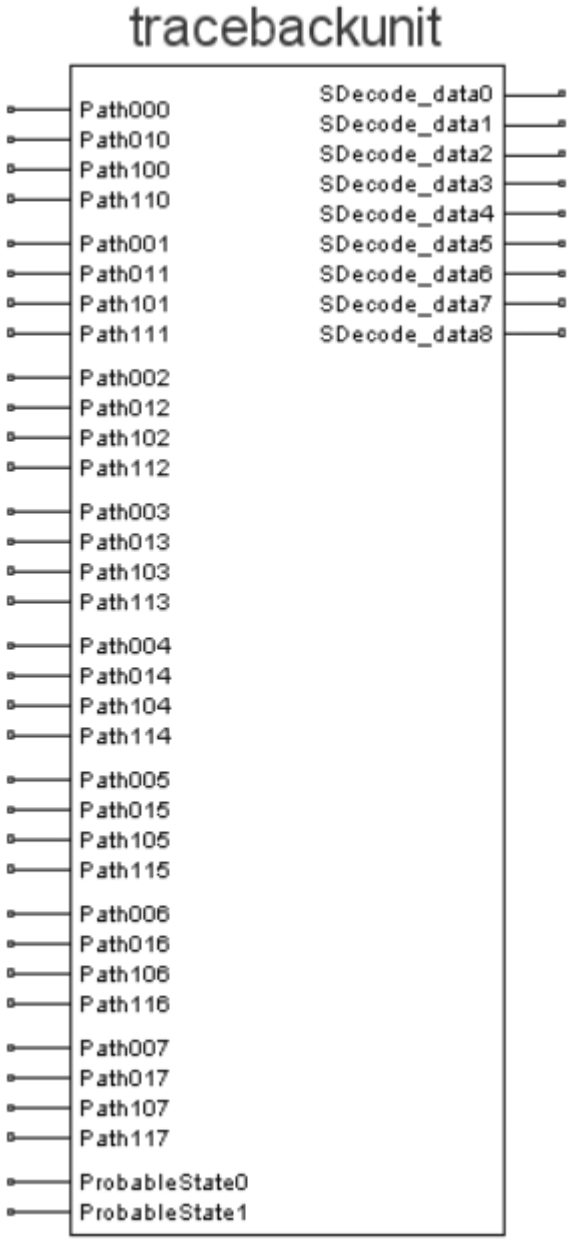


Fig. 4-16 TRACE-BACK UNIT Symbol

The traceback unit is a module, which estimates the original message using the information comings from ACSDPRAM module. After a predetermined length of observed data block, the TBU inputs the identifier of the minimum path metric

valued state from min detector. Then, TBU establishes an optimal path from the most probable final node all the way to the all zero state at the beginning of the trellis by using the transition information in each trellis cycle.

The Traceback Unit performs the reverse process of the convolutional encoding. In the convolutional coder the next state identifier is the one bit right shifted version of the present state taking the message bit for the most significant bit of the next state identifier and the least significant bit of the present state identifier is lost in the next state, as declared in Chapter 3.

Present State Identifier = S_n ... S_2 S_1 S_0

Input Message Bit = I

Next State Identifier = I S_n ... S_3 S_2 S_1

In the Viterbi decoder the state transitions depending on the convolutional coding are recreated for all probable states in the trellis to decide the transition probabilities. In the trellis as well, the least significant bit of the present state are lost in the transition towards the next state. So, for each trellis cycle the least significant bit of the present state directing to the next state is stored in the RAMPath registers of the next state DPRAM at the address which is equal to the trellis cycle (Fig. 4-17).

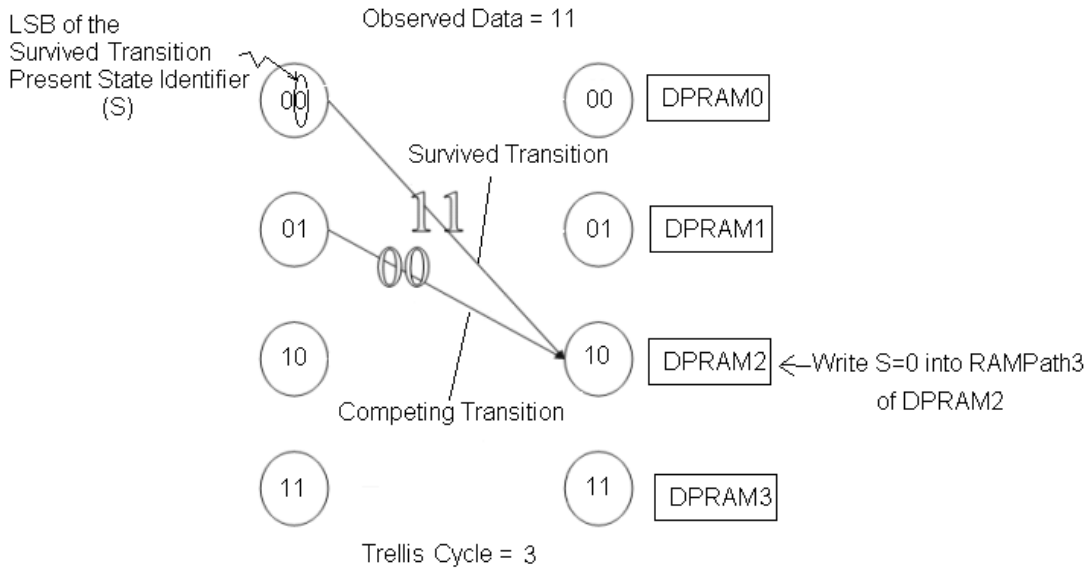


Fig. 4-17 Trellis Decision

The RAMPath registers store the survived transition information so they are called the survivor registers. In the tracing of the best path, the traceback unit uses the values in the survivor registers as in Fig. 4-18. The Fig. 4-18 is an example traceback operation for which the best state after the final trellis cycle is calculated as “01”. The state identifier is stored for the first two bits of the estimated message. The bit value of the survivor register corresponding the final cycle (cycle 7) in the DPRAM1 is “0”. So shifting the present state identifier to the left taking the survivor register as input and erasing the most significant bit, the state of “10” is obtained for the best state of cycle 6. The recursive operation is continued up to the cycle 0 while storing the survivor register values for the estimated message.

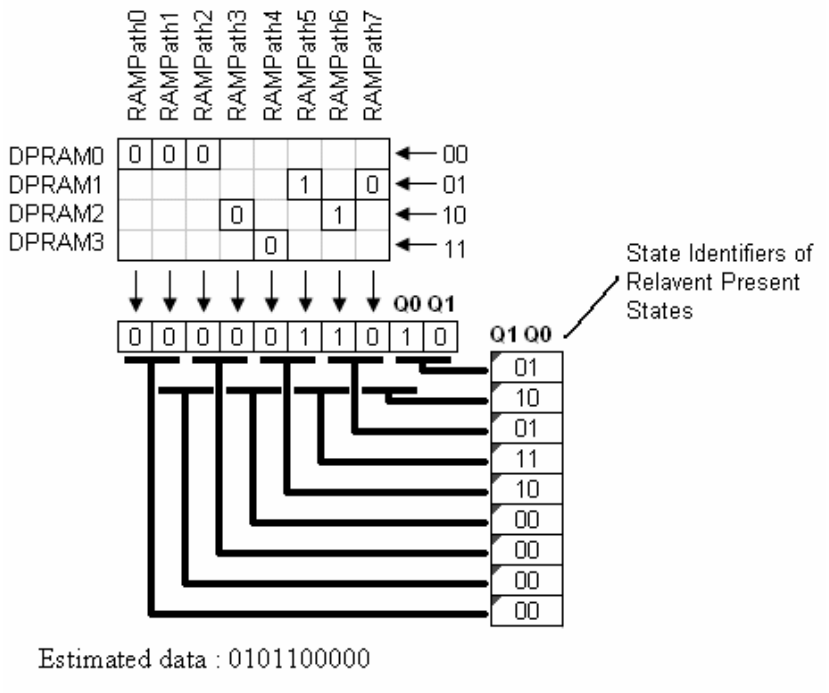


Fig. 4-18 Trace Back Algorithm

The traceback unit is composed of the MUX2x4 modules shown in Fig. 4-15 whose truth table is given in Table 4-1.

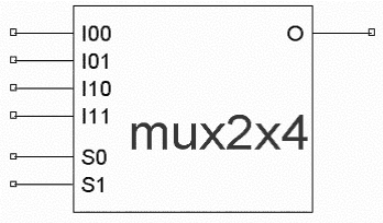


Fig. 4-19 MUX2x4 Symbol

Table 4-1 Truth Table of Mux2x4

S1	S0	O
0	0	I00
0	1	I01
1	0	I10
1	1	I11

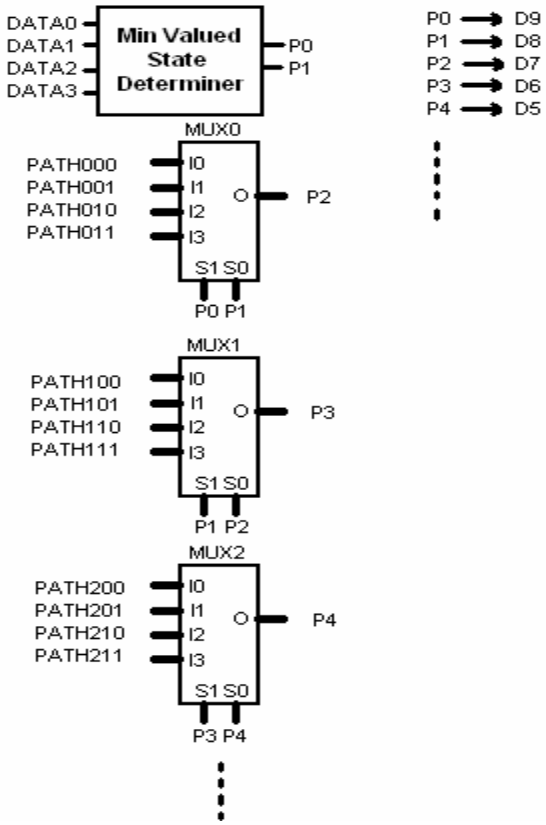


Fig. 4-20 TraceBack Operation Block Diagram

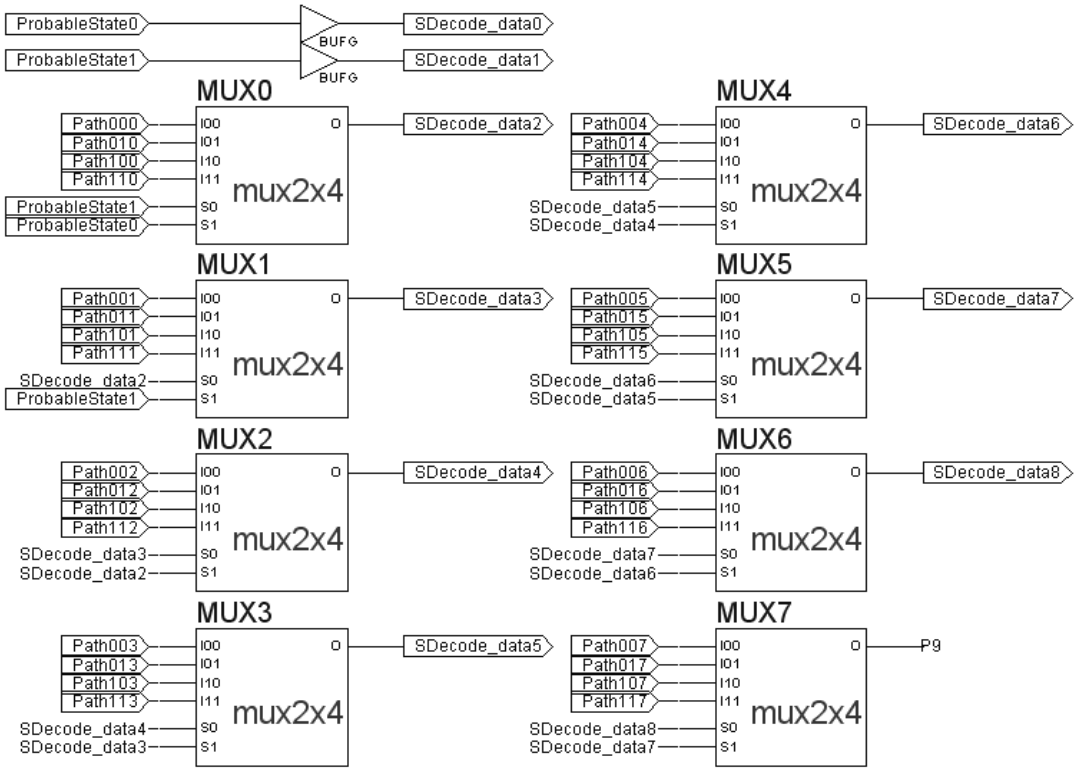


Fig. 4-21 TraceBack Unit Schematic

4.2.7 Hard Decision Viterbi Decoder Module

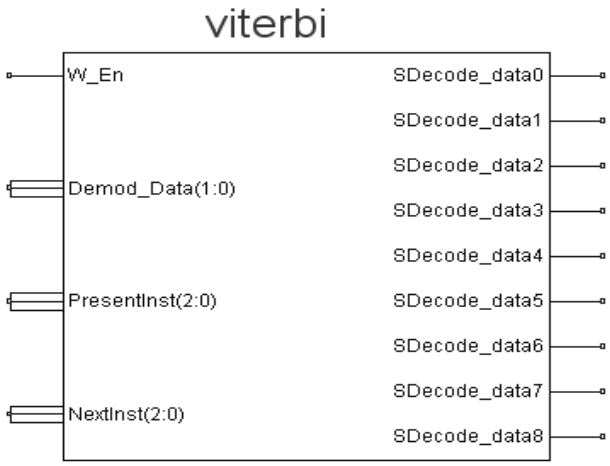


Fig. 4-22 Viterbi Decoder Symbol

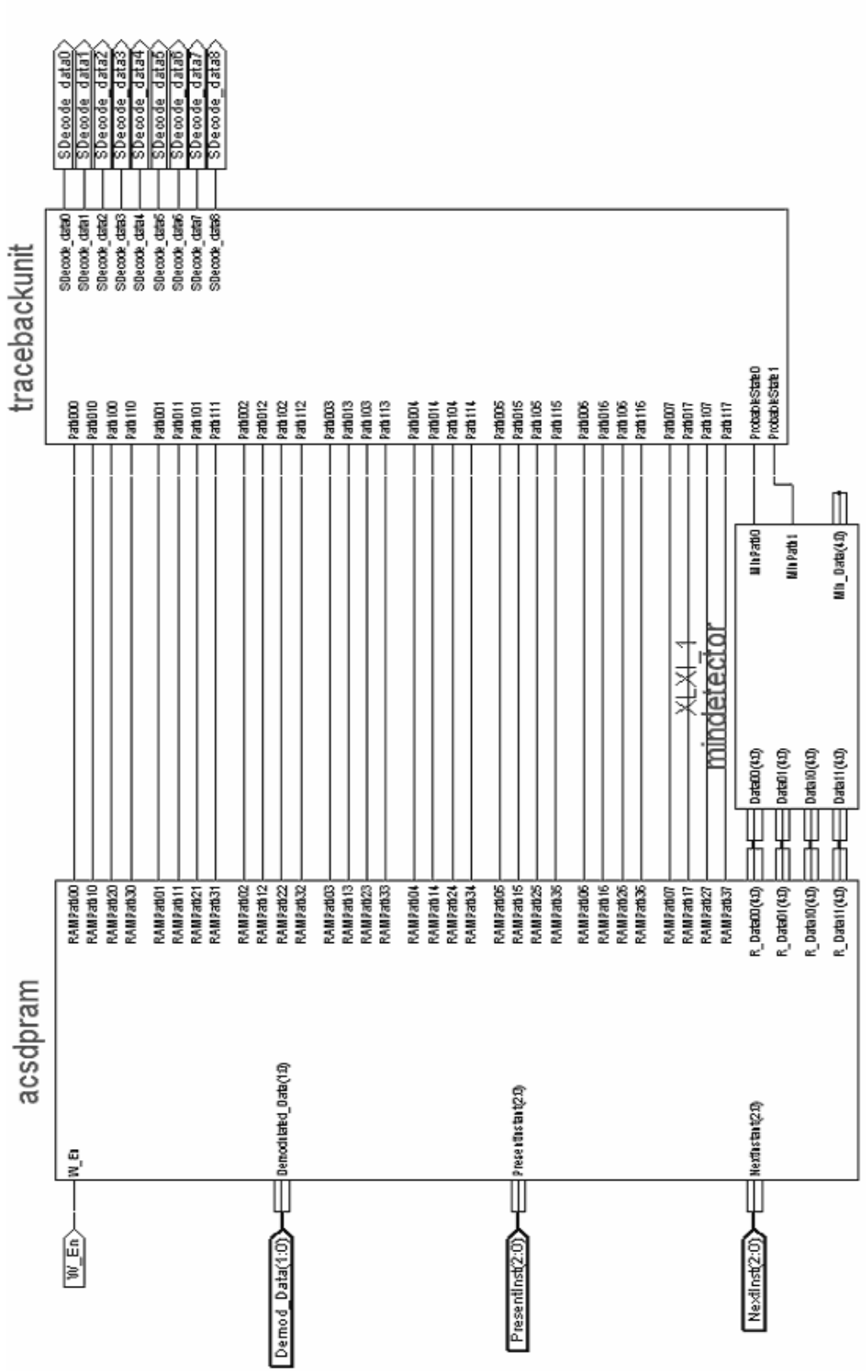


Fig. 4-23 Viterbi Schematic

Finally, the hard decision Viterbi decoder is obtained with the connections shown in Fig. 4-23.

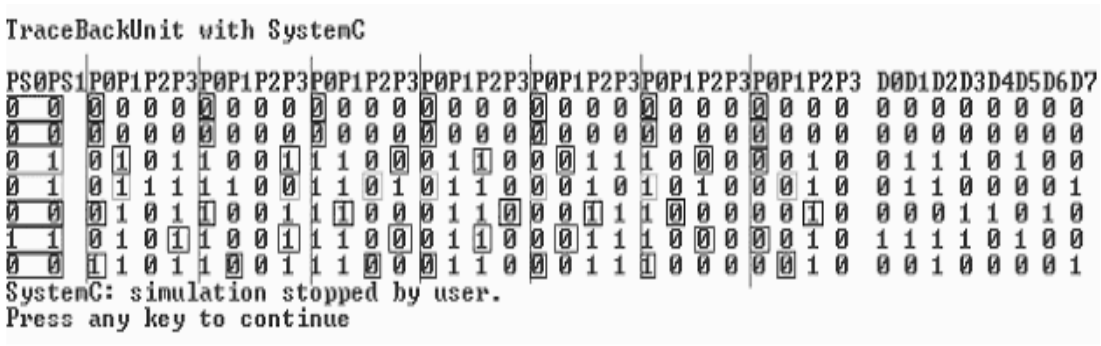


Fig. 4-24 Viterbi Decoder Simulation

The simulation result of the connections between ACSDPRAM, mindecoder and TBU is performed and simulated in Fig. 4-24.

In Chapter 3 the Viterbi Decoding of all zero input message was exampled for two observed streams to see the performance of the hard decision Viterbi decoder. In one of these examples the two errors were embedded in observed data and the estimated message was corrected through decoder as the original message. This case is simulated on the hardware and the simulation result is given in Fig. 4-25

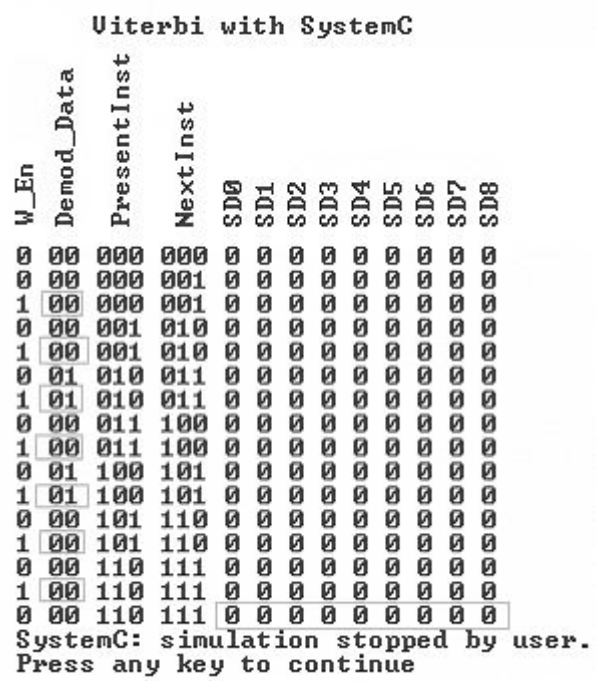


Fig. 4-25 Decoding of Observed Data with two errors

The three errors embedded in observed data case is also simulated on the hardware and the simulation result is given in Fig. 4-26

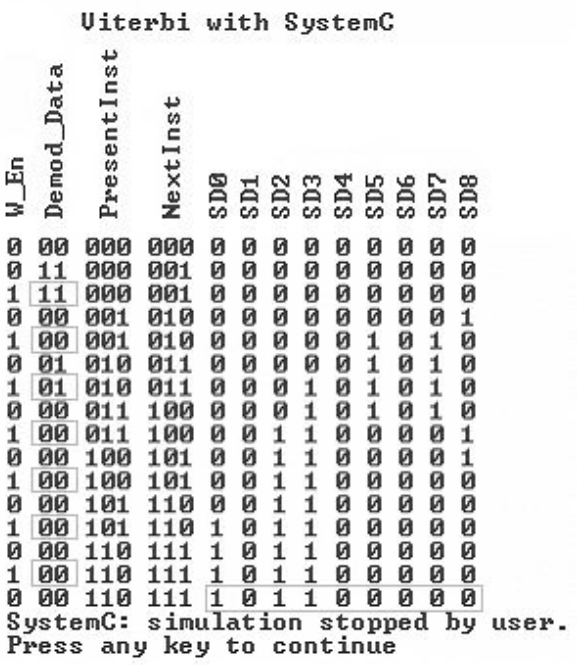


Fig. 4-26 Decoding of Observed Data with three errors

4.3 Implementation of Soft Decision Viterbi Decoder

In the previous part, the implementation of the hard decision Viterbi decoder was stated. In hard decision decoding, a stream of symbols are passed through a threshold detector in the receiver to obtain bit values definitely separated into the two quantized logic levels, one or zero as in Fig. 4-27.

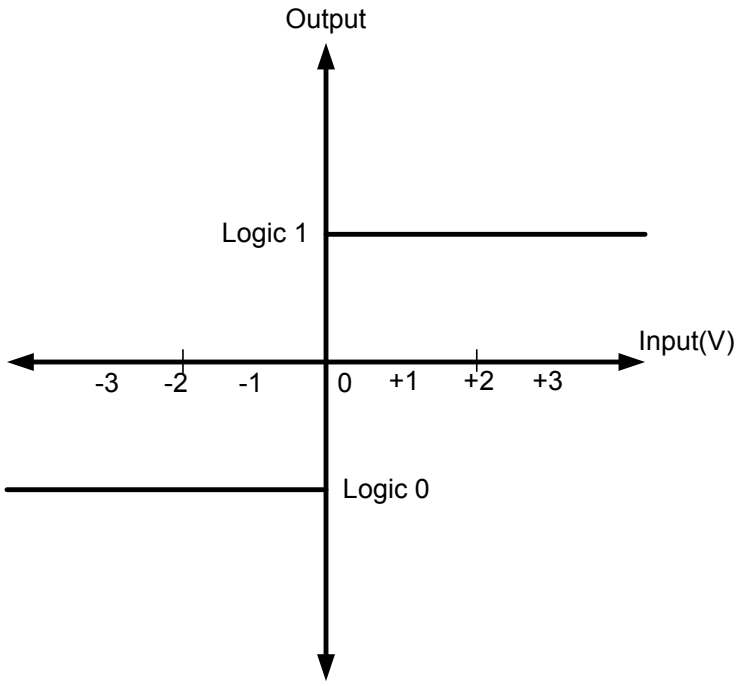


Fig. 4-27 Hard Decision Input Example for ±2V Transmitted Symbol

For a hard decision input the closeness of the received value to the transmitted voltages is not important in branch metric calculations so the receiver responds with the same branch metric for the 0.1V and 2.1V for the message pulsed with +2V and -2V signalling levels to signify the logic 1 and logic 0, respectively. In this situation, the received signal of 2.1V is very probable to be transmitted as logic 1 (+2V) and almost not probable to be transmitted as logic 0 (-2V) because the channel corruption is 0.1V for +2V transmitted signal and 4.1V for -2V. However, 0.1V is approximately equal probable to be transmitted either +2V or -2V signal levels. Thus, a mechanism to distinguish the probability of the received signals according to the value of the corruption voltage altered the transmitted voltage is required to increase the fidelity of the decoded message. For this reason a concept called soft decision viterbi decoding was developed in the literature where the input symbol is not only quantized to 1 and 0, but also quantized by several threshold levels to improve the reliability of the branch metrics relative to the input message. In Fig. 4-28 four level quantized scheme is given.

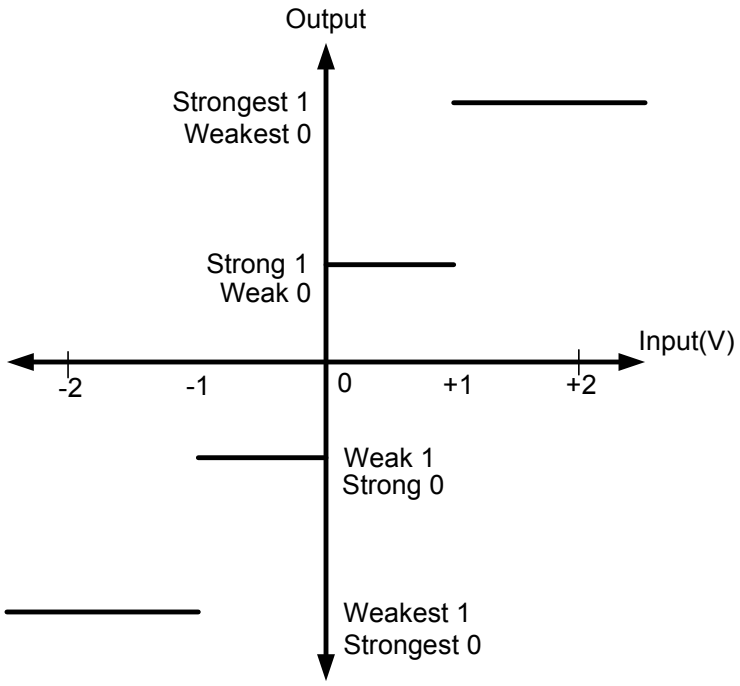


Fig. 4-28 Soft Decision Input Example

Instead of the two strict probability value in hard decision inputs, in soft decision the intermediate voltage values are assigned to several probability values for condense measure concerning probability of a binary bit being 1 or 0. For example Fig. 4-29 shows the probability of the received voltage to be belonging to the +2V transmitted signal. In other words, in the Gaussian curve located with mean value of +2, which corresponds to the probability of the received voltage belonging to the logic 1 transmitted symbol, the area of each region, separated by -1, 0, +1, represents the probability for the input voltage falling within the range reasoning of the logic 1 transmitted.

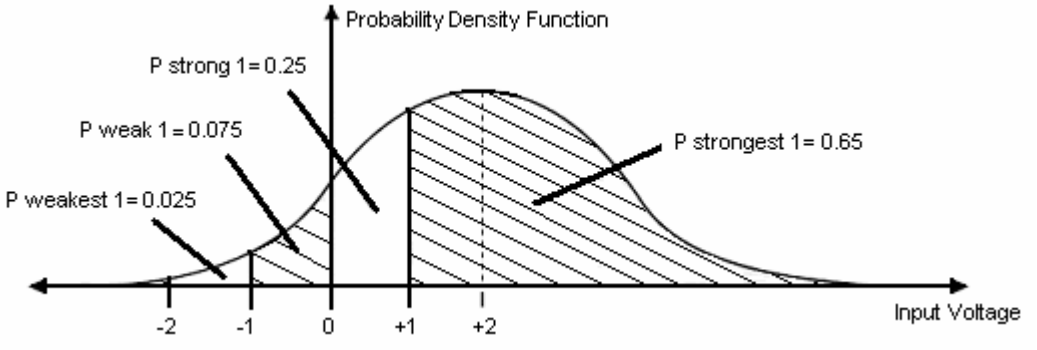


Fig. 4-29 Soft Decision Probabilities

To summarize the above discussion, the probability table is given in Table 4-2

Table 4-2 Probability of quantized voltages vs. the transmitted logic

	Strongest 1 /Weakest 0	**Strong 1 /Weak 0**	**Weak 1 /Strong 0**	**Weakest 1 /Strongest 0**
logic 0	**0.025**	**0.075**	**0.25**	**0.65**
logic 1	**0.65**	**0.25**	**0.075**	**0.025**

The probability calculations are normally obtained by multiplications, and the multiplications causes large hardware size and time consuming operations. So the probability values are converted to the logarithmic probability values to simplify the hardware into adder-subtracter circuits operating in one clock cycle. Also, the probability values are always less than 1 so the logarithmic values are always smaller than 0. Multiplicating the logarithmic probabilities by -1 makes all probability values to positive and leads the hardware to only adder circuitries.

From now on;

$$\text{Logarithmic Probability} = -\log(\text{Probability})$$

The logarithmic probabilities of Table 4-2 is given in Table 4-3.

Table 4-3 Logaritmic Probabilities of quantized voltages vs. the transmitted logic

	Strongest 1 /Weakest 0	Strong 1 /Weak 0	Weak 1 /Strong 0	Weakest 1 /Strongest 0
logic 0	**1.602**	**1.125**	**0.602**	**0.187**
logic 1	**0.187**	**0.602**	**1.125**	**1.602**

For example, the branch metric calculation by using the logarithmic probabilities are performed as follows. The branch metric of transitions for the subsequent 1.5V and -0.5V input voltages are calculated by addition of the two probability values corresponding to each encoder output bit. The 1.5V is in the strongest 1 / weakest 0 region (Fig. 4-28 and Fig. 4-29) whose logarithmic probabilities are 1.602 and 0.187 for expected encoder output bit of logic 0 and logic 1 (Table 4-3), respectively. In other case, the -0.5V is in the Weak 1/Strong 0 region whose probabilities are 0.602 and 1.125 for expected encoder output bit of logic 0 and logic 1, respectively. For the "00" expected encoder output, the first bit is 0 and the input voltage is in Weakest 0 region so the probability is 1.602 and the second bit of expected encoder output is 0 and received voltage is -0.5V causing 0.602 probability. From the summation of these two probabilities, the branch metric of the branch is found as 2.204 (Fig. 4-30) . For the "11" branch all two expected encoder outputs are '1' so the branch metric is calculated by the summation of probabilities of 0.187 and 1.125.

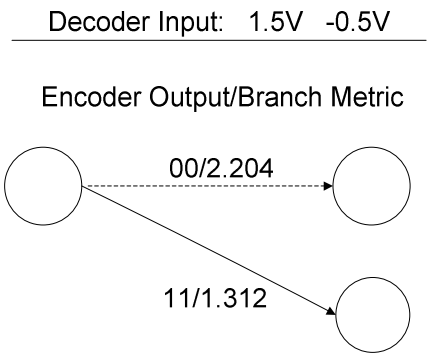


Fig. 4-30 Branch Metric Calculation

In hardware implementation the floating-point calculations causes hardware complexity. So the logarithmic probabilities are scaled by 100 and then rounded to integers (Table 4-4).

Table 4-4 Scaled Logaritmic Probabilities

	Strongest 1 /Weakest 0	Strong 1 /Weak 0	Weak 1 /Strong 0	Weakest 1 /Strongest 0
logic 0	**160**	**113**	**60**	**19**
logic1	**19**	**60**	**113**	**160**

For the hardware implementation, the modules implemented in hard decision Viterbi decoder are modified to obtain K=3 R=1/2 soft decision Viterbi decoder.

4.3.1 BMU

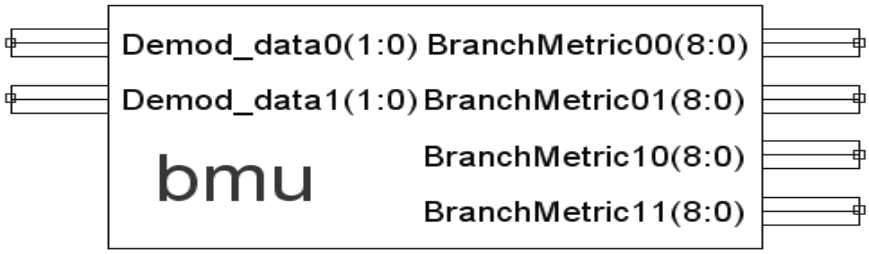


Fig. 4-31 BMU Symbol

The BMU module takes the two input symbol from the analog to digital converter of the receiver and calculates the 9 bits scaled logarithmic transitions probabilities for branch metric values of 00, 01, 10 and 11 expected encoder output.

4.3.2 ACS

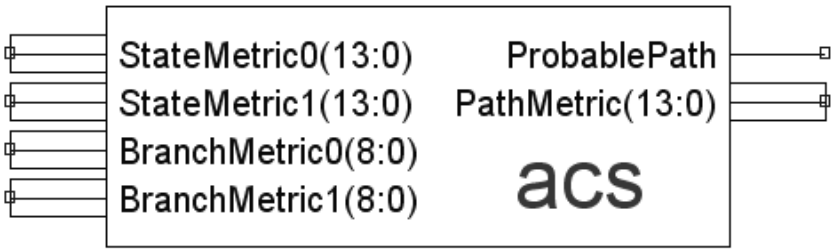


Fig. 4-32 ACS Symbol

The ACS module inputs the branch metrics and the state metrics of half a butterfly and outputs better path also the next state metric for this better path as in hard decision Viterbi decoder. The state metrics are selected as 14 bits for calculation of maximum traceback depth of 16.

4.3.3 DPRAM and MinDetector

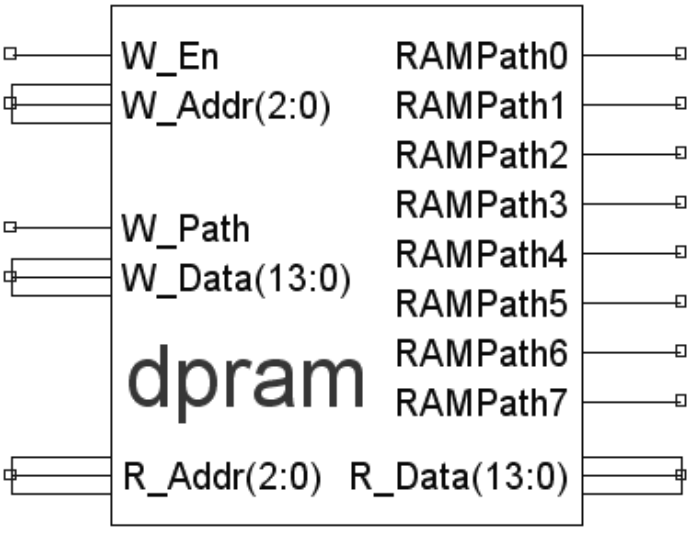


Fig. 4-33 DPRAM Symbol

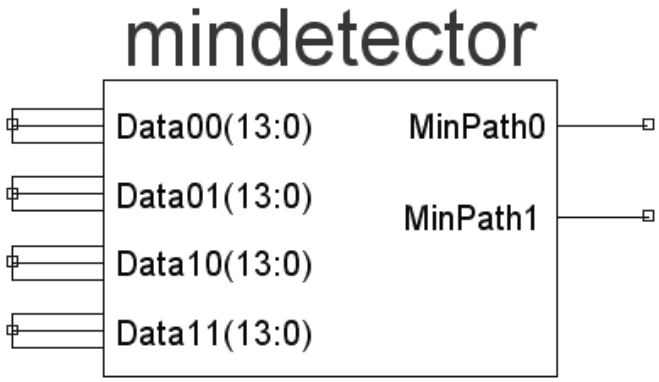


Fig. 4-34 MINDETECTOR Symbol

The DPRAM and MINDETECTOR modules are same as in the hard decision Viterbi decoder with only modification of the state metric related signals to 14 bits.

4.3.4 ACSDPRAM

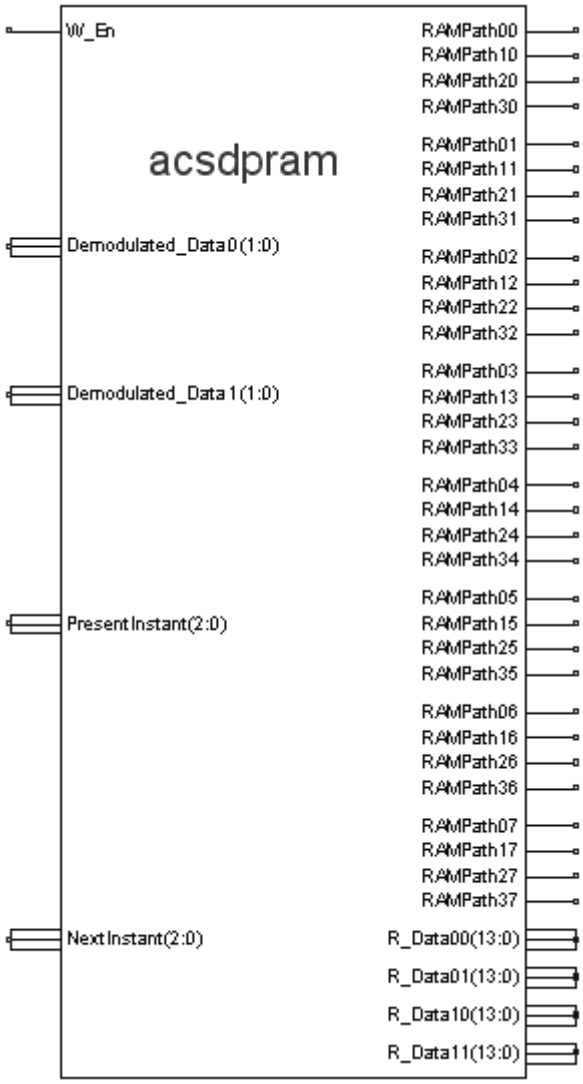


Fig. 4-35 ACSDPRAM Symbol

ACSDPRAM module is used to create the trellis structure as before. However, in this implementation one branch metric unit creates the branch metrics of all transitions in the trellis.

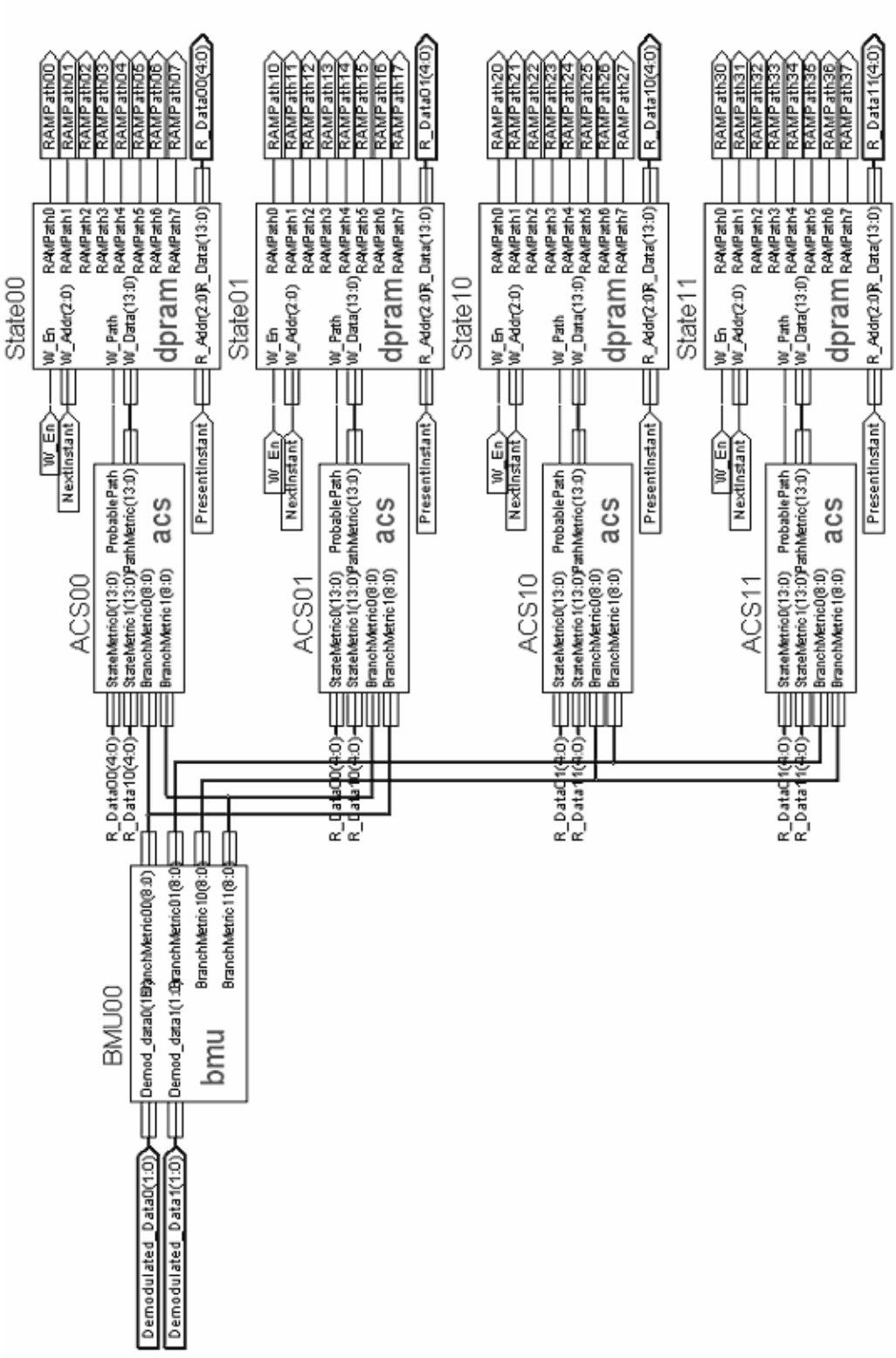


Fig. 4-36 ACSDPRAM Schematic

4.3.5 Soft Decision Viterbi Decoder

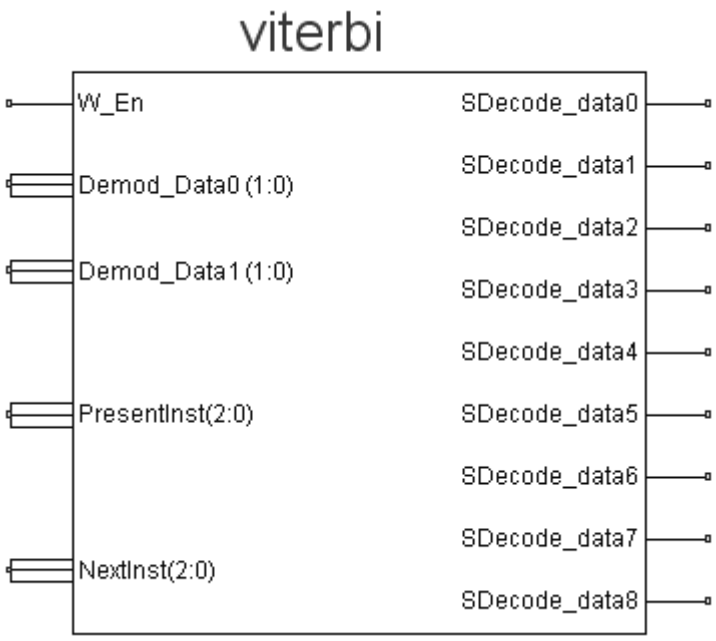


Fig. 4-37 Viterbi Decoder Symbol

The trace-back unit designed in hard decision Viterbi decoder section is used without any modification. So, the decoder operates on trace-back length of 8.

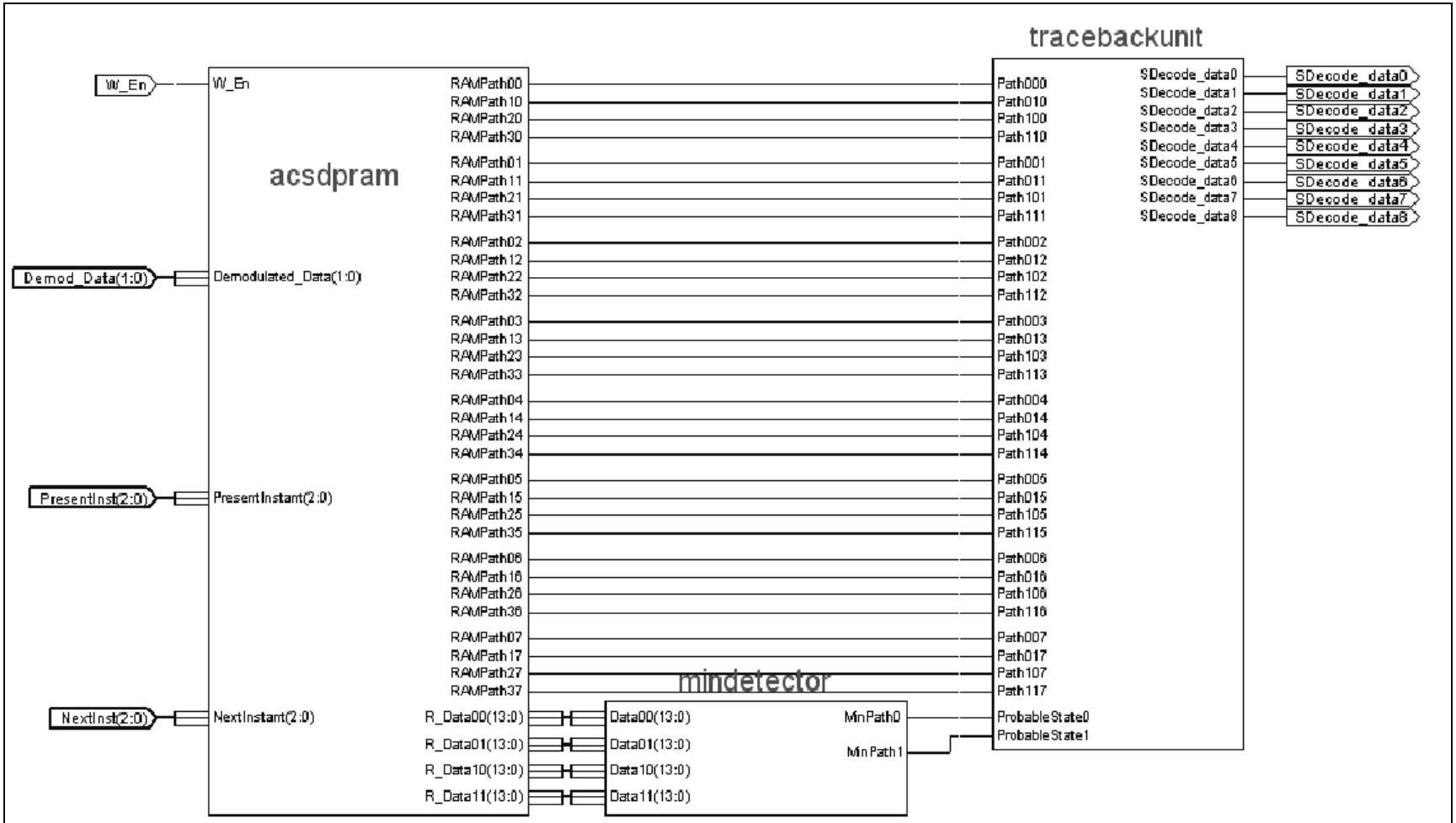


Fig. 4-38 Viterbi Decoder Schematic

CHAPTER 5

RECONFIGURABLE VITERBI DECODER IMPLEMENTATION

5.1 General

In this chapter, two new area efficient reconfigurable Viterbi decoder approaches are proposed. The improvement in these architectures are the new trellis structures which give ability to configure the constraint length by the regular usage of the same small trellis portion in subsequent time instances. The contribution of the same structures usage in all iterations decreases the hardware complexity and the new state organization offered in these approaches provides in place path metric update with only two state metric switching.

5.2 Reconfigurable Viterbi Decoder with Normal and Complemented State Identifiers at Subsequent Iterations

In this part, the first area efficient architecture is described in theoretical manner. The explanation starts with comparison of standard and suggested trellis and resumes with hardware implementation guide lines.

5.2.1 Trellis Structure

The backbone of the Viterbi decoder is the trellis structure and the major usage of the trellis structure is the decision of the more probable state transitions by calculating the branch and state probabilities related to the state transitions and received symbol.

In Fig. 5-1, a standard trellis example is demonstrated for K=4. For K=4 there are totally $2^{(K-1)}$ =8 states and 3 bit is enough to represent them.

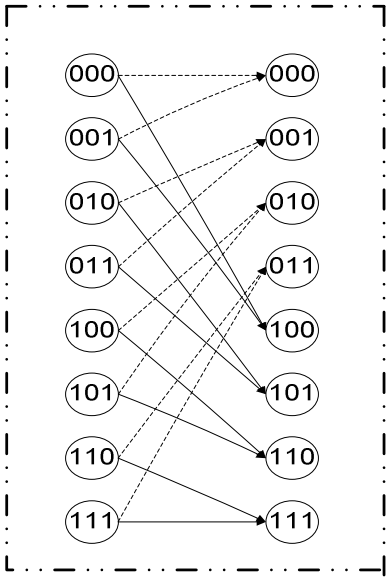


Fig. 5-1 Trellis Structure (K=4)

In Fig. 5-1 the left circles, named as previous, and the right circles, named as next, are the states whose identifiers are determined with the values inside the circles. The arrows show the state transitions corresponding to the assumed data inputs. The dashed lines denote transitions as if logic 0 input have been received and straight lines correspond to the transition for logic 1 input.

The branch and state probabilities have been converted to the branch metrics and the path (state) metric by taking the negative logarithm of base 2. So the probability calculations have been converted to addition instead of multiplication. The branch metrics are determined considering the probability of the next state depending on the previous state and the observed data. The path metrics are calculated cumulatively by the addition of the branch metrics from the beginning of the stream to be decoded.

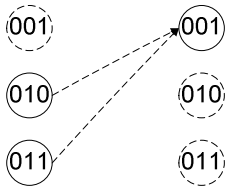


Fig. 5-2 Trellis Calculation

As an example, to obtain the path metric for next state “001”, in the trellis structure the two branches directing to the “001” states should be considered. These are transitions from the “010” state and from the “011” state for logic “0” inputs. Accordingly, (i) the path metric of previous state “010” metric is added with the branch metric of “010” to “001” transition and (ii) previous state “011”metric is added with the branch metric of “011” to “001” transition. After these calculations minimum of (i) and (ii) is selected and stored in as the path metric of the next state “001”. Also the least significant bit of the possible previous state is needed to be stored for the future trace-back operation. The path metric computations and minimum metric selections are managed by Add-Compare-Select Unit (ACS).

After a trellis cycle, calculation of the path metric for every next state, the next state metrics are copied over the previous state metrics. These calculations are recursively carried out up to the depth of trace back. After the receiving of the trace-back lenght stream input, the path metrics are used as a measure to find the most probable state to start with the trace-back.

The bare trellis structure given in Fig. 5-1 is a complicated structure to implement on an IC. For K=4, 8 states are needed and increasing K exponentially increases the number of states, so the number of memories, ACSs, interconnections. Also, the given trellis structure is not in a suitable form to be implemented using reconfigurable blocks. For this reason another kind of representation has been suggested in the literature to generate similar modules. This method, shown in Fig. 5-3, is a new rearrangement of connections in Fig. 5-1. The interchanged positions of the next states formed a trellis model which is composed of substructures called butterfly.

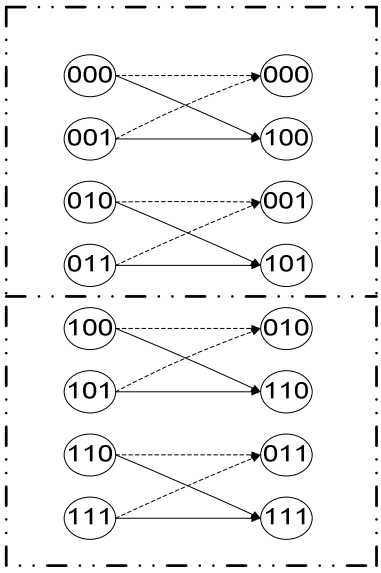


Fig. 5-3 Butterfly Demonstration

Butterfly structures provide regularity in the implementation. For example, for K=6, Viterbi decoder is composed of 2^5=32 states. To generate 32 states, the subtrellis combined with 2 butterflies(B=2, 2x2=4 states) is used eight time for different present and next states. For K=7, the subtrellis is needed to be used 16 times for the generation of 64 states.

In B=2 subtrellis, the 4 memory units correspond the 4 states in subtrellis. For K=6 Viterbi decoder implementation, each memory unit is implemented with 8 address space to represent 8 iterations. In a reconfigurable decoder, the maximum constraint length determines the maximum number of iterations so the total number of address spaces in memory is 2^{K-2}/B.

The butterfly method increased regularity and solved the reconfiguration problem of constraint length with repetition. But butterfly grouping in Fig. 5-4 brought very complicated switching mechanism to link the previous and the next states through ACS units.

To illustrate this problem the 4 state metric memory units were named with alphabets A, B, C, D from top to down in previous states. Then, the next states were named in conjunction with the previous states, introducing the same alphabet to the

same identified states between previous and next states, as demonstrated in Fig. 5-4.

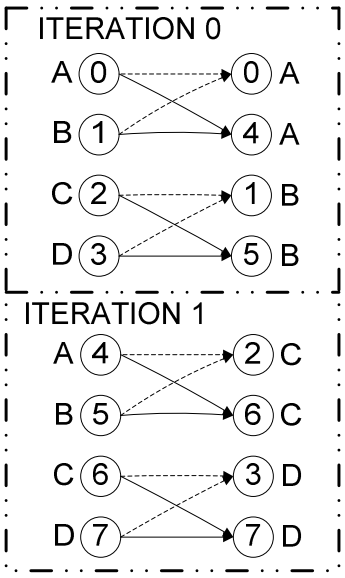


Fig. 5-4 Memory Mapping of Butterfly Structure for B=2

Considering the above iterations, it can be concluded that the next state memory identification contains naming repetitions in the same iteration meaning that the trellis tries to write two different addresses of the same memory unit. This result is an important problem to be solved for an efficient VLSI implementation Viterbi decoder

In this paper a new butterfly structure is suggested to decrease the next state path metric multiplexing and addressing complexity between the two consecutive states and to decrease the complexity of operations.

5.2.2 Complemented Identifier Reconfigurable Foldable Viterbi Decoder

The new implementation is based on a modified foldable structure so called complemented identifier reconfigurable foldable Viterbi decoder. In this new approach memory addressing and multiplexing have been simplified considerably. To construct this model, all iterations are considered subsequently and the first half

numbers of the iterations are taken the same as the original butterfly and the other iterations are rotated vertically. Independent of the constraint length, number of iteration and the order of the iterations, with this method building a Viterbi decoder with only two next state path metric multiplexing has been achieved.

On the other hand controller and traceback unit simplifications are important issues to obtain area efficient integrated circuit. Thus the new trellis model is constructed as explained below.

In the model the consecutive two iterations in subsequent time instances are treated as one group. The iterations are reconstructed and numbered to get the groups given in Fig. 5-5. Note that iterations 0, 2, 4, 6 correspond to the iterations 0, 1, 2, 3 in the ordinary butterfly trellis. Also iterations 1, 3, 5, 7 correspond to the iterations 7, 6, 5, 4 in the ordinary butterfly trellis with scrambled states.

Between first and second iterations of any iteration group the below statements can be concluded.

- All state identifiers in the second iteration of any iteration group are bitwise complemented values of the state identifiers in first iterations. For example for K=6 Viterbi decoder state identifiers are represented with five bits. The present state identified as 5, whose path metric will be stored in memory unit B, in iteration 2 of IG1 is represented as “00101”, where as in the same iteration group, at the same location of the next iteration state identified as 26 exists, with binary notation of “11010” which is the complement of the binary “00101”. Also same approach is valid for next states. For example, next state 17, binary “10001”, of iteration 0 replace with next state 14, binary “01110”, in iteration 1 which is the bitwise complement of 17. The inverted state identifiers decrease size of the state decoder unit in the implementation. Because designing the state decoder for only first iterations and complementing the state identifiers for the second iterations in iteration groups, are enough for implementation of the states identifiers in whole set of iterations.

- For the storage of the path metrics into the internal registers of the memory units the identifiers of the states will be used. But the least significant two bits of the present and the next states are the same for the same position of the first iterations in the iteration groups. So in memory address localization for path metrics the least significant bits of identifiers will not be used. Omitting the least significant two bits, results in a very light weight state decoding for the present states of the first iterations in iteration groups the iteration counter without least significant bit will be connected to the read address of the memory units through xor gates. The other terminals of the xor gates will be connected to the least significant bit of the iteration counter. For the first iterations of the iteration groups the xors behave as an ordinary buffer. For the second iterations of the iteration groups the least significant bit of the counter will be "1" so the address locators will be complemented.

- The next state identifiers of the trellis are the one bit left shifted of the present state identifiers taking the assumed branch input as the most significant bit.

- The branch structure in the consecutive iterations of an iteration group is constructed using a complemented scheme. In an iteration group the dashed branches in first iteration are replaced with the straight lines in the second iteration and straight lines are replaced with the dashed lines. Meaning that in iteration 0, state 0 to state 0 transition is obtained with logic 0 input on the other hand in iteration 1 from state 31 to state 31 transition is achieved by logic 1 input. Note that state identifier "11111" is bitwise complemented version of state identifier "00000".

- Consequently, in the even numbered iteration groups, the next states use the memory units in A, D, B, C order and in the odd ordered iteration groups the next states memory units are used in C, B, D, A order (Fig. 5-5). This regularity results in the usage of the second least significant bit of the

iteration counter directly as the selection control input of the path metric router.

K=6

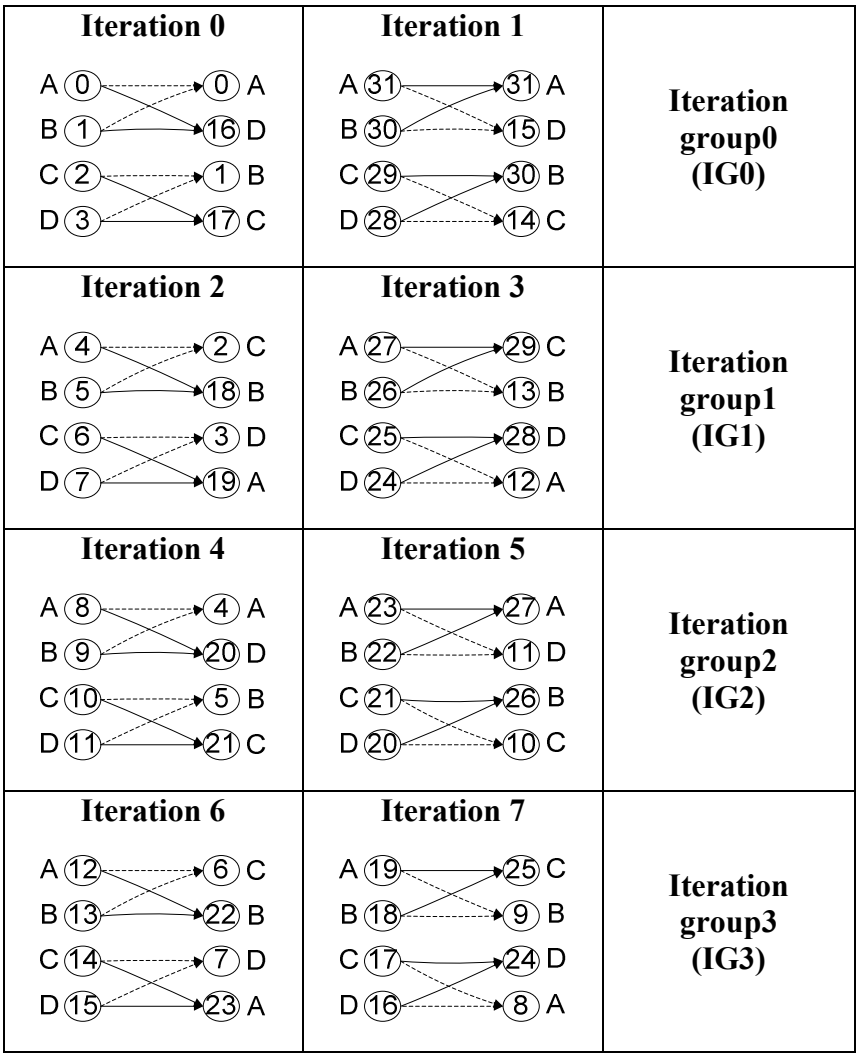


Fig. 5-5 Butterfly Based Projection (Implemented Butterly Group)

Above structure can further be rearranged to get the next states in the same order with the present states as shown in Fig. 5-6.

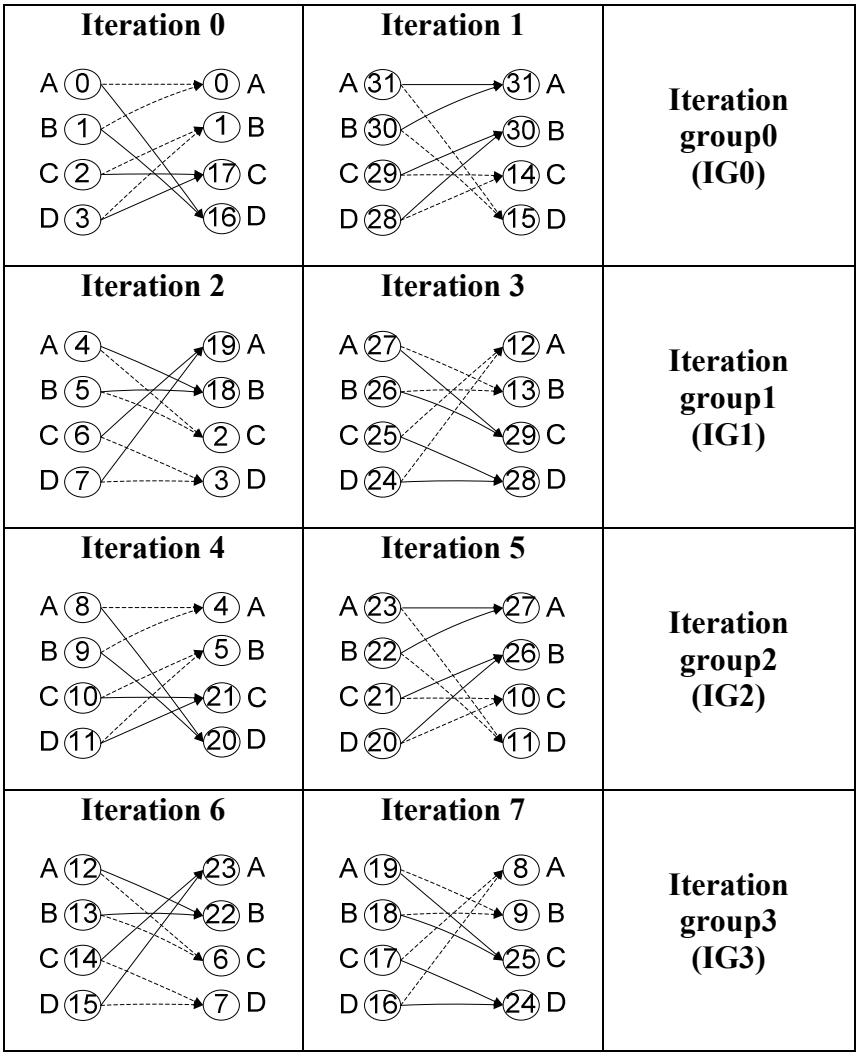


Fig. 5-6 Memory Based Projection (Composition of Butterflies and Path Metric Multiplexing)

- Another consequence is that, for the condition that reveals two iterations (Table 5-1) only one path metric multiplexing mode exists, so there is no need to use next state path metric multiplexers for such cases. For I>2, two path metric multiplexing modes are enough to cover the whole trellis (Fig. 5-5).

Table 5-1 Constraint Length, Number of Butterflies and Iterations Relation

Constraint Length (K)	Number of Butterflies (B)	Number of Iterations (I)
4	2	2
5	4	2
6	8	2
5	2	4
6	4	4
7	8	4
6	2	8
7	4	8
7	2	16

As an example, for K=4, B=2 , two iterations exists as shown in Fig. 5-7.a. As explained above the second iteration has been vertically rotated, and the structure given in Fig. 5-7.b is obtained. Fig. 5-7.c shows the binary representation of the states in Fig. 5-7.b.

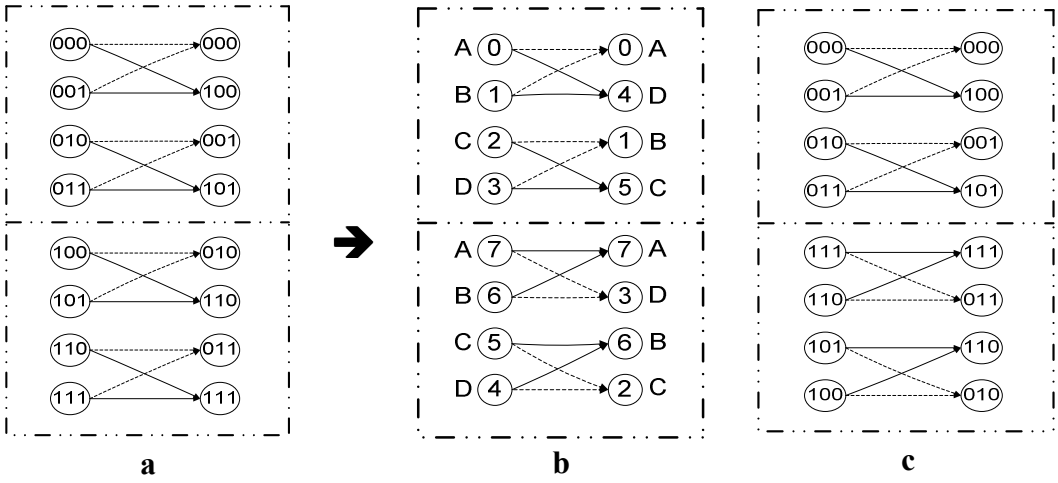


Fig. 5-7 Trellis Structure (K=4)

Rearranged form of Fig. 5-7.b is presented in Fig 9.

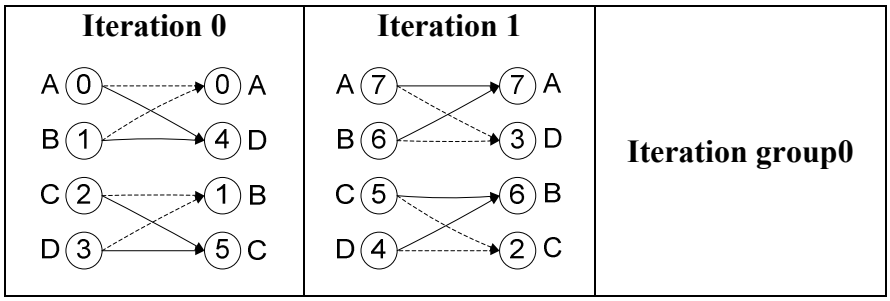


Fig. 5-8 Rearranged Next State Trellis Structure (K=4)

Also for K=5, B=4, two iterations are needed and rotating the second iteration, as explained before, will result in the same scheme (Fig. 5-9). All two iteration trellis with rotated second iteration don't use next state path metric multiplexers. By using this new method, a Viterbi trellis can be implemented with approximately half elements, so as chip area.

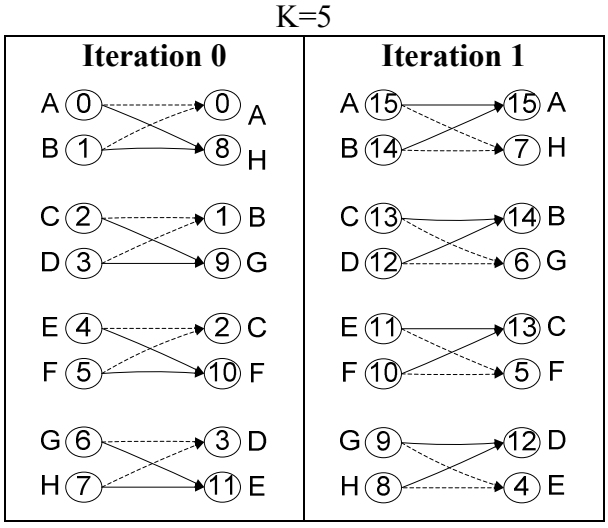


Fig. 5-9 8 State Trellis Structure (K=5)

Because of its complementing nature, the new method can be used for number of iterations greater than one. However, in practice the minimum number of iterations is chosen as 2.

The examples in this paper were given for 2 butterfly subtrellis but the method can be extended to the 2^n number of butterfly subtrellises.

5.2.3 Hardware Structure

In this section hardware implementation of the new complemented identifier reconfigurable foldable Viterbi decoder is discussed for constraint lengths varying from 4 to 7.

In hardware implementation a modified RAM structure is suggested to store path metrics of states. The new modified RAM is called Double Registers Dual Port RAM (DRDPRAM) as given in (Fig. 5-10).

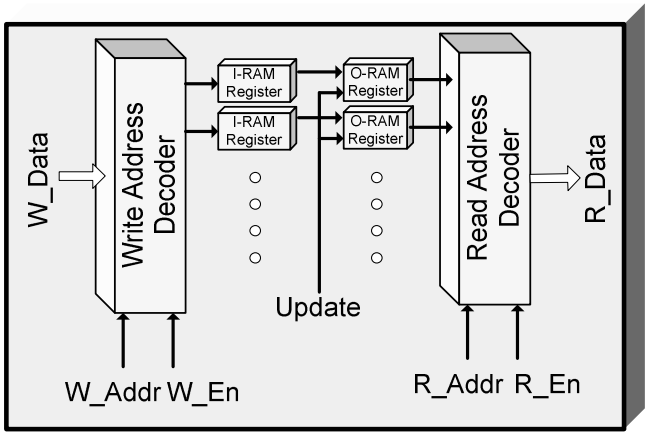


Fig. 5-10 DRDPRAM block diagram

The path metric of the previous states stored in the O-RAM is used to determine the path metric of the next state that will be stored in I-RAM. When all the I-RAM data (i.e., the path metrics of the next states) are calculated, the I-RAM content are transferred to O-RAM by activating “Update”.

To route the path metrics of the next states to the relevant memory unit, two different addressing modes have been utilized. For even indexed iteration groups, addressing shown in Fig. 5-11 is used and for odd indexed iteration groups address multiplexing shown in Fig. 5-12 is used.

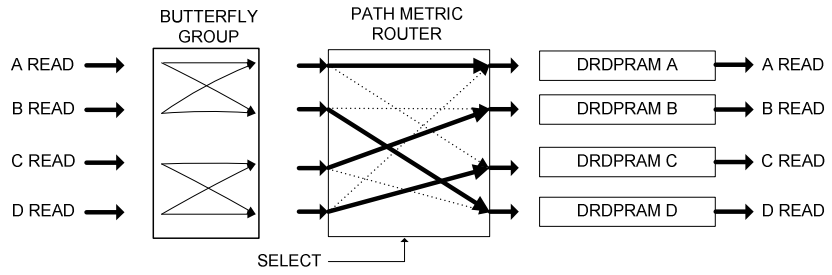


Fig. 5-11 Next State Metric Routing for Even Numbered Iteration Group

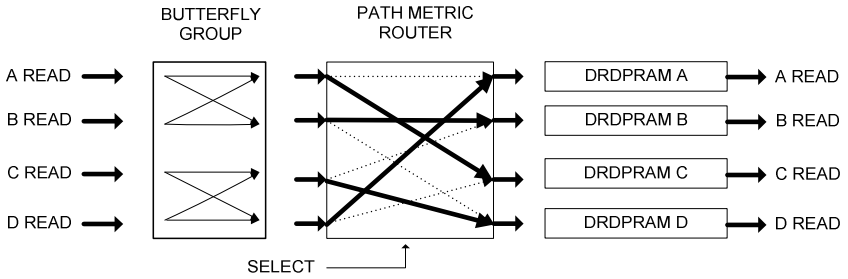


Fig. 5-12 Next State Metric Routing for Odd Numbered Iteration Group

The design should cover the maximum number of states which is reachable with the maximum value of constraint length of 7. For K=7, there are 64 states constituting 16 iterations with B=2 butterfly groups. The 16 iterations results in 16 address locations in which DRDPRAMs store the state metric values. During an iteration, the generation of the relevant present state address for read operation, the next state address for write operation and calculation of the branch metrics should be accomplished. The state decoder designed for this purpose is given in Fig. 5-13.

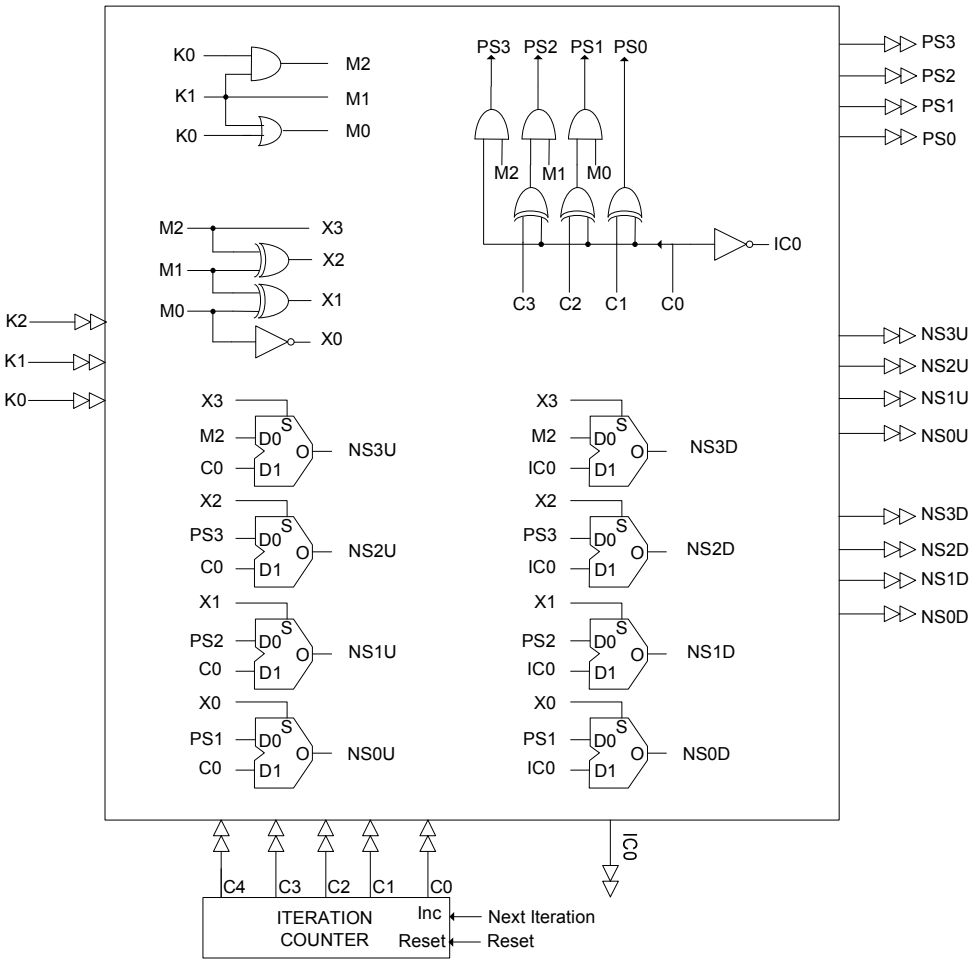


Fig. 5-13 State Decoder

PS3, PS2, PS1 and PS0 are present state identifier bits (used for addressing to read the state metric from the relevant internal O-RAM register of DRDPRAM) and NS3U, NS2U, NS1U, NS0U and NS3D, NS2D, NS1D, NS0D are the bits (used for addressing to write the state metric into the relevant internal I-RAM register of DRDPRAM) of the two next state identifiers which are dependent on the iteration counter and constraint length. The detailed diagrams of the trellis corresponding to the state identifiers are shown in Fig. 5-14 and Fig. 5-15 for the odd and even iteration groups respectively.

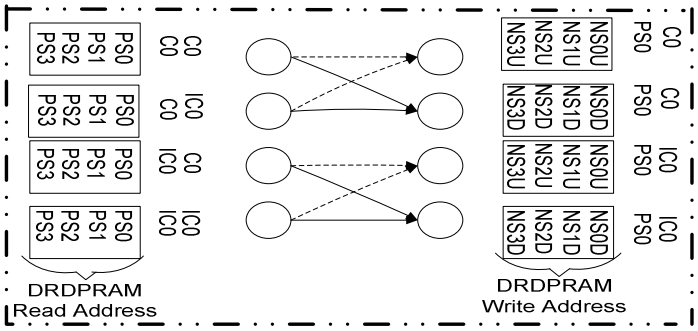


Fig. 5-14 New Trellis Model Hardware Implementation Bits For Odd Iteration Groups

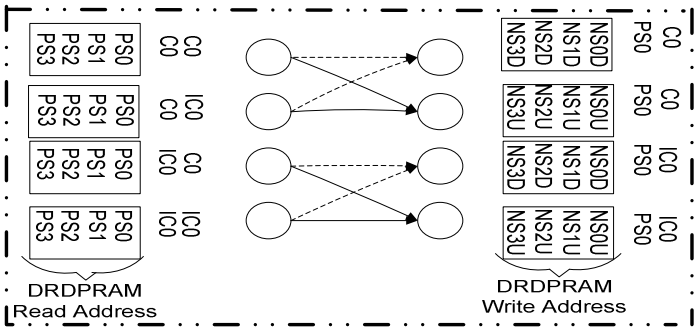


Fig. 5-15 New Trellis Model Hardware Implementation Bits For Even Iteration Groups

The W_Data port is used to store the next state metric into DRDPRAM and R_Data port is used to get the present state metric.

The Fig. 5-16 illustrates detailed block diagram of the decoder. In the schematic, the input bits SU0, SU1, SU2, and SU3 to the Survivor Unit block are generated. This block stores the most probable path information of each nodes guiding to the next states. With this information, the traceback unit can estimate the original message.

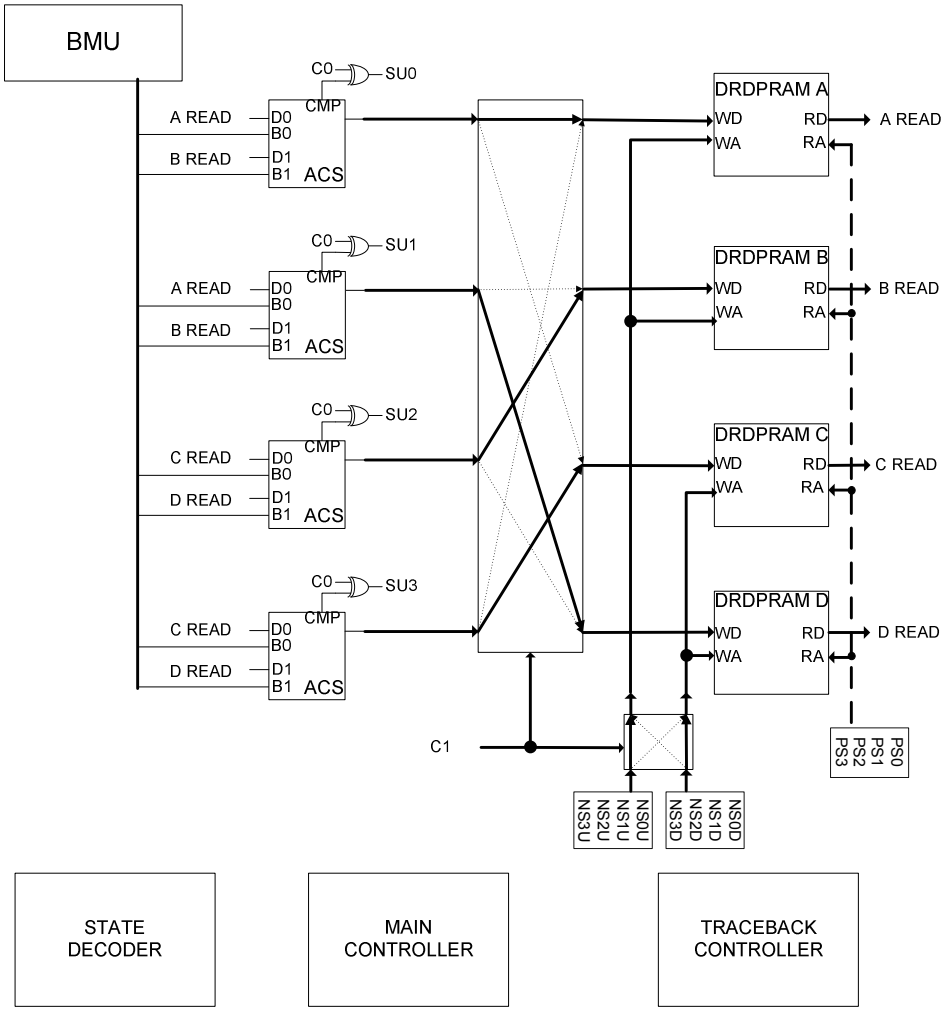


Fig. 5-16 Viterbi Decoder Block Diagram

5.3 Implementation of Reconfigurable Viterbi Decoder with Normal And Complemented State Identifiers At The Same Iteration

Finally, the implementation of second area efficient reconfigurable Viterbi decoder in SystemC is described. The design of the scope gives ability to online configuration of parameters

- constraint length
- code rate
- transition probabilities
- traceback depth
- generator polynomial.

5.3.1 Theory

The design covered in this chapter is another rearrangement of states in the trellis for the implementation of the Viterbi decoder. This new method resembles very much to the first approach in the previous part but the states in the iterations are arranged to contain the normal and complemented identifiers in the same iteration (Fig. 5-17).

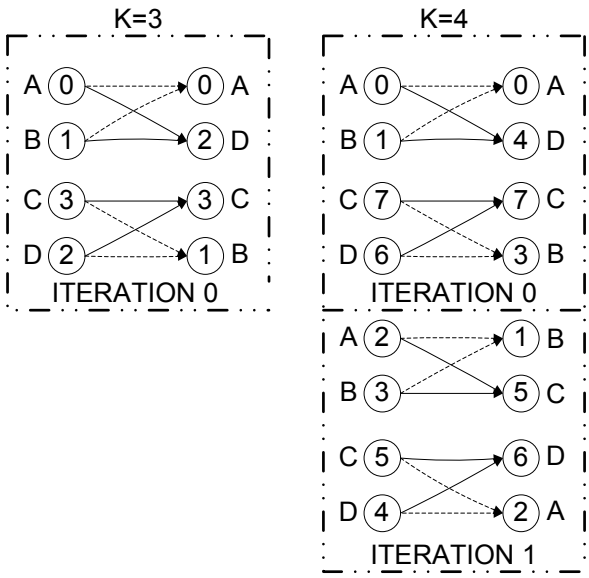


Fig. 5-17 The 4 States Sub-Trellis Structure

In previous part a term called iteration group was introduced as the composition of the two iterations for 4 states subtrellis. For the 8 state subtrellis in this new method two subsequent iterations of an iteration group for 4 states subrellis of previous part can be thought to be used in one iteration, considering second iterations below the first iterations. So the states in the below half of the trellis would be the complemented of the states identifiers of the upper states. For the 8 states sub-trellis structure two examples is given in Fig. 5-18 for K=4 and K=5.

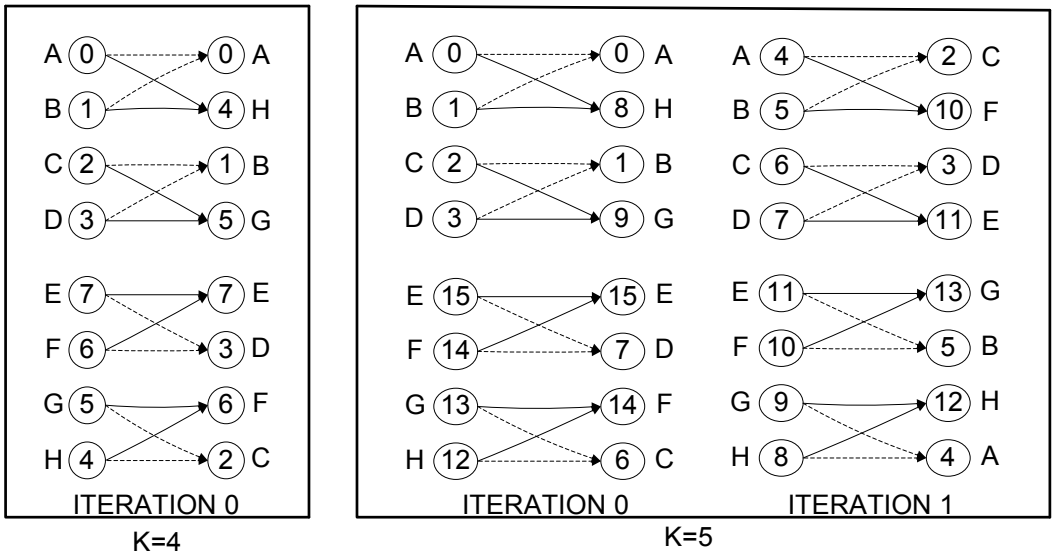


Fig. 5-18 The 8 States Sub-Trellis Structure

In the examples and implementation the 4 and 8 states sub-trellises are introduced for this new approach but increasing the number of states in the sub-trellis with the power of two is also functional.

The first approach was only operative for the number of iterations equal to or greater than two. On the other hand this new approach is functional for even one iteration trellis For example constraint length of 3 is possible in a 4 state trellis. (K=3 in Fig. 5-17 and K=4 in Fig. 5-18).

In the implementation of the reconfigurable Viterbi decoder the 8 states sub trellis structure is used so in the subsequent parts of this chapter the descriptions are based on 8 states sub-trellis structure. To extract the new outcomes for the simplification of the reconfigurable hardware to be implemented, the binary representation of the 8 states sub-trellis structure is given in Fig. 5-19.

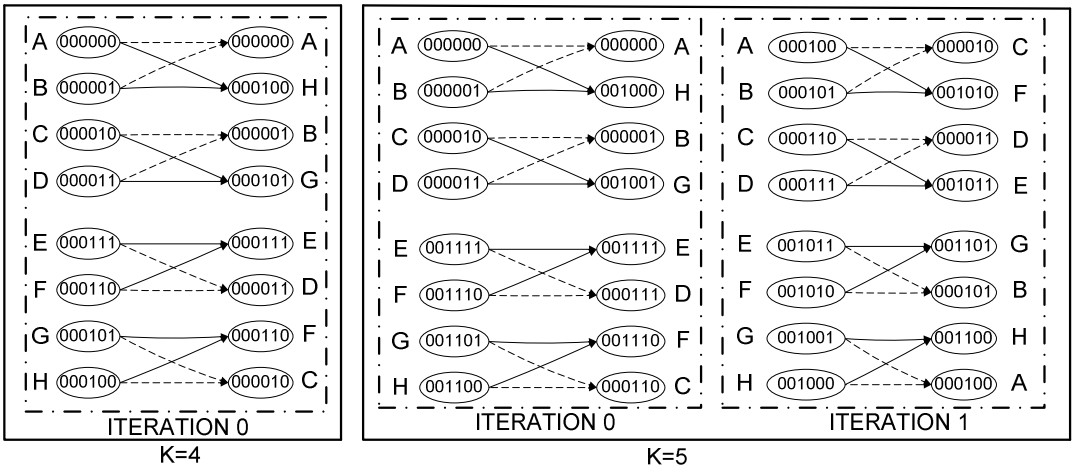


Fig. 5-19 Binary Representation of The New Trellis Structure

In the binary representation, states are identified with 6 bits because the decoder should operate on the constraint length ranging from 4 to 7. For the constraint length of 7, K-1=6 bit is required to identify each states.

In the decoding process same labelled state's metrics are stored in same DRDPRAMs' different registers as usual. For example, for K=5, the DRDPRAM A stores both the path metric of state 0 and state 4 (Fig. 5-19). So there should be a way to address the memory locations related to the states in DRDPRAM units. The 2 least significant bits of present state identifiers are same for the same DRDPRAM in any iteration. So in the addressing operation there is no need to use these two bits. Also another exciting point is that the most meaningful bit* of state identifiers are always 1 in the lower half of the iterations and always 0 in upper states (*The most meaningful bit is located on the most significant position of the minimum number of bits enough to represent all states of a constraint length). This means, direct usage of the most significant 4 bits to address the memory, there will be some blind parts of the memory to where no data would be written. To overcome the memory blind memory problem, there are two methods. These are omitting the most meaningful bit by forcing it to 0 or second way is the complementing all the meaningful bits of the lower half states identifiers** (**Meaningful bits are the minimum number of bits enough to represent all states of a constraint length). The

complementing way will be used in the implementation process. By this method the present state memory location (Read Address of DRDPRAMs) will be same as the iteration number for every DRDPRAMs so the iteration counter output can be directly connected to the read address of the DRDPRAMs. On the next state side the DRDPRAM addresses composed of 2 addresses the first address is connected to the A, B, E, F labeled and the second is connected to the C, D, G, H labelled DRDPRAMs.

Table 5-2 State Table Representation K=5

Iteration Number	DPRAM Identifier	DPRAM Address	Present State Identifiers	DPRAM Read Addresses	Next State Identifiers	DPRAM Write Addresses
Iteration 0	**A**	**0000**	**0000**00	0000	**0000**00	0000
	B	**0000**	**0000**01	0000	**0000**01	0000
	C	**0000**	**0000**10	0000	**0001**10	0001
	D	**0000**	**0000**11	0000	**0001**11	0001
	E	**0000**	**0011**11	0000	**0011**11	0000
	F	**0000**	**0011**10	0000	**0011**10	0000
	G	**0000**	**0011**01	0000	**0010**01	0001
	H	**0000**	**0011**00	0000	**0010**00	0001
Iteration 1	**A**	**0001**	**0001**00	0001	**0001**00	0001
	B	**0001**	**0001**01	0001	**0001**01	0001
	C	**0001**	**0001**10	0001	**0000**10	0000
	D	**0001**	**0001**11	0001	**0000**11	0000
	E	**0001**	**0010**11	0001	**0010**11	0001
	F	**0001**	**0010**10	0001	**0010**10	0001
	G	**0001**	**0010**01	0001	**0011**01	0000
	H	**0001**	**0010**00	0001	**0011**00	0000

The next state memory is arranged on A, H, B, G, E, D, F, C order in iteration 0 and on C, F, D, E, G, B, H, A order in iteration 1. In the hardware the butterfly group is implemented a fixed hardware and a two position state metric multiplexer (path metric router) is designed to direct the state metric to the correct DRDPRAM.

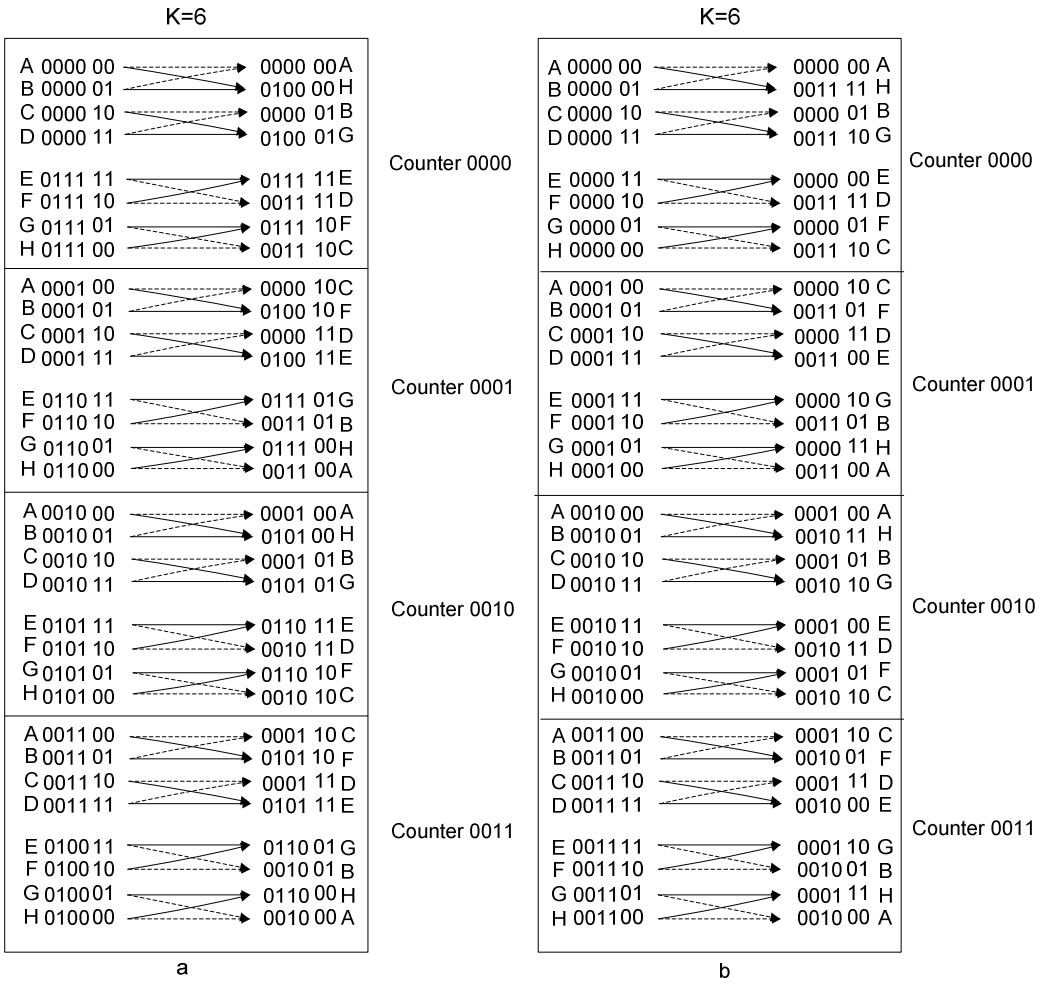


Fig. 5-20 State Inversion (K=6)

The Fig. 5-20.a shows the suggested trellis structure and Fig. 5-20.b shows the memory addressing scheme inside the Viterbi implementation. For K=6 most significant bit of state identifiers is the 5th bit. In the Fig. 5-20.a the most significant bit is always 1 for memories E, F, G, H and 0 for others. The state identifiers of the states stored in E, F, G, H are inverted as shown in Fig. 5-20.b. For register resolution in the DRDPRAMs the least significant 3rd and 4th bits of the state identifiers are used. In Fig. 5-20.b the present state identifiers, neglecting the least significant 2 bits, are the same as iteration counter value so with this approach there is no need to use any additional circuit to address the present states. Also, the state identifiers of the next states contain only two separate values in an iteration, but

also, one of these separate value is one bit shifted of the iteration counter and the other is the inverse of these state identifier bits in the meaningful bits positions. So the design will be so compact in state decoder perspective also.

5.3.2 Implementation

From a top front of view the design can be divided into sub modules as in Fig. 5-21. In the implementation, to increase controllability of hardware development and to avoid errors, the design is divided into submodules also from the early phase of design the submodules are connected to each other and simulated to create bigger modules cumulatively up to the Viterbi decoder.

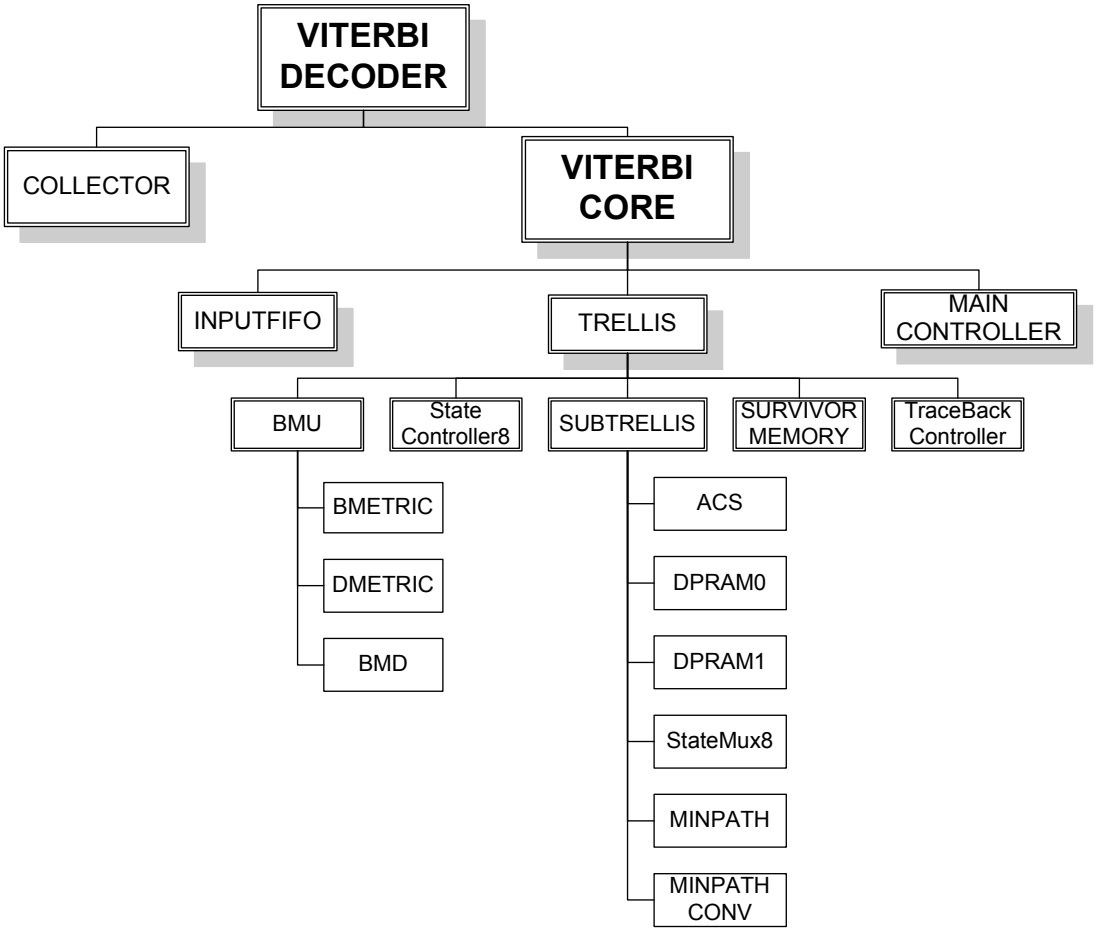


Fig. 5-21 Hierarchical Diagram of the Viterbi Decoder

5.3.2.1 Viterbi Decoder

From the top view, the Viterbi decoder is composed of COLLECTOR module and Viterbi Core module as shown in Fig. 5-21. The COLLECTOR module takes the input symbols, then groups the input symbols according to the code rate and supplies the input symbols to the Viterbi Core. The Viterbi Core (Fig. 5-22) takes the demodulated data groups coming from COLLECTOR and makes decisions of the pattern sent from the transmitter.

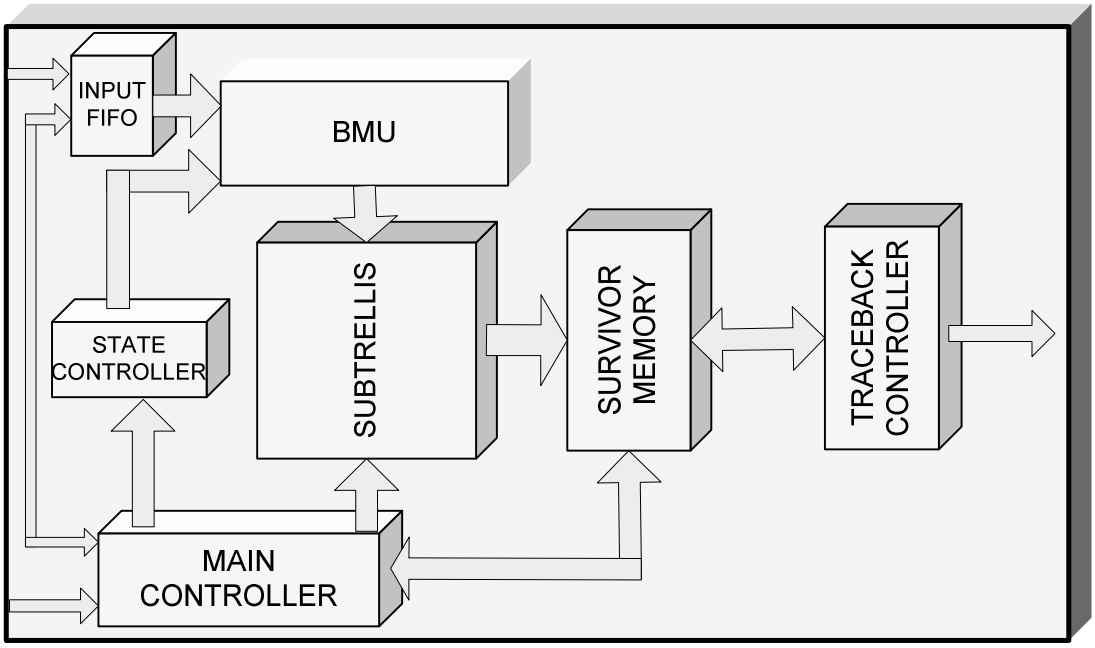


Fig. 5-22 Viterbi Core Block Diagram

In the Viterbi Core three different processes runs concurrently to protect data lose and synchronization problems. These processes are controlled by Input FIFO, Main Controller and Traceback Unit. The Input FIFO is the first stage of the core and accumulates input data. Then FIFO transmits the received data to the trellis section when the trellis section is ready to calculate new data. The second process takes place in main controller section which controls State Controller, SUBTRELLIS BMU and SURVIVOR MEMORY modules for the trellis operations. The last

process is controlled in TRACEBACK CONTROLLER to estimate the transmitted message.

In the subsequent parts of this chapter the functionality of the modules are explained in details.

5.3.2.2 Collector

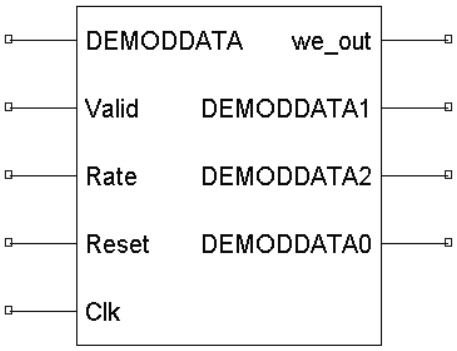


Fig. 5-23 COLLECTOR Symbol

The main functionality of collector unit is to give ability to configure the code rate of Viterbi decoder to select 1/2 or 1/3 basic code rates.

When the reset signal is activated, the COLLECTOR module points to the DEMODDATA0 internal register for write purpose and the content of all three DEMODDATA0, DEMODDATA1, and DEMODDATA2 registers are reset to all zero. This module takes soft decision input from DEMODDATA port on the positive edge of Clk signal if the valid signal is active. Then writes this value to the DEMODDATA0 register and points to DEMODDATA1 register, with the next valid signal the COLLECTOR writes the new content of DEMODDATA port to DEMODDATA1 register. At this point the Rate (Code Rate) port is checked whether 1/2 or 1/3. If the rate signal is in 1/2 value (logic 0) the we_out port is activated to inform the Input FIFO of the Viterbi core to poll the one set of demodulated data. But if the rate is at 1/3 value (logic 1) the collector waits for the

new valid signal to write the new DEMODDATA pattern to the DEMODDATA3 register and we_out is activated.

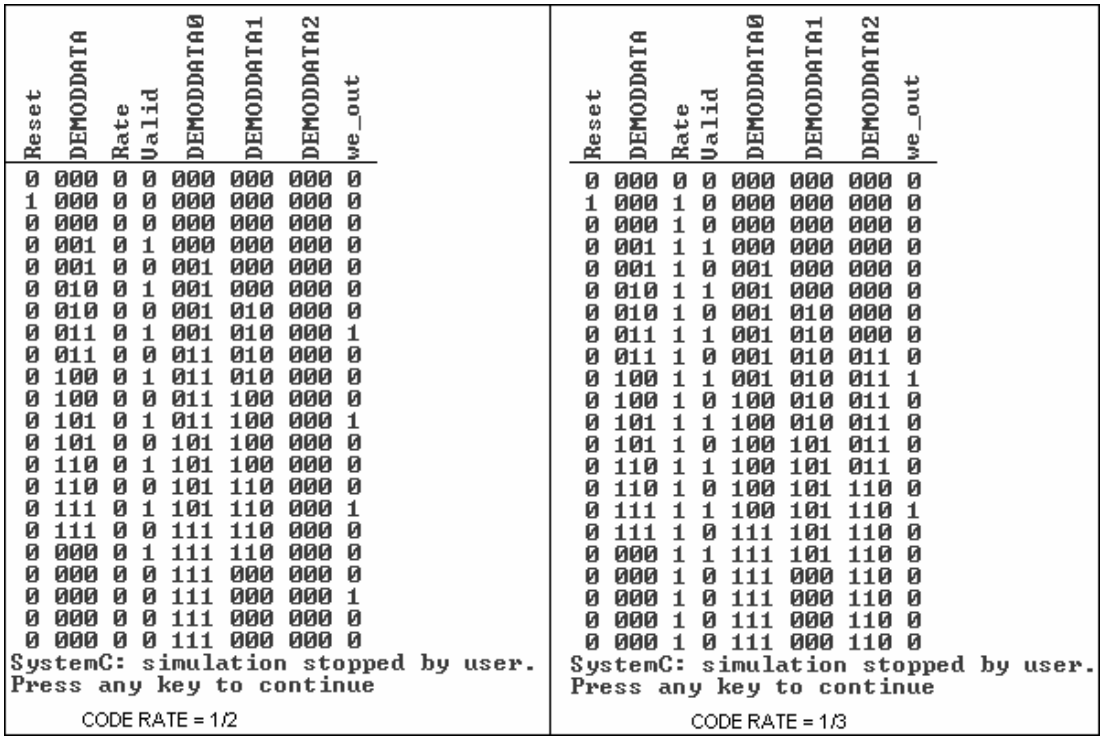


Fig. 5-24 COLLECTOR Simulation

5.3.2.3 Input FIFO

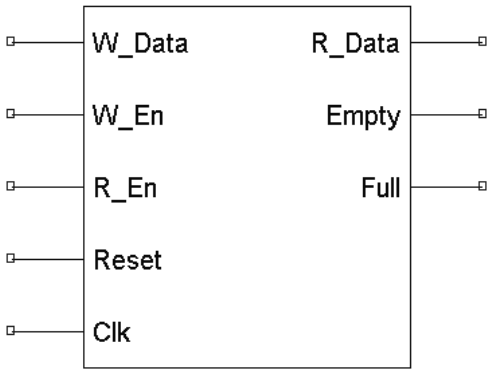


Fig. 5-25 Input FIFO Symbol

The first stage in the Viterbi core is the Input FIFO which is used to guarantee the synchronization between the internal core of the decoder and the COLLECTOR side containing the receiver.

The Input FIFO is directly located in front of the COLLECTOR and inputs the three bits wide three demodulated data (DEMODDATA0, DEMODDATA1, DEMODDATA2) merged at the 9 bits wide port W_Data. The content of the W_Data port is written in the circular memory registers pointed by internal register called W_Addr whenever the W_En port is high at the positive edge of the Clk signal. However, before writing the new data to the internal registers, the user should be aware of the available capacity of the memory if there is room for the new data. A port named Full serves for this purpose. The logic 1 state Full port reflects that the memory is full and the new data will be overridden on the other data by mistake. On the contrary, the logic 0 Full port states that there is available registers for new demodulated patterns. At the other side of the input FIFO the main controller of the core checks the empty signal of the FIFO. The not empty condition states that there is at least one unprocessed demodulated pattern coming from the COLLECTOR. At this time, the main controller strobes a read enable signal to get the data from the internal registers pointed by the internal R_Addr registers to the R_Data port of the FIFO. The R_Data port of the FIFO is connected

to the BMU to generate the branch metrics for the state transitions in iterative trellis.

As stated before, the memory registers of the FIFO are located in a circlar manner. With help of to memory pointers called R_Addr and W_Addr the contents of the registers are reached. The read address pointer indicates the address of registers to be read in the next read enable stroption and increases by one after every read operations. The write address pointer on the other hand indicates the address of the register to where the new demodulated pattern will be written with write enable signal and also this register is increased by one after every write operations. The Reset signal is used to initialize the R_Addr and W_Addr registers to address 0. The Empty and Full signals are also operates on the simple subtraction operation. In the FIFO the W_Addr is subtracted from the R_Addr if the result is 1 the FIFO strobes Full signal else if the result equals to 0 the FIFO strobes Empty signal.

The simulation of the FIFO is given in Fig. 5-26.

W_Data	R_Data	W_En	R_En	Empty	Full
000000	000000	0	0	1	0
000000	000000	0	0	1	0
001111	000000	0	0	1	0
001111	000000	1	0	1	0
001110	000000	0	0	0	0
001110	000000	1	0	0	0
001101	000000	0	0	0	0
001101	000000	1	0	0	0
001100	000000	0	0	0	0
001100	000000	1	0	0	0
001011	000000	0	0	0	0
001011	000000	1	0	0	0
001010	000000	0	0	0	0
001010	000000	1	0	0	0
001001	000000	0	0	0	0
001001	000000	1	0	0	0
001000	000000	0	0	0	0
001000	000000	1	0	0	0
000111	000000	0	0	0	0
000111	000000	1	0	0	0
000110	000000	0	0	0	0
000110	000000	1	0	0	0
000101	000000	0	0	0	0
000101	000000	1	0	0	0
000100	000000	0	0	0	0
000100	000000	1	0	0	0
000011	000000	0	0	0	0
000011	000000	1	0	0	0
000010	000000	0	0	0	0
000010	000000	1	0	0	0
000001	000000	0	0	0	0
000001	000000	1	0	0	0
000001	000000	1	1	0	1
000001	001111	1	0	0	0
000001	001111	0	0	0	1
000001	001111	1	0	0	1
000001	001111	0	0	0	1
000001	001111	0	1	0	1
000001	001110	0	0	0	0
000001	001110	0	1	0	0
000001	001101	0	0	0	0
000001	001101	0	1	0	0
000001	001100	0	0	0	0
000001	001100	0	1	0	0
000001	001011	0	0	0	0
000001	001011	0	1	0	0
000001	001010	0	0	0	0
000001	001010	0	1	0	0
000001	001001	0	0	0	0
000001	001001	0	1	0	0
000001	001000	0	0	0	0
000001	001000	0	1	0	0
000001	000111	0	0	0	0
000001	000111	0	1	0	0
000001	000110	0	0	0	0
000001	000110	0	1	0	0
000001	000101	0	0	0	0
000001	000101	0	1	0	0
000001	000100	0	0	0	0
000001	000100	0	1	0	0
000001	000011	0	0	0	0
000001	000011	0	1	0	0
001110	000010	0	1	0	0
001110	000001	1	1	0	0
001110	000001	0	1	0	0
001110	001110	0	0	1	0
001110	001110	0	1	1	0
001110	001110	0	0	1	0
001110	001110	0	1	1	0
001110	001110	0	0	1	0
001110	001110	0	1	1	0
001110	001110	0	0	1	0
001110	001110	0	1	1	0

Fig. 5-26 Input FIFO Simulation

5.3.2.4 State Controller (State Decoder)

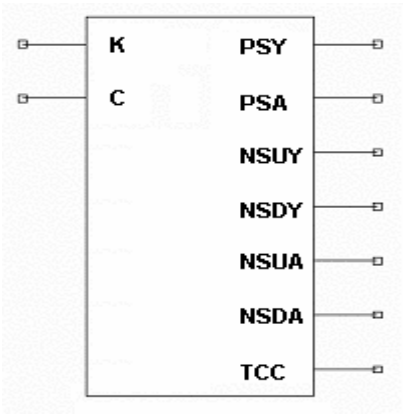


Fig. 5-27 STATE CONTROLLER Symbol

State Controller is a combinational logic circuitry and generates the state identifiers of the trellis in each iteration (Fig. 5-28).

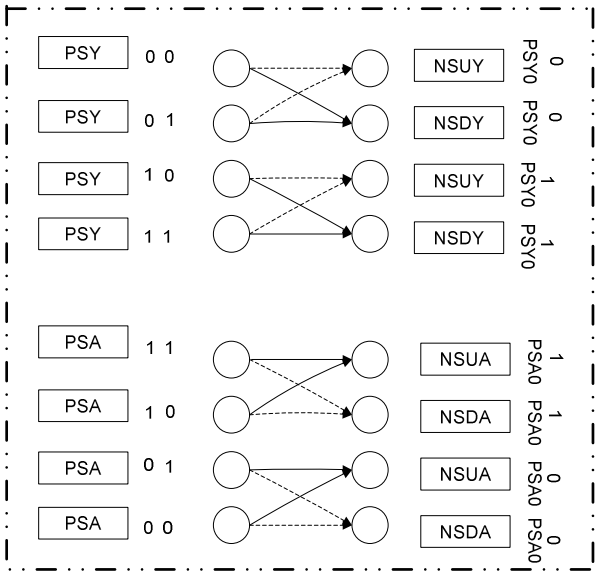


Fig. 5-28 State Identifiers in Iteration

For this reason, the State Controller takes Constraint length from input port of the Viterbi decoder and the iteration counter from main controller, then operates on the truth table as in Table 5-3.

Table 5-3 State Controller Truth Table

K	C	PSY	PSA	NSUY	NSDY	NSUA	NSDA
100	0000	0000	0000	0000	0001	0001	0000
101	0000	0000	0001	0000	0010	0010	0000
101	0001	0001	0000	0000	0010	0010	0000
110	0000	0000	0011	0000	0100	0101	0001
110	0001	0001	0010	0000	0100	0101	0001
110	0010	0010	0001	0001	0101	0100	0000
110	0011	0011	0000	0001	0101	0100	0000
111	0000	0000	0111	0000	1000	1011	0011
111	0001	0001	0110	0000	1000	1011	0011
111	0010	0010	0101	0001	1001	1010	0010
111	0011	0011	0100	0001	1001	1010	0010
111	0100	0100	0011	0010	1010	1001	0001
111	0101	0101	0010	0010	1010	1001	0001
111	0110	0110	0001	0011	1011	1000	0000
111	0111	0111	0000	0011	1011	1000	0000

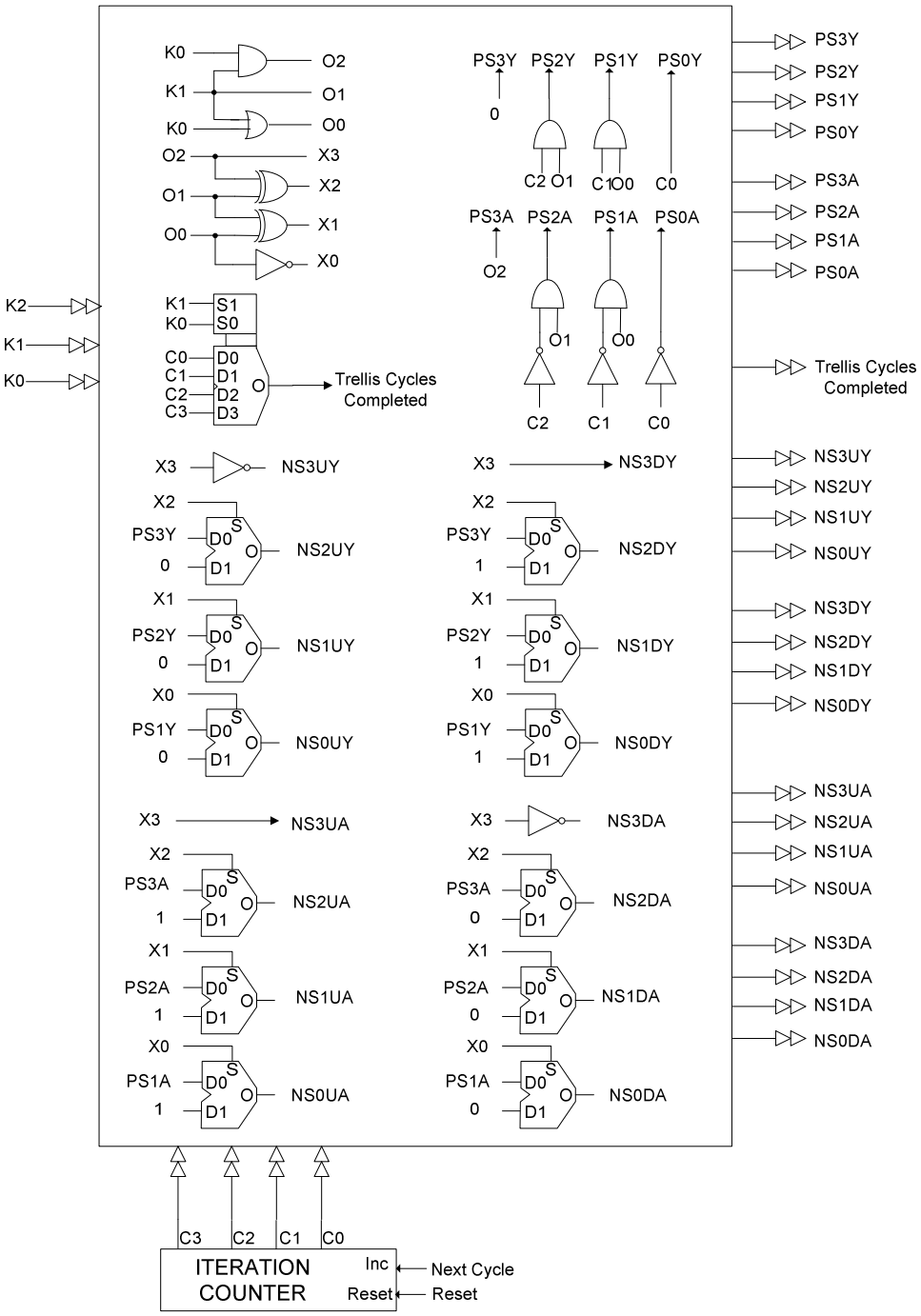


Fig. 5-29 State Controller Schematic

```
        K       C       PSY     PSA     NSUY    NSDY    NSUA    NSDA
        100     0000    0000    0001    0000    0001    0001    0000
        100     0000    0000    0001    0000    0001    0001    0000
        101     0000    0000    0011    0000    0010    0011    0001
        101     0001    0001    0010    0000    0010    0011    0001
        110     0000    0000    0111    0000    0100    0111    0011
        110     0001    0001    0110    0000    0100    0111    0011
        110     0010    0010    0101    0001    0101    0110    0010
        110     0011    0011    0100    0001    0101    0110    0010
        111     0000    0000    1111    0000    1000    1111    0111
        111     0001    0001    1110    0000    1000    1111    0111
        111     0010    0010    1101    0001    1001    1110    0110
        111     0011    0011    1100    0001    1001    1110    0110
        111     0100    0100    1011    0010    1010    1101    0101
        111     0101    0101    1010    0010    1010    1101    0101
        111     0110    0110    1001    0011    1011    1100    0100
        111     0111    0111    1000    0011    1011    1100    0100
        111     1000    1000    0111    0100    1100    1011    0011
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-30 State Controller Simulation

5.3.2.5 BMU

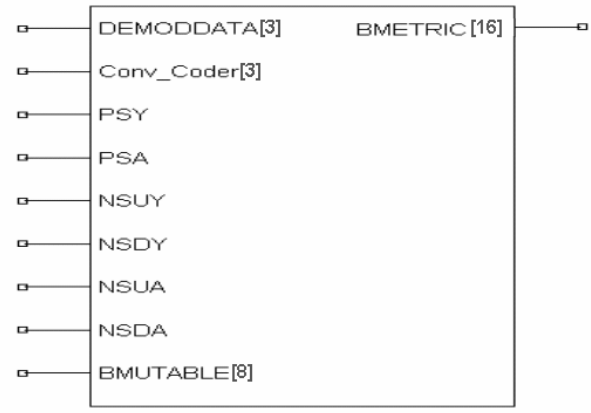


Fig. 5-31 BMU Symbol

Branch Metric Unit gets three demodulated data (DEMODDATA[3]), generator polynomials (Conv_Coder[3]), Quantization Probability Table (BMUTABLE[8]) and Present-Next State Identifiers to create branch metric values of the transitions in the iterations (BMETRIC[16]).

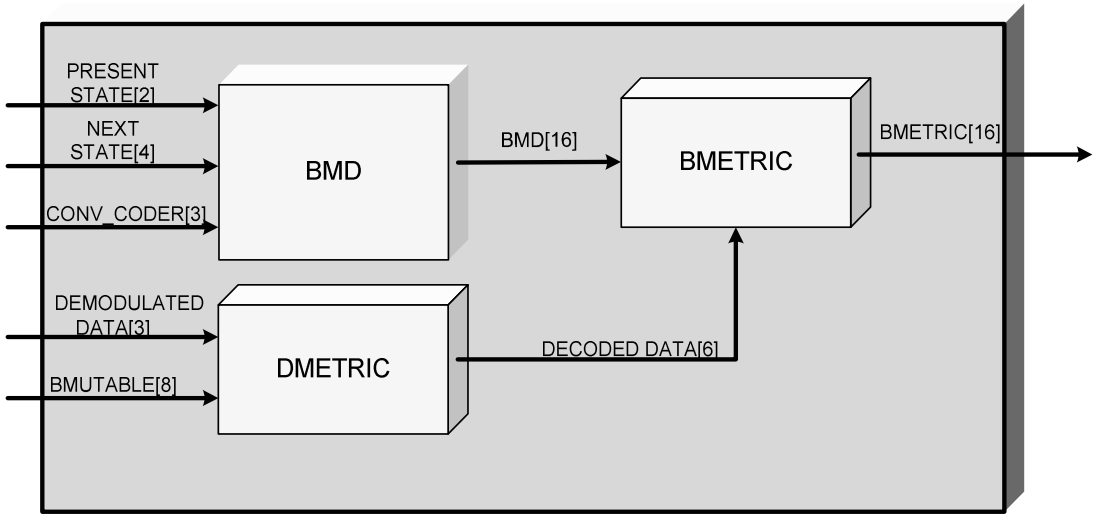


Fig. 5-32 BMU Block Diagram

The BMU unit is composed of three modules named BMD, DMERIC and BMETRIC.

The BMD unit calculates the expected encoder outputs related to the transitions in the iteration by using the generator polynomial and present-next state identifiers of transitions came from the state controller.

The eight BMUTABLE are the soft decision probability constants inputted from the configurable inputs of Viterbi decoder (Fig. 5-33).

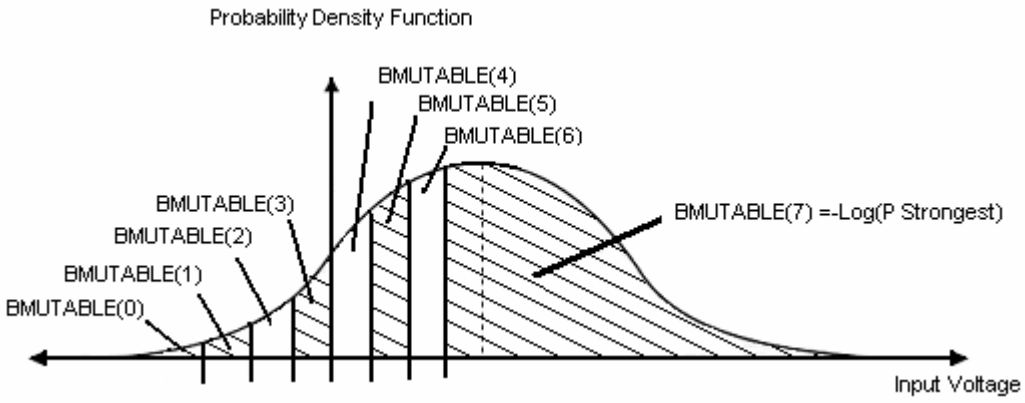


Fig. 5-33 Soft Decision Probabilities

The DMETRIC supplies six probability metrics using demodulated data and BMUTABLE. The DMETRICs multiplexes the soft decision probabilities to the DECODED DATA ports for expected encoder bits of 0 and 1. The soft decision probabilities are symmetrical for logic 0 and 1 expected encoder outputs. The demodulated data of 7 is the most probable to be sent by logic 1 and least probable to be sent by 0. So for the logic 1 expected data the BMUTABLE[7] is sent to output port on the other hand BMUTABLE[0] is sent for the logic 0 expected encoder output. Caring this fact, all six probable Soft Decision Probabilities are calculated by the equations below.

DECODED DATA0=BMUTABLES[DEMODDATA0];
DECODED DATA1=BMUTABLES[complement(DEMODDATA0)];
DECODED DATA2=BMUTABLES[DEMODDATA1];
DECODED DATA3=BMUTABLES[complement(DEMODDATA1)];
DECODED DATA4=BMUTABLES[DEMODDATA2];
DECODED DATA5=BMUTABLES[complement(DEMODDATA2)];

The BMETRIC module is the final stage of BMU module and by the BMD (expected encoder outputs of the branches) and DECODED DATA for the SUBTRELLIS processes, BMETRIC calculates the branch metrics.

5.3.2.6 SubTrellis

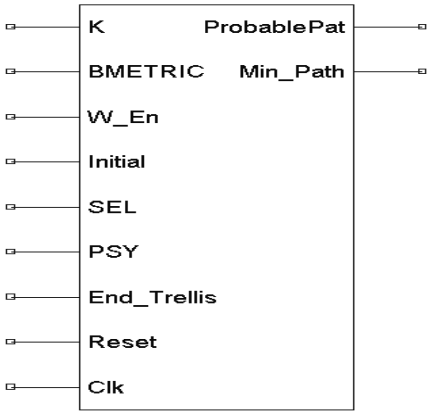


Fig. 5-34 SUBTRELLIS Symbol

Subtrellis module is a connection module to create the sub-trellis structure.

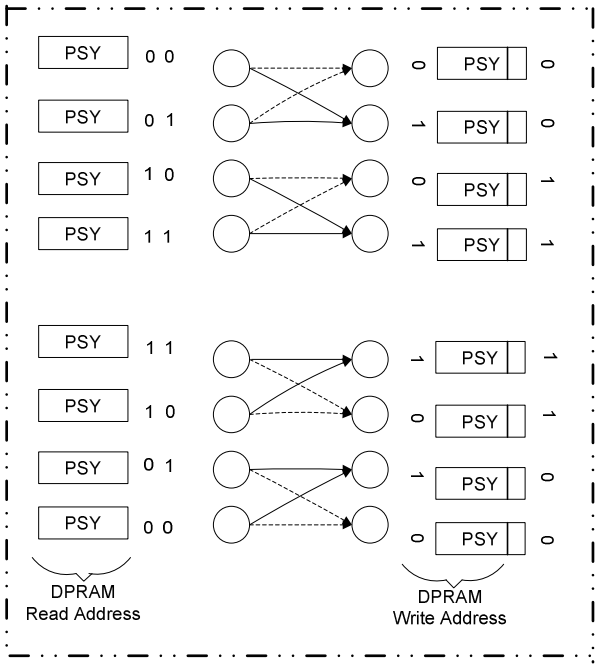


Fig. 5-35 Sub-Trellis

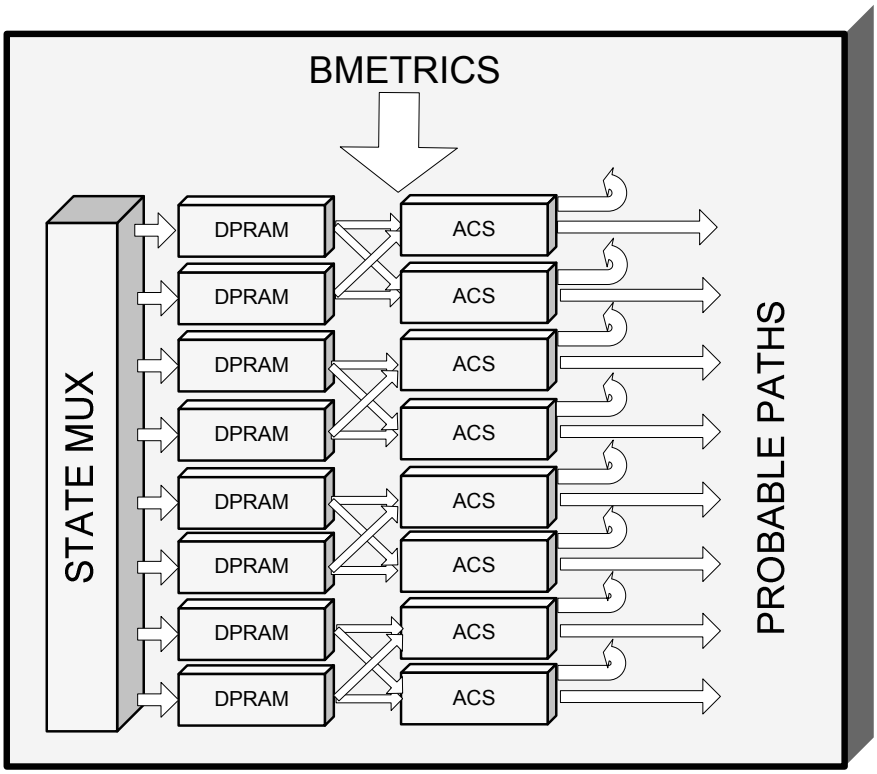


Fig. 5-36 SUBTRELLIS Block Diagram

5.3.2.7 STATEMUX

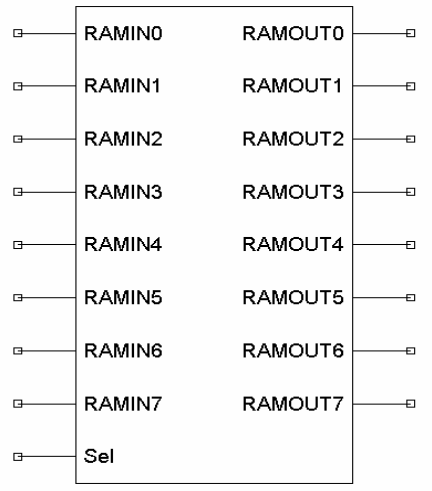


Fig. 5-37 STATEMUX Symbol

The method in this chapter offers the complexity reduction in state metrics multiplexing. With this method only two way multiplexing of the state metrics are enough for the realization of all trellises. The STATEMUX module is the component which multiplexes the state metrics by the value of Sel input.

5.3.2.8 ACS

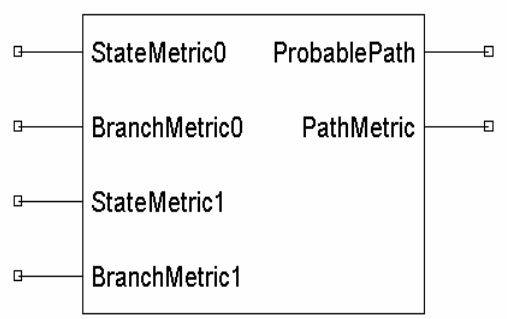


Fig. 5-38 ACS Symbol

The ACS unit is used same as in the soft decision Viterbi decoder. The ACS is responsible to add the branch metrics with the state metrics for the transitions and the selection of the better paths between two competing transitions.

```
Add Compare Select Unit Implementation with SystemC
        SM0     BM0     SM1     BM1     PM      PP
        0       0       0       0       0       0
        0       0       0       0       0       0
        3       1       4       2       4       0
        12      2       11      1       12      1
        15      1       14      0       14      1
        14      0       15      2       14      0
        24      0       25      1       24      0
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-39 ACS Simulation

5.3.2.9 DRDPRAM

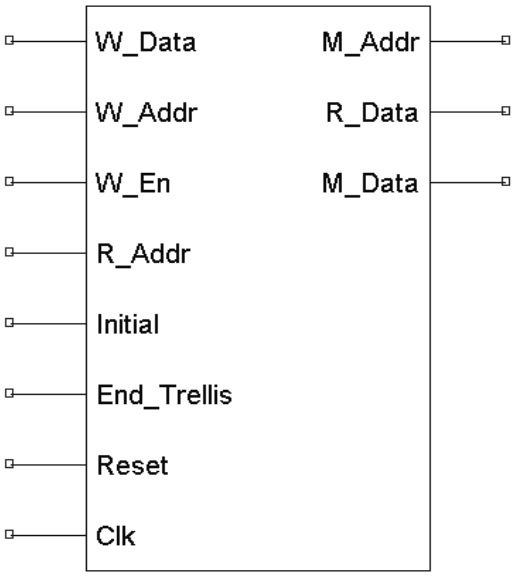


Fig. 5-40 DPRAM Symbol

The DPRAMs in soft decision Viterbi decoder are modified for this implementation. For this purpose, two ports are added to the DPRAM to ease the decision of the starting state for best path in the traceback. These ports indicates the address and value of the minimum valued memory register. Apart from this difference and the number of bits in the memory registers, the DPRAM operates same as the ones in the first part of this chapter.

Also in this chapter, two kinds of DPRAMs are used to supply the zero metric to the initial state identified with zero and higher metric for the undefined stated in the first trellis iterations. The DPRAM doesn't have intelligence for the decision of the initial trellis cycle, thus, a port named "initial" triggered by maincontroller is used for initialization of path metrics.

5.3.2.10 Survivor Memory

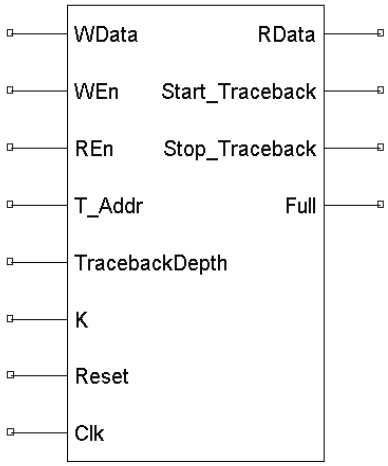


Fig. 5-41 SURVIVOR MEMORY Symbol

Survivor Memory is the interface between trellis and traceback controller and synchronizes the operations of the trellis and traceback unit. The Survivor Memory stores the surviving branches identifier of the competing branches decided by each ACS.

The Survivor Memory is designed as a LIFO (Last In First Out) memory. In the LIFO two pointers R_Addr and W_Addr are designed to point the internal memory registers. The read and write directions are designed to identify the W_Addr and R_Addr alteration direction in either increasing or decreasing. The survivor memory gets the traceback depth length information from Viterbi decoder ports.

To show the operation of the Survivor Memory an example in Fig. 5-42 is constructed for constraint length of 5 and traceback depth of 3 Viterbi decoder.

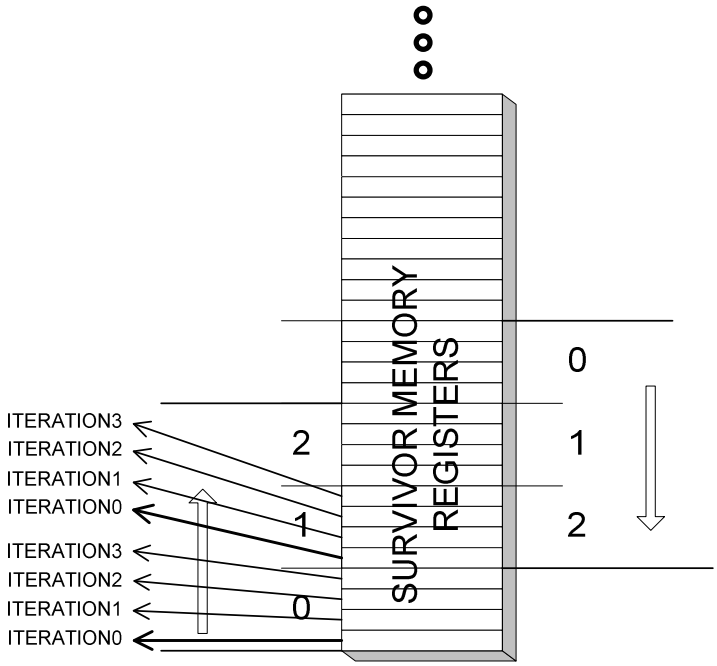
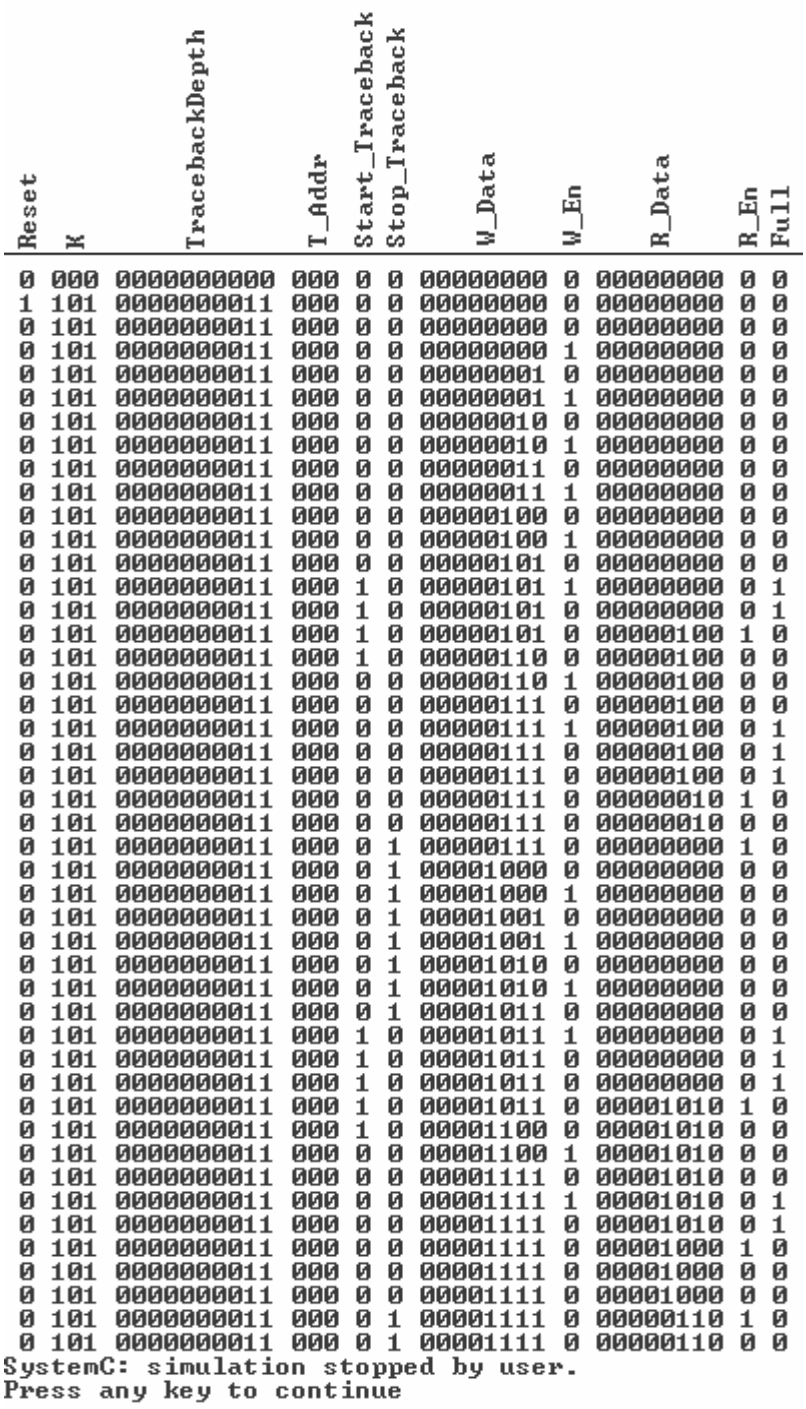


Fig. 5-42 SURVIVOR MEMORY Example

Initially the read address is pointed to the iteration 0 of the trellis 3, the W_Addr is pointed to the address 0 which is equivalent to the iteration 0 of trellis 0. With the W_En signal the surviving branch information of states in iteration 0 are writen to the memory and the W_Addr will point to the address 1 (iteration 1 of the trellis 0). On every W_En the surviving branches are written to the memory and W_Addr increases by one until the address equals to the iteration 0 of trellis 3. When the W_Addr is pointed to the iteration 0 trellis 3 which is equal to the R_Addr the survivor memory outputs the Full signal to inform the main controller not to write any information to the full survivor memory. Also survivor memory strobes Start_Traceback signal to activate the traceback controller for trace-back operation. For the traceback, the traceback controller aims to get the correct transition identifier among 32 states of trellis. So the T_Addr is used to give offset to the read address to locate the related iteration containing the correct state. Then the R_En is applied to read the content of the Survivor Memory then the R_Addr is decreased by four. At this stage the traceback controller releases Full signal because the R_Addr is different from the W_Addr.

Then, the main controller resumes writing the new surviving paths with W_En signal and W_Addr decreases by one whenever the W_Addr not equals to R_Addr.



Reset	K	TracebackDepth	T_Addr	Start_Traceback	Stop_Traceback	W_Data	W_En	R_Data	R_En	Full
0	000	00000000000	000	0	0	00000000	0	00000000	0	0
1	101	00000000011	000	0	0	00000000	0	00000000	0	0
0	101	00000000011	000	0	0	00000000	0	00000000	0	0
0	101	00000000011	000	0	0	00000000	1	00000000	0	0
0	101	00000000011	000	0	0	00000001	0	00000000	0	0
0	101	00000000011	000	0	0	00000001	1	00000000	0	0
0	101	00000000011	000	0	0	00000010	0	00000000	0	0
0	101	00000000011	000	0	0	00000010	1	00000000	0	0
0	101	00000000011	000	0	0	00000011	0	00000000	0	0
0	101	00000000011	000	0	0	00000011	1	00000000	0	0
0	101	00000000011	000	0	0	00000100	0	00000000	0	0
0	101	00000000011	000	0	0	00000100	1	00000000	0	0
0	101	00000000011	000	0	0	00000101	0	00000000	0	0
0	101	00000000011	000	1	0	00000101	1	00000000	0	1
0	101	00000000011	000	1	0	00000101	0	00000000	0	1
0	101	00000000011	000	1	0	00000101	0	00000100	1	0
0	101	00000000011	000	1	0	00000110	0	00000100	0	0
0	101	00000000011	000	0	0	00000110	1	00000100	0	0
0	101	00000000011	000	0	0	00000111	0	00000100	0	0
0	101	00000000011	000	0	0	00000111	1	00000100	0	1
0	101	00000000011	000	0	0	00000111	0	00000100	0	1
0	101	00000000011	000	0	0	00000111	0	00000100	0	1
0	101	00000000011	000	0	0	00000111	0	00000010	1	0
0	101	00000000011	000	0	0	00000111	0	00000010	0	0
0	101	00000000011	000	0	1	00000111	0	00000000	1	0
0	101	00000000011	000	0	1	00001000	0	00000000	0	0
0	101	00000000011	000	0	1	00001000	1	00000000	0	0
0	101	00000000011	000	0	1	00001001	0	00000000	0	0
0	101	00000000011	000	0	1	00001001	1	00000000	0	0
0	101	00000000011	000	0	1	00001010	0	00000000	0	0
0	101	00000000011	000	0	1	00001010	1	00000000	0	0
0	101	00000000011	000	0	1	00001011	0	00000000	0	0
0	101	00000000011	000	1	0	00001011	1	00000000	0	1
0	101	00000000011	000	1	0	00001011	0	00000000	0	1
0	101	00000000011	000	1	0	00001011	0	00000000	0	1
0	101	00000000011	000	1	0	00001011	0	00001010	1	0
0	101	00000000011	000	1	0	00001100	0	00001010	0	0
0	101	00000000011	000	0	0	00001100	1	00001010	0	0
0	101	00000000011	000	0	0	00001111	0	00001010	0	0
0	101	00000000011	000	0	0	00001111	1	00001010	0	1
0	101	00000000011	000	0	0	00001111	0	00001010	0	1
0	101	00000000011	000	0	0	00001111	0	00001000	1	0
0	101	00000000011	000	0	0	00001111	0	00001000	0	0
0	101	00000000011	000	0	0	00001111	0	00001000	0	0
0	101	00000000011	000	0	1	00001111	0	00000110	1	0
0	101	00000000011	000	0	1	00001111	0	00000110	0	0

SystemC: simulation stopped by user.
Press any key to continue

Fig. 5-43 SURVIVOR MEMORY Simulation

5.3.2.11 Min_Path

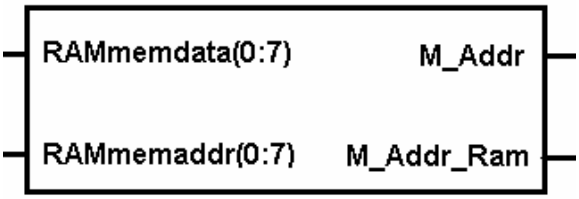


Fig. 5-44 MIN_PATH Symbol

The MIN_PATH module selects the DPRAM containing the most probable state metric (MEMORY, M_Addr) and directs the pre calculated register address of the state in DPRAM (Address, M_Addr_RAM). Fig. 5-42 illustrates the operation for DPRAM D, register 1.

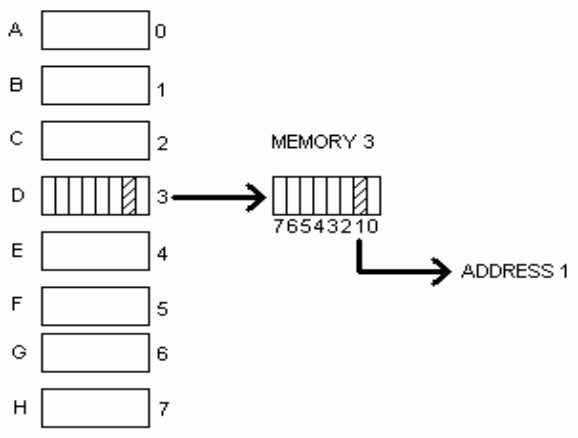
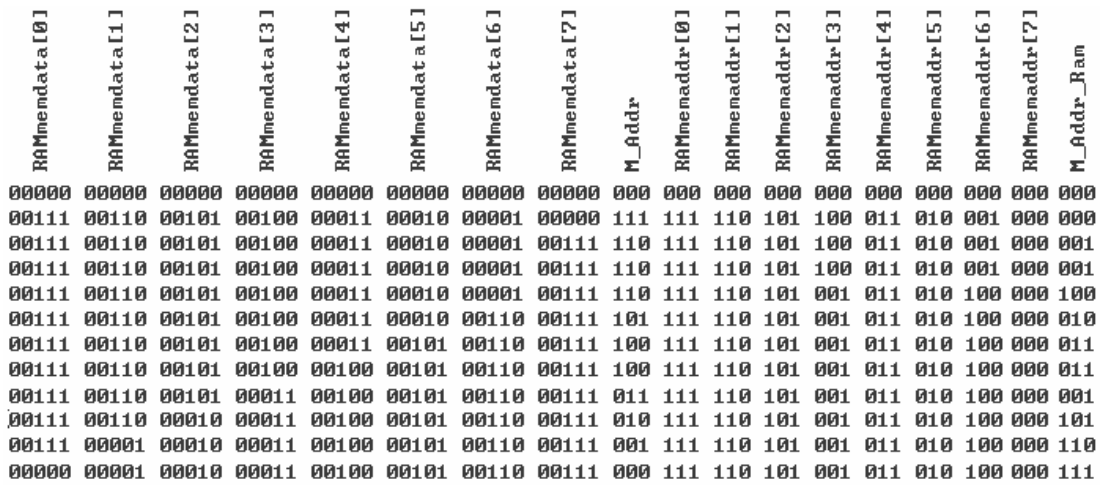


Fig. 5-45 MIN_PATH Example



RAMmemdata[0]	RAMmemdata[1]	RAMmemdata[2]	RAMmemdata[3]	RAMmemdata[4]	RAMmemdata[5]	RAMmemdata[6]	RAMmemdata[7]	M_Addr	RAMmemaddr[0]	RAMmemaddr[1]	RAMmemaddr[2]	RAMmemaddr[3]	RAMmemaddr[4]	RAMmemaddr[5]	RAMmemaddr[6]	RAMmemaddr[7]	M_Addr_Ram
00000	00000	00000	00000	00000	00000	00000	00000	000	000	000	000	000	000	000	000	000	000
00111	00110	00101	00100	00011	00010	00001	00000	111	111	110	101	100	011	010	001	000	000
00111	00110	00101	00100	00011	00010	00001	00111	110	111	110	101	100	011	010	001	000	001
00111	00110	00101	00100	00011	00010	00001	00111	110	111	110	101	100	011	010	001	000	001
00111	00110	00101	00100	00011	00010	00001	00111	110	111	110	101	001	011	010	100	000	100
00111	00110	00101	00100	00011	00010	00110	00111	101	111	110	101	001	011	010	100	000	010
00111	00110	00101	00100	00011	00101	00110	00111	100	111	110	101	001	011	010	100	000	011
00111	00110	00101	00100	00100	00101	00110	00111	100	111	110	101	001	011	010	100	000	011
00111	00110	00101	00011	00100	00101	00110	00111	011	111	110	101	001	011	010	100	000	001
00111	00110	00010	00011	00100	00101	00110	00111	010	111	110	101	001	011	010	100	000	101
00111	00001	00010	00011	00100	00101	00110	00111	001	111	110	101	001	011	010	100	000	110
00000	00001	00010	00011	00100	00101	00110	00111	000	111	110	101	001	011	010	100	000	111

Fig. 5-46 MIN_PATH Simulation

5.3.2.12 Min_Path_Conv

This module gets the identifier of DPRAM containing the minimum state metric (M_Location) and the iteration number in the DPRAM (M_Iteration) from MIN_PATH, then, converts these information to state identifier of the most probable state for the starting state of traceback operation.

K	M_Location	M_Iteration	Min_Path
000	000	000	000000
111	000	000	000000
111	001	000	000001
111	010	000	000010
111	011	000	000011
111	100	000	111111
111	101	000	111110
111	110	000	111101
111	111	000	111100
111	000	001	000100
111	001	001	000101
111	010	001	000110
111	011	001	000111
111	100	001	111011
111	101	001	111010
111	110	001	111001
111	111	001	111000
111	000	010	001000
111	001	010	001001
111	010	010	001010
111	011	010	001011
111	100	010	110111
111	101	010	110110
111	110	010	110101
111	111	010	110100
111	000	011	001100
111	001	011	001101
111	010	011	001110
111	011	011	001111
111	100	011	110011
111	101	011	110010
111	110	011	110001
111	111	011	110000
111	000	100	010000
111	001	100	010001
111	010	100	010010
111	011	100	010011
111	100	100	101111
111	101	100	101110
111	110	100	101101
111	101	100	101110
111	000	101	010100
111	001	101	010101
111	010	101	010110
111	011	101	010111
111	100	101	101011
111	101	101	101010
111	110	101	101001
111	111	101	101000
111	000	110	011000
111	001	110	011001
111	010	110	011010
111	011	110	011011
111	100	110	100111
111	101	110	100110
111	110	110	100101
111	111	110	100100
111	000	111	011100
111	001	111	011101
111	010	111	011110
111	011	111	011111
111	100	111	100011
111	101	111	100010
111	110	111	100001
111	111	111	100000

Fig. 5-47 SURVIVOR MEMORY Simulation

5.3.2.13 TraceBack Controller

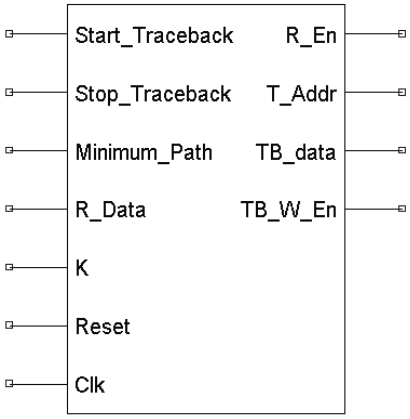


Fig. 5-48 TRACEBACK CONTROLLER Symbol

The traceback unit is the final stage of the decoder. Stating with the Start_Traceback signal the traceback unit reads the most probable (minimum path metric state) final state from MIN_PATH_CONV module. From the most probable state and by using the better transition information of each trellises in Survivor Memory, traceback unit estimates the transmitted message up to activation of Stop_Traceback port. The controller reads the probable transition information for the states of an iteration at a time through R_Data port by strobing R_En signal.

In a trellis for the Constraint Lenght greater than 4, there are more than one iteration so in each trellis there should be an offset value T_Addr to address the best state of iteration among other states in the trellis.

5.3.2.14 Main Controller

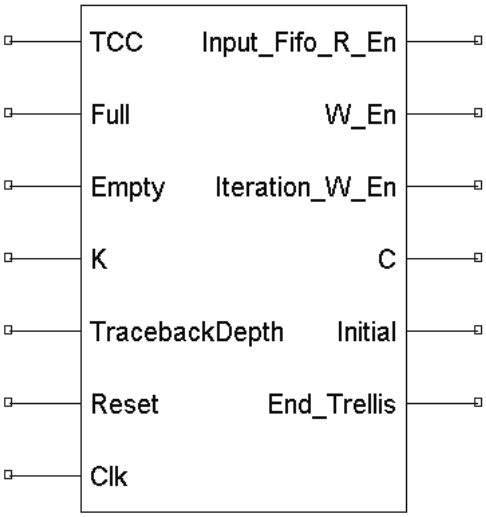


Fig. 5-49 MAINCONTROLLER Symbol

The Main Controller administers the Input FIFO, Trellis and Survivor Memory. The Main Controller gets K, Traceback Depth, Reset and Clk from Viterbi decoder inputs.

In the initial trellis main controller strobes the Initial signal to Reset thePath metrics of the Trellis. Then, the main controller checks Empty information of the Input FIFO. If the input FIFO is not empty, maincontroller activates Input_Fifo_R_En signal to supply the demodulated data from FIFO to the trellis. After that, Iteration Counter (C) is counted starting from 0 to 2^{K-4} for each trellis cycle. For every Iteration Counter value main controller checks the full signal of the Survivor memory and if the survivor memory is full main controller stand at that state until the memory is not full. In the sub-trellis unit the operations are performed asynchronously. With the memory not full signal, main controller strobes the Iteration_W_En signal to store the path metric values of the states in iteration to the DPRAM and strobes the W_En signal to write the surviving transition informations to the Survivor Memory. When the iteration counter reaches the 2^{K-4} main

controller sends the end_trellis signal to update the output path metric memory of the DPRAM.

5.3.3 Simulations

The simulation of the decoder started with the simulation of the basic modules as shown above and continued with the interconnected modules. After the completion of reconfigurable Viterbi decoder module the SystemC simulations are carried out for several parameters. For simulation purpose, the test bench shown in figure below was established.

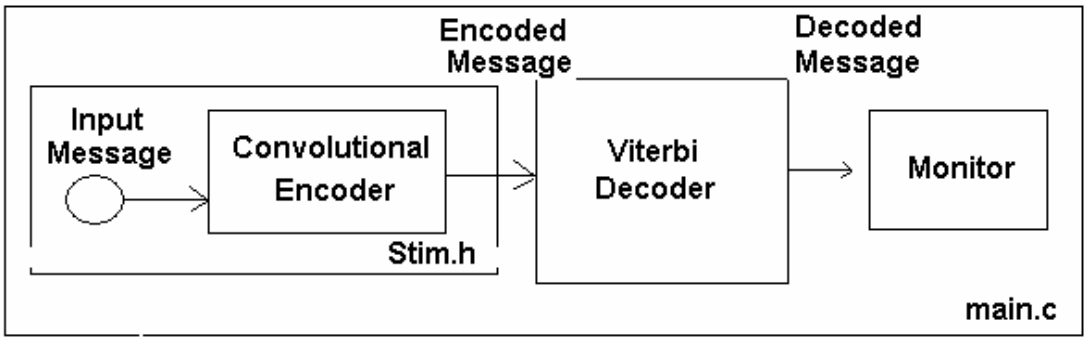


Fig. 5-50 Test Bench

Explanation of the simulation parameters can be summarized as in Fig. 5-51

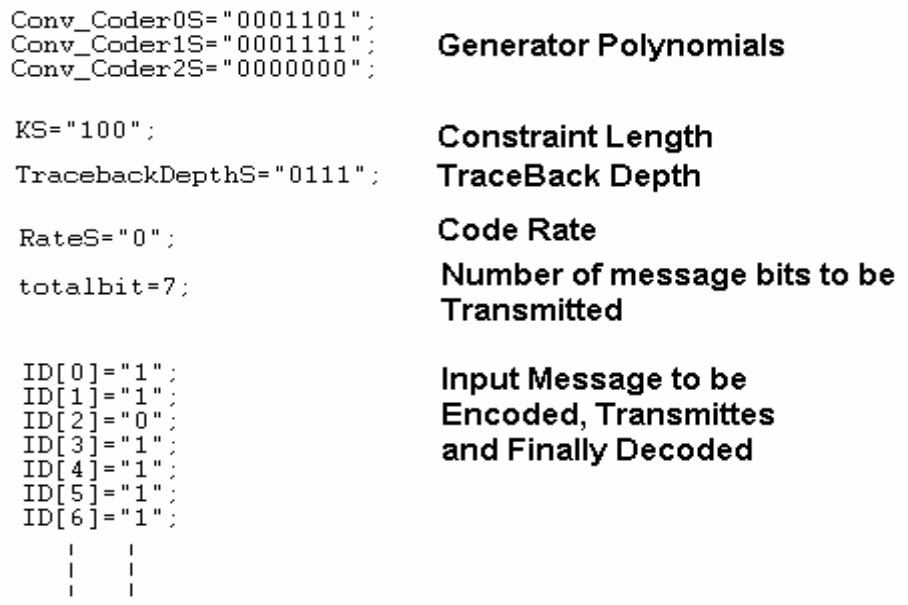


Fig. 5-51 Paramerters in the Simulations

In all the simulations results below the bottom line denotes the decoded message last transmitted message to first transmitted message bit order.

5.3.3.1 Configuration of Constraint Length

- **Viterbi Decoder K=4**

```
Conv_Coder0S="0001101";
Conv_Coder1S="0001111";
Conv_Coder2S="0000000";
KS="100";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="1";
ID[2]="0";
ID[3]="1";
ID[4]="1";
ID[5]="1";
ID[6]="1";
```

Fig. 5-52 Simulation Paramerters

```
                                    ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
1  1  1  1  0  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-53 Monitor Based Simulation Result

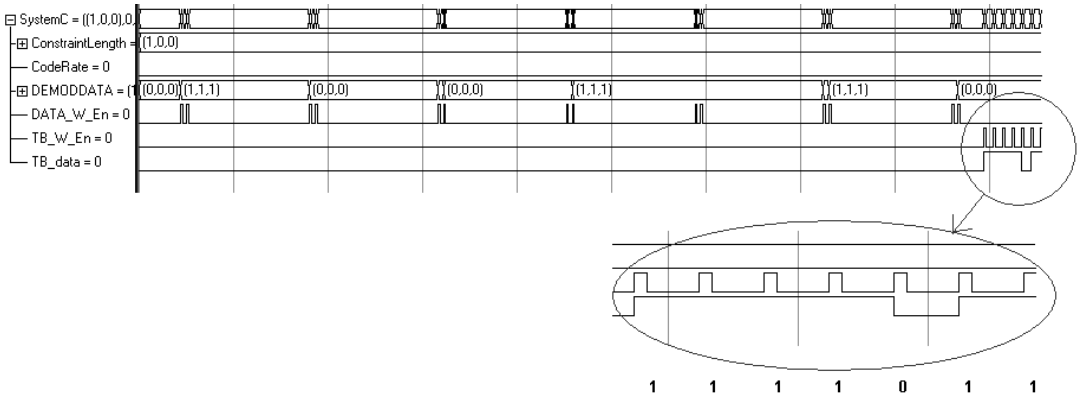


Fig. 5-54 Simulation Waveform

➢Viterbi Decoder K=5

```
Conv_Coder0S="0011011";
Conv_Coder1S="0011111";
Conv_Coder2S="0000000";
KS="101";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="1";
ID[2]="0";
ID[3]="1";
ID[4]="1";
ID[5]="1";
ID[6]="1";
```

Fig. 5-55 Simulation Paramerters

```
0  1  2  3  4  5  6  7  8  9  10 11
1  1  1  1  0  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-56 Mpnitor Based Simulation Result

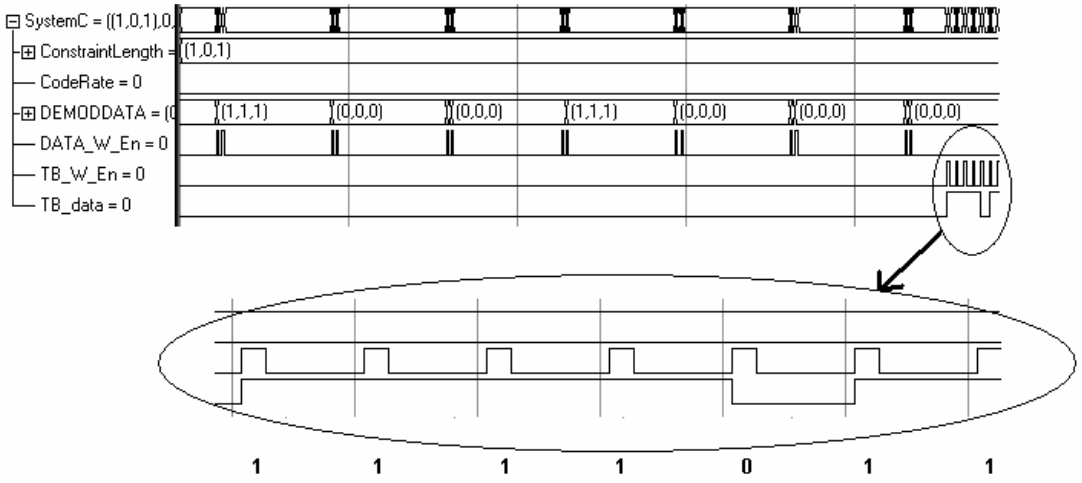


Fig. 5-57 Simulation Waveform

➢Viterbi Decoder K=6

```
Conv_Coder0S="0110011";
Conv_Coder1S="0111111";
Conv_Coder2S="0000000";
KS="110";
TracebackDepthS="111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="1";
ID[2]="0";
ID[3]="1";
ID[4]="1";
ID[5]="1";
ID[6]="1";
```

Fig. 5-58 Simulation Paramerters

```
                                ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
1  1  1  1  0  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-59 Monitor Based Simulation Result

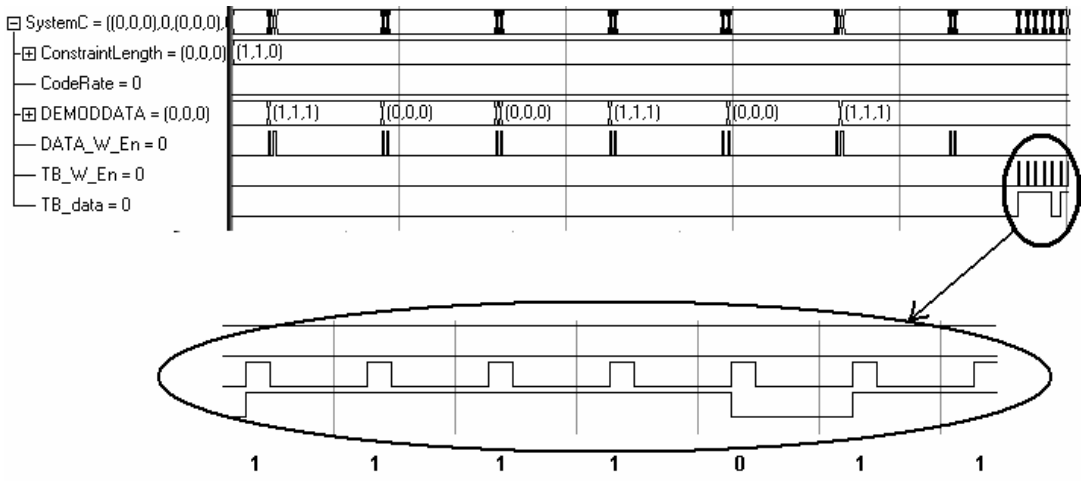


Fig. 5-60 Simulation Waveform

➢Viterbi Decoder K=7

```
Conv_Coder0S="1101011";
Conv_Coder1S="1110001";
Conv_Coder2S="0000000";
KS="111";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="1";
ID[2]="0";
ID[3]="1";
ID[4]="1";
ID[5]="1";
ID[6]="1";
```

Fig. 5-61 Simulation Parameters

```
                                ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11 1
1  1  1  1  0  1  1
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-62 Monitor Based Simulation Result

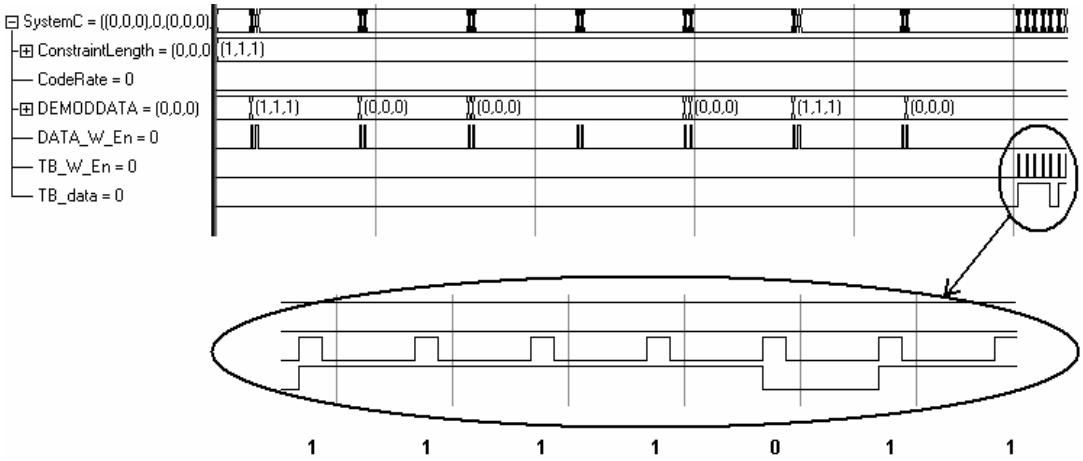


Fig. 5-63 Simulation Waveform

5.3.3.2 Configuration of Traceback Depth

➢ **Traceback Depth=11**

```
Conv_Coder0S="0110011";
Conv_Coder1S="0111111";
Conv_Coder2S="0000000";
KS="110";
TracebackDepthS="1011";
totalbit=11;
RateS="0";

ID[0]="1";
ID[1]="0";
ID[2]="1";
ID[3]="0";
ID[4]="0";
ID[5]="0";
ID[6]="1";
ID[7]="1";
ID[8]="0";
ID[9]="1";
ID[10]="0";
```

Fig. 5-64 Simulation Parameters

```
                    ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
0  1  0  1  1  0  0  0  1  0  1
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-65 Monitor Based Simulation Result

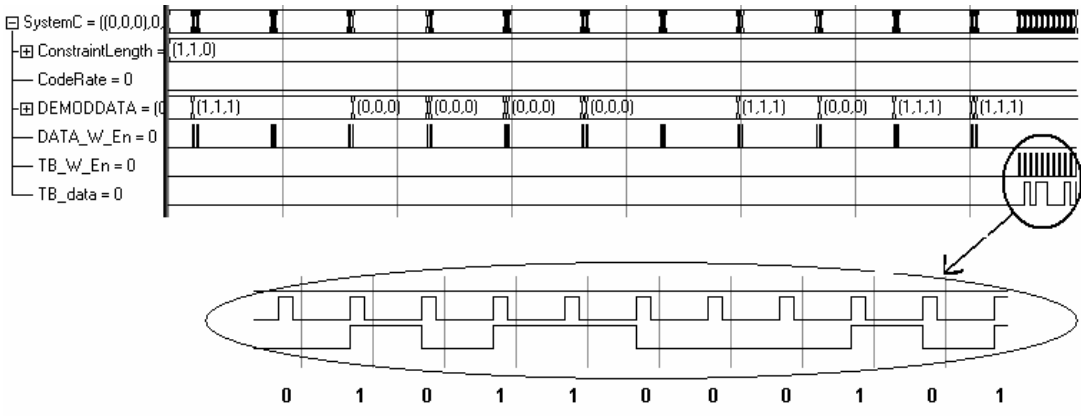


Fig. 5-66 Simulation Waveform

➢**Traceback Depth=15**

```
Conv_Coder0S="0110011";
Conv_Coder1S="0111111";
Conv_Coder2S="0000000";
KS="110";
TracebackDepthS="1111";
totalbit=15;
RateS="0";

ID[0]="1";
ID[1]="0";
ID[2]="1";
ID[3]="0";
ID[4]="0";
ID[5]="0";
ID[6]="1";
ID[7]="1";
ID[8]="0";
ID[9]="1";
ID[10]="0";
ID[11]="1";
ID[12]="1";
ID[13]="1";
ID[14]="1";
```

Fig. 5-67 Simulation Parameters

```
        Copyright (c) 1996-2002 by all Contributors
                    ALL RIGHTS RESERVED
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 :
1  1  1  1  0  1  0  1  1  0  0  0  1  0  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-68 Monitor Based Simulation Result

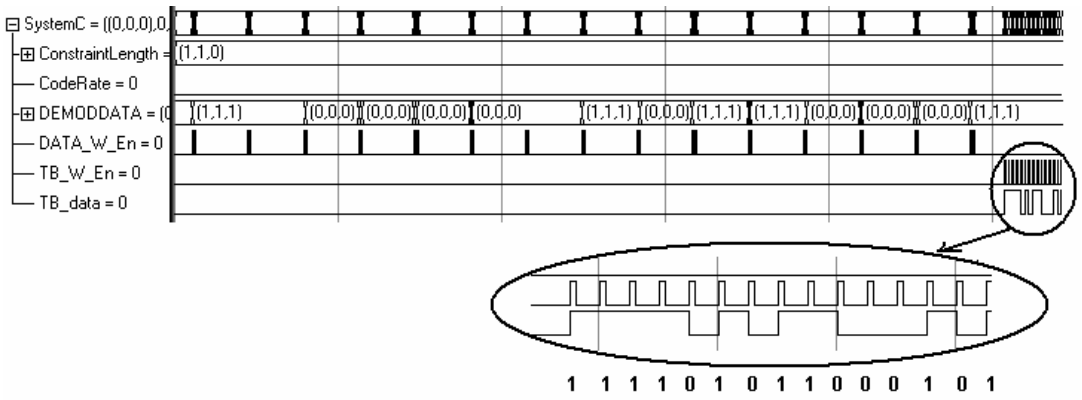


Fig. 5-69 Simulation Waveform

5.3.3.3 Configuration of Code Rate

➢ **Rate=1/2**

```
Conv_Coder0S="0110011";
Conv_Coder1S="0111111";
Conv_Coder2S="0000000";
KS="110";
TracebackDepthS="111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="0";
ID[2]="1";
ID[3]="0";
ID[4]="0";
ID[5]="0";
ID[6]="1";
```

Fig. 5-70 Simulation Parameters

```
0  1  2  3  4  5  6  7  8  9  10 11 12
111 111
111 111
111 000
111 000
111 000
111 000
000 000
1  0  0  0  1  0  1
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-71 Monitor Based Simulation Result

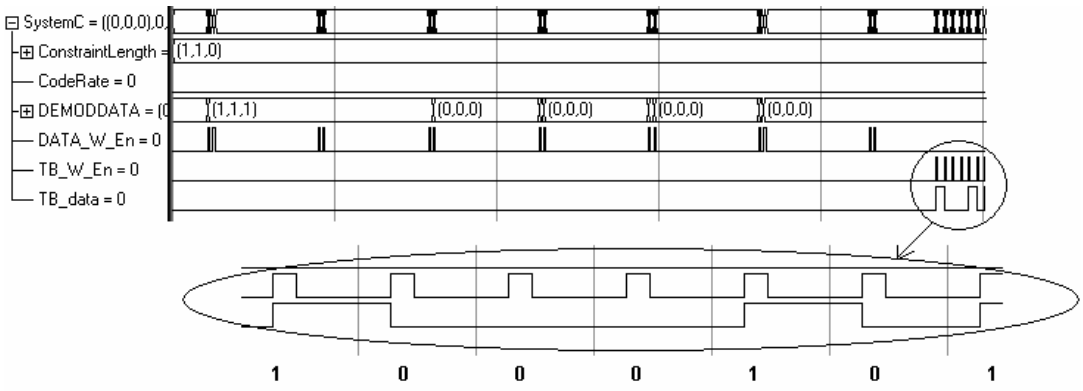


Fig. 5-72 Simulation Waveform

In these examples, the patterns above the decoded message show the convolutionally coded data to be decoded.

➢ **Rate=1/3**

```
Conv_Coder0S="0110011";
Conv_Coder1S="0111111";
Conv_Coder2S="0111011";
KS="110";
TracebackDepthS="111";
totalbit=7;
RateS="1";

ID[0]="1";
ID[1]="0";
ID[2]="1";
ID[3]="0";
ID[4]="0";
ID[5]="0";
ID[6]="1";
```

Fig. 5-73 Simulation Parameters

```
0  1  2  3  4  5  6  7  8  9  10 11 :
111 111 111
111 111 111
111 000 000
111 000 111
111 000 000
111 000 111
000 000 000
1  0  0  0  1  0  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-74 Monitor Based Simulation Result

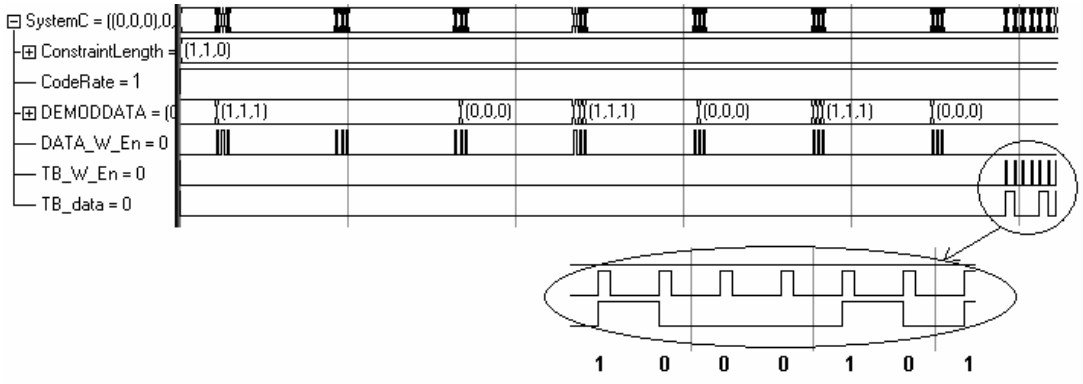


Fig. 5-75 Simulation Waveform

5.3.3.4 K=7 Viterbi Decoder with Different Messages

- **Msg=[0 1 0 1 0 1 0]**

```
Conv_Coder0S="1101011";
Conv_Coder1S="1110001";
Conv_Coder2S="0000000";
KS="111";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="0";
ID[1]="1";
ID[2]="0";
ID[3]="1";
ID[4]="0";
ID[5]="1";
ID[6]="0";
```

Fig. 5-76 Simulation Parameters

```
                    ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
0  1  0  1  0  1  0
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-77 Monitor Based Simulation Result

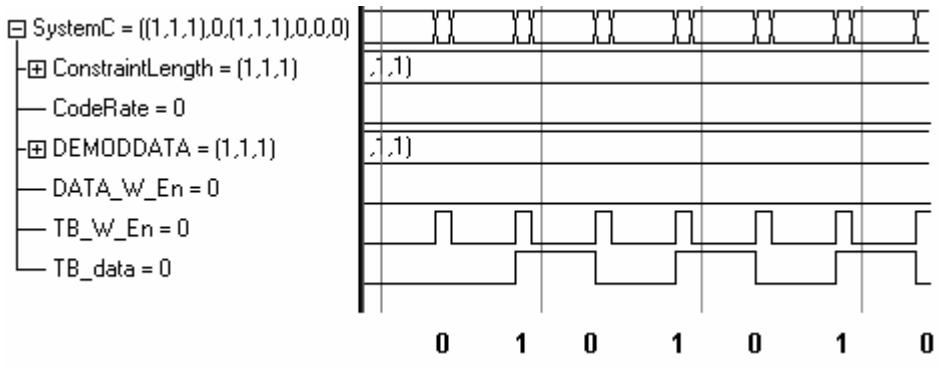


Fig. 5-78 Simulation Waveform

➢ **Msg=[1 0 1 0 1 0 1]**

```
Conv_Coder0S="1101011";
Conv_Coder1S="1110001";
Conv_Coder2S="0000000";
KS="111";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="0";
ID[2]="1";
ID[3]="0";
ID[4]="1";
ID[5]="0";
ID[6]="1";
```

Fig. 5-79 Simulation Parameters

```
0  1  2  3  4  5  6  7  8  9  10 11
1  0  1  0  1  0  1
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-80 Monitor Based Simulation Result

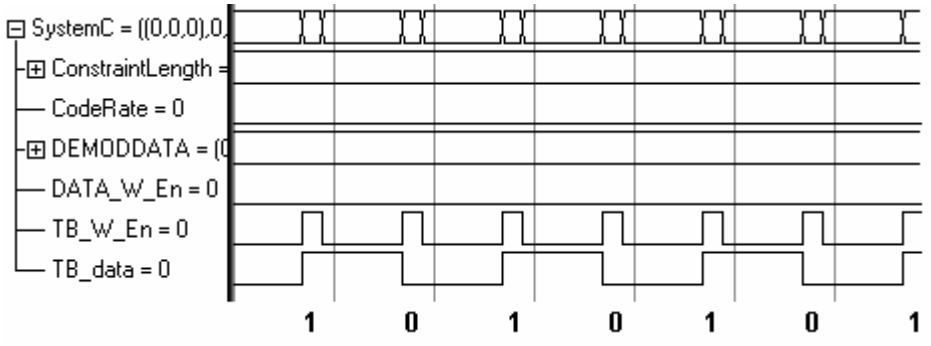


Fig. 5-81 Simulation Waveform

➢ **Msg=[1 1 0 0 1 0 1]**

```
Conv_Coder0S="1101011";
Conv_Coder1S="1110001";
Conv_Coder2S="0000000";
KS="111";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="1";
ID[2]="0";
ID[3]="0";
ID[4]="1";
ID[5]="0";
ID[6]="1";
```

Fig. 5-82 Simulation Parameters

```
                    ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
1  0  1  0  0  1  1
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-83 Monitor Based Simulation Result

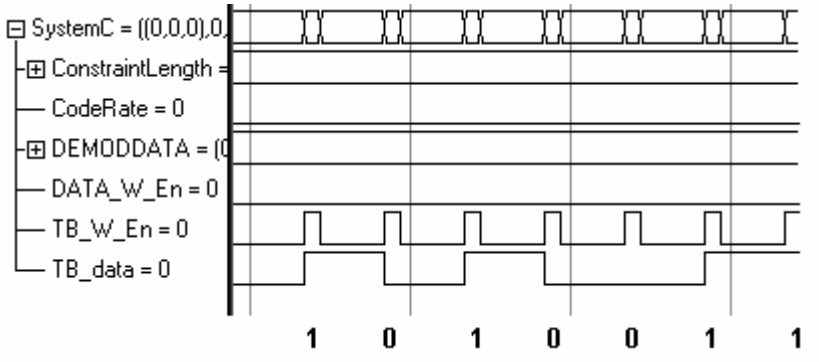


Fig. 5-84 Simulation Waveform

➢ **Msg=[0 0 0 0 0 0 0]**

```
Conv_Coder0S="1101011";
Conv_Coder1S="1110001";
Conv_Coder2S="0000000";
KS="111";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="0";
ID[1]="0";
ID[2]="0";
ID[3]="0";
ID[4]="0";
ID[5]="0";
ID[6]="0";
```

Fig. 5-85 Simulation Parameters

```
                    ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
0  0  0  0  0  0  0
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-86 Monitor Based Simulation Result

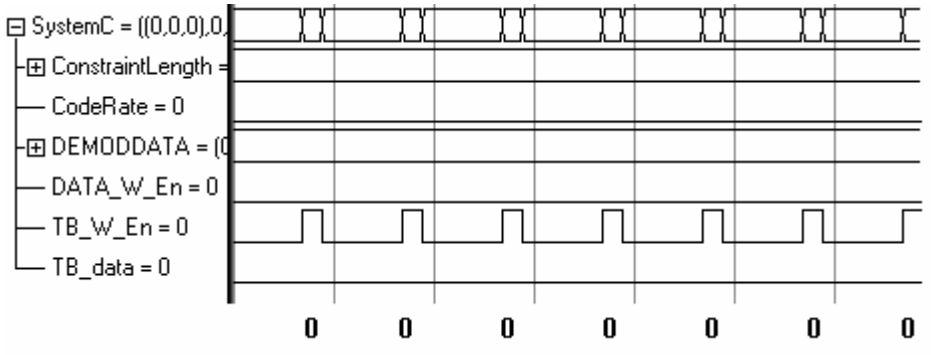


Fig. 5-87 Simulation Waveform

➢ **Msg=[1 1 1 1 1 1 1]**

```
Conv_Coder0S="1101011";
Conv_Coder1S="1110001";
Conv_Coder2S="0000000";
KS="111";
TracebackDepthS="0111";
totalbit=7;
RateS="0";

ID[0]="1";
ID[1]="1";
ID[2]="1";
ID[3]="1";
ID[4]="1";
ID[5]="1";
ID[6]="1";
```

Fig. 5-88 Simulation Parameters

```
                    ALL RIGHTS RESER
0  1  2  3  4  5  6  7  8  9  10 11
1  1  1  1  1  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-89 Monitor Based Simulation Result

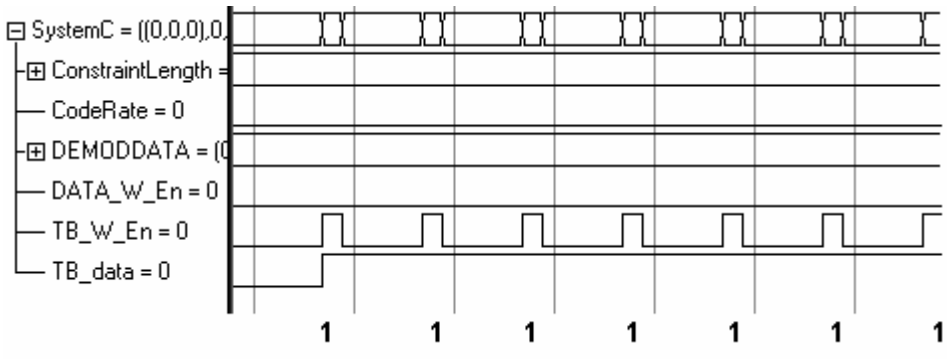


Fig. 5-90 Simulation Waveform

5.3.3.5 Error Correction Examples of Viterbi Decoder

- **Msg=[1 1 1 1 1 1 1]**

```
                    ALL RIGHTS RESERVED
Original Message :  1 1 1 1 1 1 1

Transmitted  :  111 111 000
Received     :  111 111 000

Transmitted  :  000 000 000
Received     :  000 000 000

Transmitted  :  000 111 000
Received     :  000 111 000

Transmitted  :  111 111 000
Received     :  111 111 000

Transmitted  :  111 111 000
Received     :  111 111 000

Transmitted  :  000 111 000
Received     :  000 111 000

Transmitted  :  111 000 000
Received     :  111 000 000

1  1  1  1  1  1  1
SystemC: simulation stopped by user.
Press any key to continue_
```

Fig. 5-91 Error Free Operation (Transmitted and Received Messages are Same)

In the error correction examples original message is given at the top. Then the convolutionally coded data is given in "Transmitted" labelled lines and the received signal as if transmitted in noisy channel is given in "Received" labelled

lines. Finally, the Viterbi Decoded messages of received signals are supported at the bottom lines.

```
Original Message :  1 1 1 1 1 1 1

Transmitted  :  111 111 000
Received     :  111 100 000

Transmitted  :  000 000 000
Received     :  000 000 000

Transmitted  :  000 111 000
Received     :  000 101 000

Transmitted  :  111 111 000
Received     :  111 110 000

Transmitted  :  111 111 000
Received     :  111 011 000

Transmitted  :  000 111 000
Received     :  000 111 000

Transmitted  :  111 000 000
Received     :  110 000 000

1  1  1  1  1  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-92 Decoding with Inserted Error

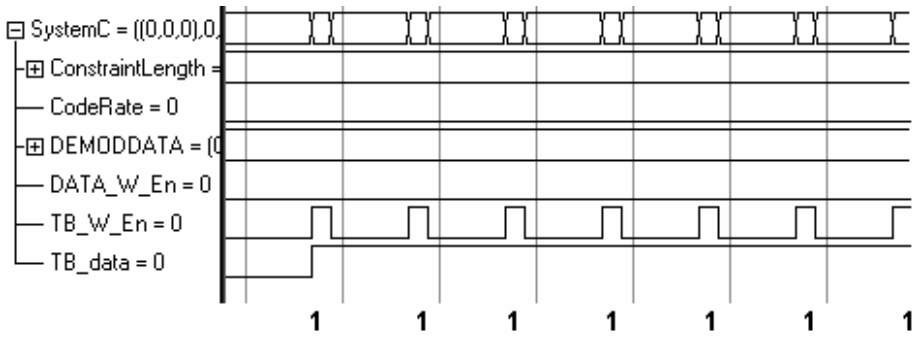


Fig. 5-93 Simulation Waveform

➢ **Msg=[1 0 1 0 1 0 1]**

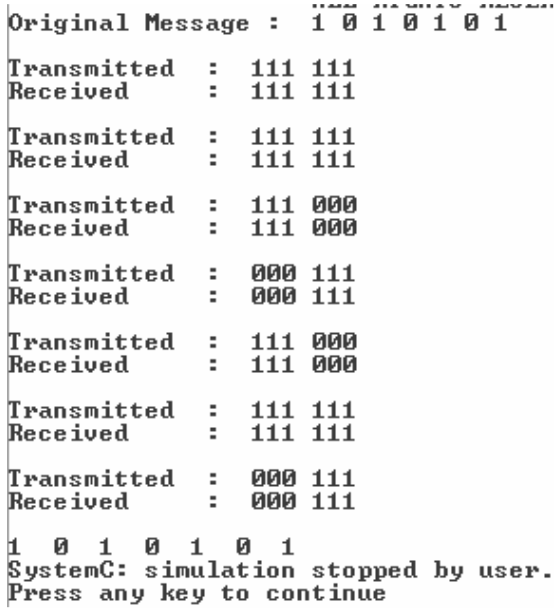


Fig. 5-94 Error Free Operation

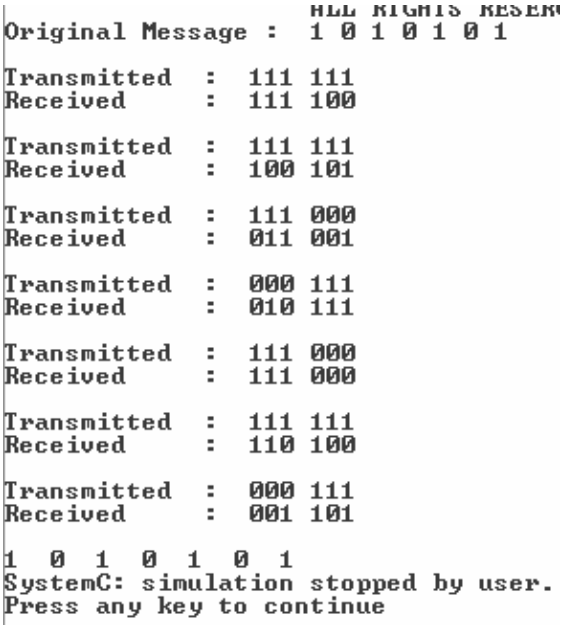


Fig. 5-95 Decoding with Inserted Error

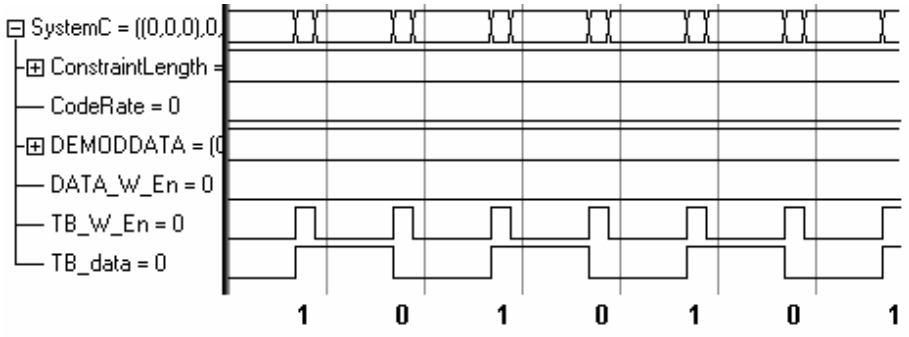


Fig. 5-96 Simulation Waveform

➢ **Msg=[1 0 1 0 1 0 1]**

```
        Copyright (c) 1996-2002 by a
                      ALL RIGHTS RESER
Original Message :  1 1 1 0 1 0 0

Transmitted  :  111 111
Received     :  111 111

Transmitted  :  000 000
Received     :  000 000

Transmitted  :  000 111
Received     :  000 111

Transmitted  :  000 000
Received     :  000 000

Transmitted  :  000 000
Received     :  000 000

Transmitted  :  111 111
Received     :  111 111

Transmitted  :  000 000
Received     :  000 000

0  0  1  0  1  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-97 Error Free Operation

```
                    ALL RIGHTS RESER
Original Message :  1 1 1 0 1 0 0

Transmitted  :  111 111
Received     :  111 100

Transmitted  :  000 000
Received     :  100 001

Transmitted  :  000 111
Received     :  011 101

Transmitted  :  000 000
Received     :  010 001

Transmitted  :  000 000
Received     :  011 011

Transmitted  :  111 111
Received     :  110 100

Transmitted  :  000 000
Received     :  001 011

0  0  1  0  1  1  1
SystemC: simulation stopped by user.
Press any key to continue
```

Fig. 5-98 Decoding with Inserted Error

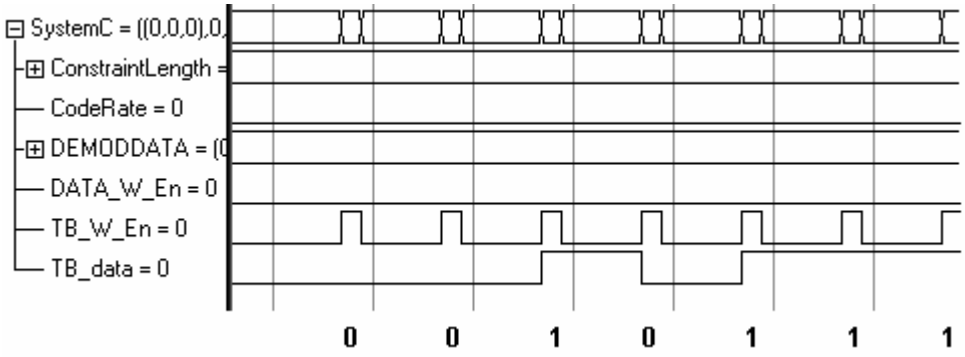


Fig. 5-99 Simulation Waveform

CHAPTER 6

CONCLUSION

In this thesis, VLSI implementation of a popular Maximum Likelihood error correction method called the Viterbi Algorithm has been studied. In the digital communication the Viterbi decoders are generally used to decode the messages correlated by convolutional encoders. The two decision types namely hard decision and soft decision Viterbi decoders have been implemented using SystemC. To decrease the switching and memory addressing complexity of the decoder designs, two novel area efficient reconfigurable Viterbi decoders, depending on the rearrangement of the states of the trellis structures, have been suggested. Finally, the implementation and the simulations of the second suggested reconfigurable Viterbi decoders architecture has been carried out in SystemC, with the configurable parameters of constraint length, code rate, transition probabilities, traceback depth and generator polynomials.

In the VLSI design of the area efficient implementation of reconfigurable Viterbi decoders, SystemC platform has been used. SystemC has some advantages over the traditional design cycle. In the thesis, both design procedures have been analysed and the comparisons have been stated. In addition, the SystemC supporting C++ platforms have been listed and configuration of the Microsoft Visual Studio 6.0 as a SystemC design and simulation platform has been described with an example implementation.

In the digital communication, the Viterbi decoder operates on the Convolutional Encoded data. Thus, an error correction capability is provided to Viterbi decoder. There are two decision types of the Viterbi decoders called soft and hard decision

Viterbi decoders. In the thesis, the operations of each decision types have been described and the both decision types have been implemented in SystemC.

In reconfigurable Viterbi decoder, increasing the constraint length causes an increase in the number of states. For the implementation of the concern, the increase in the number of states cause increase in the number of Add Compare Select units, Branch Metric units and interconnections between these units, so increase in the area. However, in this thesis, instead of implementation of a huge trellis, the implementation of a small portion of the trellis and several usage of this subtrellis in consecutive time instances, called iteration, has been suggested to increase the area efficiency.

On the other hand, the suggested structures in the literature have complicated memory addressing and switching mechanisms. So, in the implementation phase one to several port multiplexing path metric routers, several preliminary registers and area consuming look up tables embedded in ROM units have been used. During the implementation, a new structure has been sought to solve this complexity problems and it has been found that rearranging the states with complemented and vertically rotated way, decreases both switching and memory addressing complexities. Thus, two new reconfigurable Viterbi decoders have been obtained. The first method uses the normal states in the even and the vertically rotated states in the odd numbered iterations. For this method, a paper has been submitted to the IEEE Wireless Communication Magazine. The method was an area efficient method with a constraint that the method is operative for the number of iterations equal to or greater than two. However, in the second method normal and vertically rotated states have been used in the same iterations to get rid of the constraint of the first method.

Finally, the thesis lasted with implementation and simulation of the further developed reconfigurable Viterbi decoder architecture suggested in this thesis with configurable parameters listed below:

- Code Rate (selected as r=1/2 or r=1/3)

- Constraint Length (K) (ranging from 4 to 7)
- Generator Polynomial (determined externally)
- Traceback Depth (configurable)
- Transition Probabilities (adjustable)

In the implementation phase, for the synchronization, speed optimization and area efficiency, the techniques below have been utilized:

- The state metric location addressing of the lower DPRAMs were re-complemented. So, DPRAM addressing was simplified to be consistent with the iteration number leading the direct connection of iteration counter to the read and write address lines of the DPRAMs for trellis realization.
- An input FIFO was added to the design to simplify the signal timings of the demodulator side and synchronize the demodulator and the main controller of the core.
- To decrease the number of memory locations for state metric storage, the in-place path metric updating technique was used. In this technique the same state metric containing memories were overwritten for every trellis calculations.
- To obtain real time operation a special LIFO was generated as survivor memory. The survivor memory enables both the write operation of main controller and the read operation of the traceback unit. The number of the registers were tried to be minimized, so, only one set of memory registers just required for the maximum traceback depth were used. And both the main controller and traceback controller were optimized for the read and write operations from the same memory locations to overwrite the processed data and without corruption of the unprocessed data.

For the future works, to decrease the transmission bandwidth the convolutionally encoded message can be passed through another process called puncturation. So the depuncturer module can be added to the input of Viterbi decoder. The SystemC implementations can be converted into HDL by automatic conversion tools like CoCentric System Studio or System Crafter. (In the thesis period, the demo version of System Crafter was tried but because of the restricted usage of functionality the implementation couldn't converted into VHDL.) Also, the suggested trellis structures in Chapter 5 can be analysed for the implementation of FFT units in Digital Signal Processors.

REFERENCES

[1] Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", IEEE Trans. Information Theory, vol. IT-13, pp. 260-269, 1967.

[2] Forney, G.D., *The Viterbi Algorithm*, Proc. IEEE, vol. 61, pp.268-278, March 1973

[3] Black, P. J., Meng, T.H., *A 140-Mb/s, 32-state radix-4 Viterbi decoder*, IEEE J. Solid-State Circuits, vol. 27, pp. 1877–1885, Dec. 1992.

[4] Shung, C.B., Ling, H.D., Cypher, R., Siegel, P. H., Thapar, H.K., *Area-efficient architectures for the Viterbi algorithm—Part I: Theory*, IEEE Trans. Commun., vol. 41, pp. 636–644, Apr. 1993.

[5] McEliece, R.J., Lin, W., *The trellis complexity of convolutional codes*, IEEE Trans. Inform. Theory, vol. 42, pp. 1855–1864, Nov. 1996.

[6] Black, P.J., Meng, T.H., *A 1-Gb/s, four state, sliding block Viterbi decoder*, IEEE J. Solid-State Circuits, vol. 32, pp. 797–805, June 1997.

[7] Ju, W.S., Shieh, M.D., Sheu, M.H., *A Low-Power VLSI Architecture for the Viterbi Decoder,* 40th Midwest Symposium on Circuits and Systems, Sacramento, vol.2, pp. 1201- 1204, Aug. 1997

[8] Page, K., Chau, P.M., *Improved Architectures for the Add–Compare–Select Operation in Long Constraint Length Viterbi Decoding*, IEEE J. Solid-State Circuits, Vol. 33, pp. 151–155, no. 1, January 1998

[9] Kang, I., Willson, A., *Low-Power Viterbi Decoder for CDMA Mobile Terminals*, IEEE J. of Solid-State Circ., vol.33, no. 3, pp. 473-482, Mar. 1998.

[10] Suzuki, H., Chang, Y., Parhi, K.K., *Low-power bit-serial Viterbi decoder for next generation wide-band CDMA systems*, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Phoenix, AZ, vol. 4, pp. 1913 –1916, March 1999,

[11] Kunkel, J., Kranen, K., *SystemC demonstrates rapid progress*, EE Times, Sept. 2000.

[12] Lee, I., Sonntag, J.L., A *new architecture for the fast Viterbi algorithm*, IEEE Global Telecommunications Conference, San Francisco, vol. 3, pp. 1664 –1668, Nov. 2000.

[13] Swan, S., *An Introduction to System Level Modeling in SystemC 2.0*, Cadence Design Systems, Inc., draft report, May 2001.

[14] Chadha, K., Cavallaro, J.R., *A Reconfigurable Viterbi Algorithm*, Conference Record of 35th Asilomar Conference on Signals, Systems and Computers, vol.1, pp. 66-71, 2001.

[15] Benaissa, M., Zhu, Y., *A Novel High-Speed Configurable Viterbi Decoder for Broadband Access*, EURASIP Journal on Applied Signal Processing, pp.1317–1327, 2003

[16] Baird, M., *SystemC 2.0.1 Language Reference Manual*, Open SystemC Initiative (2003)

[17] Yarom, I., Glasser, G., Davidovitch, I., Mamet, D., *Three Different Usages of SystemC in Chip Design*, Intel ICGJ, 2003

[18] Zhu, Y., Benaissa, M., "Reconfigurable Viterbi decoding using a new ACS pipelining technique," Proceedings of the 2003 IEEE international conference on Application-Specific Systems, Architectures, and Processors (ASAP2003), The Hague, The Netherlands, pp. 360-368., 24-26 June 2003

[19] Bruels, N., Sicheneder, E., Loew, M., Schackow, A., Gliese, J., Sauer, C., *A 2.8 Gb/s, 32-State, Radix-4 Viterbi Decoder Add-Compare-Select Unit*, Symposium on VLSI Circuits Digest of Technical Papers, pp.170-173, June 2004

[20] Kesen, L., *Implementation of an 8-Bit Microcontroller with System C*, MSc. Thesis, Metu, November 2004

[21] Gang, Y., Arslan, T., Erdogan, A., *An Efficient Reformulation Based VLSI Architecture For Adaptive Viterbi Decoding In Wireless Applications*, SIPS 2004, IEEE Workshop on Signal Processing Systems, pp. 206-210, Oct. 2004

[22] Gang, Y., Arslan, T., Erdogan, A.T., *An Efficient Pre-Traceback Approach for Viterbi Decoding in Wireless Communication*, ISCAS 2005, IEEE International Symposium on Circuits and Systems, Vol. 6, pp. 5441- 5444, May. 2005

[23] Tessier, R., Swaminathan, S., Ramaswamy, R., Goeckel, D., Burleson, W., *A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder*, IEEE Transactions On Very Large Scale Integration (VLSI) Systems, vol. 13, no. 4, pp. 484 – 488, April 2005

[24] Dinhand, A., Xiao, H., *A Hardware-Efficient Technique to Implement a Trellis Code Modulation Decoder*, IEEE Transactions On Very Large Scale Integration (VLSI) Systems, vol. 13, no. 6, pp. 745-750, June 2005

[25] Askar, M., Sozen, S., *New Area Efficient Trellis Architecture for Reconfigurable Viterbi Decoder*, IEEE Wireless Communication Conference, 2007 (Submitted for publication)