جمهورية العراق

وزارة التعليم العالي والبحث العلمي

جامعة بابل

كلية الهندسة

قسم الهندسة الكهربائية

تصميم ومحاكاة مرمز ومعيد الترميز التلفيفي بالاعتماد على مصفوفة البوابات القابلة للبرمجة في المجال

بحث

مقدم إلى قسم الهندسة الكهربائية

جامعة بابل

كجزء من متطلبات نيل شهادة الدبلوم العالي

في الهندسة الكهربائية

من قبل:

عبير فاضل حسين محمد

بأشراف:

أ.م.د. أحمد عبد الكاظم حمد

٢٠٢٢ م

Republic of Iraq
Ministry of Higher Education
And Scientific Research
Babylon University
College of Engineering
Department of Electrical Engineering

Design and Simulation of Convolution Codec Based on Field Programmable Gate Array

A Research

Submitted to the Faculty of Engineering University of Babylon in Partial Fulfilment of the Requirements of Degree of Diploma in Engineering/ Electrical Engineering

BY

Abeer Fadhel Hussein Mohammed

***Supervised by*:**

Asst. Prof. Dr. Ahmed AL. Rekabi

2022

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

﴿وَلَقَدْ كَرَّمْنَا بَنِي آدَمَ وَحَمَلْنَاهُمْ فِي الْبَرِّ وَالْبَحْرِ وَرَزَقْنَاهُم مِّنَ الطَّيِّبَاتِ وَفَضَّلْنَاهُمْ عَلَىٰ كَثِيرٍ مِّمَّنْ خَلَقْنَا تَفْضِيلًا﴾

صَدَقَ اللهُ العَلِيُّ العَظيم

سورة الإسراء (٧٠)

Thanks and appreciation

- Praise be to God, the Mighty, the Almighty, who has bestowed upon me by His grace and generosity by completing this research to obtain a higher diploma in electrical engineering. Thanks be to God first and last, and to the Noble Messenger Muhammad (may God bless him and his family), and to the pure Ahl al-Bayt (peace be upon them).
- Then I would like to extend my gratitude to the supervisor of my research, Dr. (Ahmed Abdul-Kadhim) for his great effort in guiding me to complete this scientific work.
- I also have the right to extend my heartfelt thanks to everyone who supported me, supported me, and helped me from the best and the best, in order to complete this research.

Dedication

FOR

- Whose guardianship is my fortress, and his intercession is my hope

Ali Ibn Abi Talib (peace be upon him)

- My kind father, my role model, and my role model in life; He is the one who taught me how to live with dignity and honor.

- My tender mother, the epic of love and the joy of a lifetime, and an example of dedication

- My husband, the companion of the struggle who spared no time or effort to help me

- My brethren, I am my support and my arm, and I share my joys and my sorrows.

- My child and my beating heart.

I dedicate this blessed effort

researcher

List of Contents

SUBJECT	***PAGES***
Abstract	i
Table of Contents	ii
List of Abbreviations	v
List of Symbols	vi
List of Tables	vii
List of Figures	viii
CHABTER ONE	
1.1 Background	1
1.2 CHANNEL CODING	2
1.3 MOTIVATIONS AND OBJECTIVES	3
1.4 LITERATURE REVIEW	3
1.5 ORGANISATION OF THESIS	5
CHABTER TWO: CONVOLUTIONAL CODES AND VITERBI DECODING	
2.1 Introduction	6
2.2 BLOCK CODES	6
2.3 Convolutional codes	7
2.4 Convolution Encoder	7
2.5 Representations of Convolutional Codes	9

SUBJECT	***PAGES***
2.6 Recursive encoder	12
2.7 Trellis Termination	14
2.8 Catastrophic Encoders	14
2.9 Viterbi Decoder	15
2.10 TYPES OF VITERBI DECODING	17
2.10.1 HARD VITERBI DECODING	17
2.10.2 SOFT VITERBI DECODING	18
2.11 LIMITATION OF VITERBI ALGORITHM	20
2.12 APPLICATIONS OF CONVOLUTIONAL CODES	21
Chapter three: FPGA Architecture and Design Flow	
3.1 Introduction	22
3.2 FPGA Configuration Methods	22
3.3 FPGA architecture	22
3.4 FPGA Implementation	26
3.4.1 Overview of FPGA Design Flow	26
3.4.2 Design Entry	27
3.4.3 Behavioral Simulation	28
3.4.4 Design Synthesis	30
3.4.5 Design Implementation	32
3.5 Pipelining	34
3.6 Exporting the design as intellectual property (IP)	36

SUBJECT	***PAGES***
3.7 FPGA Applications	36
3.7.1 Medical applications	36
3.7.2 On-Board Artificial intelligent for Autonomous Space Exploration	37
3.7 Exporting the design as intellectual property (IP)	37
CHAPTER FOUR: HARDWARE AND SOFTWARE SIMULITION RESULTS	
4.1 Implementation of convolutional encoder	38
4.2 Implementation of trellis	43
4.3 Implementation of convolutional hard Decoder Circuit	46
4.4 Implementation of convolutional soft Decoder Circuit	49
4.5 Bit-error rate (Pe) performance of Convolutional code	54
Chapter Five Conclusions and Suggestions for Future Works	
5.1 Conclusions	64
5.2 Future Works	65
Refrence	66

List of Abbreviations

Abbreviation	Definition
AND	AND gate
ARQ	automated repeat request
AWGN	Additive White Gaussian Noise
BCH	Bose Chaudhuri Hochquenghem
BRAM	Block Random Access Memory
CT	Computed tomography
CW	Code word
DSP	digital signal processor
FF	Flip-Flops
FEC	Forward Error Correction
FIR	Finite-Impulse-Response
FPGA	Fie1d Programmab1e Gate Array
GP	Generator Polynomial
GSM	Global System Mobile
HDL	Hardware Description Language
HLS	High-Level-Synthesis
lOB	Input Output B1ocks
IP	Intellectual Property
ISE	Integrated Synthesis Environment
LC	logic cell
LE	logic element
LUT	Look-Up Tables
MRI	Magnetic resonance imaging
NASA	National Aeronautics and Space Administration
NCD	Native Circuit Description
NGC	Native Generic Circuit
NGD	Native Generic Database
RAM	Random-access memory
RTL	Register Transfer Level
PAR	Place and route
PET	positron emission tomography
RSC	Recursive Systematic Convolutional
SDK	Software Development Kit
TDMA	time division multiple access
UCF	User Constraints File
VA	Viterbi algorithm
VHDL	VHSIC-Hardware Description Language
XOR	EX-ORing

List of Abbreviations

List of Figure

Figure number	Title
1.1	Digital Communication System
2.1	Convolutional encoder and Viterbi decoder
2.2	A convolution encoder's block diagram
2.3	state diagram
2.4	tree diagram
2.5	trellis diagram
2.6	The RSC encoder
2.7	RSC encoder trellis diagram
2.8	RSC encoder termination approach using a trellis.
2.9	Viterbi decoder block diagram.
2.10	The Add Compare Unit
2.11	The trellis of hard decoding
2.12	The Trellis with bipolar coded
2.13	The trellis of soft decoding
3.1	Applications of FPGA in Diagnostic imaging systems
3.2	Systems for robotic surgery
3.3	Underlying FPGA fabric.
3.4	A simplified representation of a Xilinx LC
3.5	A slice with two logic cells.
3.6	A CLB with 4 slices is depicted (The number of slices is determined by the FPGA family).
3.7	Flow for FPGA Design
3.8	The development of core function ConvDecHard in HLS IDE.
3.9	The test bench function ConvHDTB and the resulted output.
3.10	RTL/ Cosimulation for the core function ConvEnc.

List of Figure

Figure number	Title
3.11	synthesis report for the ConvDecHard function.
3.12	Behavior of Function Pipelining
3.13	Adding pipeline to the design.
3.14	the IP block for the ConvDecHard core function.
4.1	IP of the convolutional encoder
4.2	Flow Chart of convolutional encoder
4.3	IP of the trellis
4.4	IP of convolutional hard decoder
4.5	IP of convolutional soft decoder
4.6	Flow Chart of convolutionald ecoder
4.7	BER and FER performance against E_b/N_o for massage length k=32, without trellis termination
4.8	BER and FER performance against E_b/N_o for massage length k=32, with trellis termination
4.9	BER and FER performance against E_b/N_o for massage length k=46, without trellis termination
4.10	BER and FER performance against E_b/N_o for massage length k=64, with trellis termination
4.11	BER and FER performance against E_b/N_o for massage length k=128, without trellis termination
4.12	BER and FER performance against E_b/N_o for massage length k=128, with trellis termination
4.13	BER and FER performance against E_b/N_o for massage length k=156, without trellis termination
4.14	BER and FER performance against E_b/N_o for massage length k=256, with trellis termination
4.15	compare of different lengths

List of Symbols

Symbol	Definition
a	Input 0
b	Input 1
C (x)	The output signal
CW_BUS	the vector of codeword
DecodedBitsH_BUS	the vector of output bits from decoder
E_b	broadcast energy per bit
E_b/N_0	Energy per bit over noise spectral density
g(x)or g(D)	matrix transfer function
gFBE_BUS	feedback generator polynomial
gFFE_BUS	feedforward generator polynomial
INP_BUS	the vector of input message
K	constraint length
k	input symbols
L	The signal Length in the Input.
l	length of the convolutional encoder
lenH_BUS	the length of the massage sent
M	the length of a shift register stage
m(x)	The input signal
memE_BUS	memory length of convolutional encoder
msgLen_BUS	message length
n	output symbols
NS	Next state
nstateH_BUS	the number of state
N_0	Noise spectral density
$N_o/2$	spectral density of two-sided power
Pe	Bit-error rate
PS	Present state
R	coding rate
rxH_BUS	the vector of the message sent
TERM_BUS	trellis termination indicator
x or D	Flip flop
2^k	Number of paths
$2^{(k-1)}$	Number of state
$2^{k(K-1)}$	Number of metrics
σ^2	noise-variance

List of Tables

Table number	Title
2.1	The encoder's truth table
4.1	delay time and required space from convolutional encoder circuit for solution 1.
4.2	delay time and required space with pipelining from convolutional encoder circuit for solution 2.
4.3	presents a comparison between solution 1 and 2.
4.4	delay time and required space from trellis circuit for solution 1.
4.5	delay time and required space with pipelining of trellis circuit for solution 2.
4.6	presents a comparison between solution 1 and 2.
4.7	Performance and utilization estimates for convolutional hard decoder circuit for solution 1.
4.8	delay time and required space with pipelining from circuit for convolutional hard decoder solution 2.
4.9	presents a comparison between solution 1 and 2.
4.10	Performance and utilization estimates for convolutional soft decoder circuit for solution 1.
4.11	delay time and required space with pipelining from circuit for convolutional hard decoder solution 2.
4.12	presents a comparison between solution 1 and 2.

Abstract

In this work, a design of the fundamental intellectual property (IP) blocks necessary to implement digital communication laboratory boards was introduced using High-Level-Synthesis (HLS). The boards simulating the error detection and correction systems in digital data which are transmitted over additive white Gaussian Noise channel (AWGN). The high speed and pipelining features of the Field Programmable Gate Array FPGA processor (xc7a100tcsg324-1) have been exploited to reduce the processing delay and hence increase the data throughput.

The design consists of two parts; The first one represents the transmitter and contains a random data source, ½ rate convolutional code with variable feedback and feedforward generators, and Cyclic Redundancy Check code (CRC) to detect the errors as the convolutional code fails to correct all errors in the received codeword as a result of its limited capability. The second board consists of hard and soft Viterbi decoders. Several C-simulation and Register Transfer Level RTL/cosimulation are carried out to test the performance of different coded system in terms of bit and frame error rate and reveals a matching between them.

Generally, the results show that coded systems present improvement over uncoded system especially for codes that use trellis termination. It is shown that as the codeword length becomes large the coded systems present further improvement. The results also reveal that the soft Viterbi decoding demonstrate best performance compared to the hard decoding. For the massage length k=32, with trellis termination and generator of $(7,5)_8$. The hard Viterbi decoder present a coding gain by 1.2 dB over uncoded system at BER of 10^{-4}. Meanwhile, the soft decoder has coding gain of 3.3 dB. The soft decoder introduces an improvement by 2.1 dB over hard decoder at BER of 10^{-4}.

الملخص

في هذا العمل ، تم تقديم تصميم لكتل الملكية الفكرية الأساسية (IP) اللازمة لتنفيذ لوحات مختبر الاتصالات الرقمية باستخدام برنامج (HLS). اللوحات تحاكي أنظمة اكتشاف الأخطاء وتصحيحها في البيانات الرقمية التي يتم إرسالها عبر قناة غاوسية بيضاء مضافة (AWGN) تم استغلال ميزات السرعة العالية وتوصيل الأنابيب لمعالج البوابات القابلة للبرمجة في المجال من نوع (xc7a100tcsg324-1 FPGA لتقليل تأخير المعالجة وبالتالي زيادة سرعة ارسال البيانات.

يتكون التصميم من جزأين ؛ يمثل الأول جهاز الإرسال ويحتوي على مصدر بيانات عشوائي ، ومرمز تلافيفية بمعدل ترميز ½ مع مولدات ترميز امامية وتراجعية متغيرة ، ورمز التحقق من التكرار الدوري(CRC) لاكتشاف الأخطاء حين يفشل الكود التلافيفي في تصحيح جميع الأخطاء في كلمة المرور المستلمة نتيجة لقدرته المحدودة. تتكون اللوحة الثانية من أجهزة معيد الترميز Viterbi الصلبة والناعمة. تم إجراء العديد من المحاكاة C و RTL / Cosimulation لاختبار أداء نظام ترميز مختلف من حيث معدل خطأ في البتات والفريم ويكشف عن تطابق بينهما.

بشكل عام ، تُظهر النتائج أن الأنظمة المرمزه تقدم تحسينًا على النظام غير المرمز خاصة بالنسبة للمرمزات التي تستخدم إنهاء الشبكة. يتضح أنه عندما يصبح طول الكلمة المرمزه كبيرًا ، فإن الأنظمة المرمزه تقدم مزيدًا من التحسين. تظهر النتائج أيضًا أن معيد الترميز Viterbi الناعم يُظهر أفضل أداء مقارنة بمعيد الترميز الصلب. بالنسبة لطول الرسالة k = 32 ، مع نهاية التشعير ومولد ٨(٧،٥). يقدم معيد الترميز الصلب Viterbi كسبًا للترميز ١,٢ ديسيبل على نظام غير مرمز عند معدل خطا بالبت $\mathrm{BER}=10^{-4}$. وفي الوقت نفسه ، يتمتع معيد الترميز الناعم بكسب ترميز يبلغ ٣,٣ ديسيبل. يظهر معيد الترميز الناعم تحسينًا ٢,١ ديسيبل على معيد الترميز الصلب عند معدل الخطأ في البتات$\mathrm{BER}=10^{-4}$.

CHAPTER-1
INTRODUCTION

1.1 Background

Error detection and repair is critical in digital communication systems for communication reliability. Forward Error Correction (FEC) is substantially complex. However, error detection methods have drawbacks. A function called Automated Repeat Request (ARQ) allows for the retransmission of blocks segments or packets that have been recognized as having problems.

In this case a protocol is required to reserve time for retransmission of incorrect blocks and reinsertion of repaired blocks in the right order. For this work there must be enough overall delay and buffering. A quarter second transmission delay in either direction makes synchronized satellite communication challenging.

The inefficiency of ARQ error detection at or beyond the system noise threshold is also a disadvantage. Decreased throughput occurs as the error rate approaches the packet length. A combination of forward error correction and ARQ error detection may greatly enhance throughput in certain situations.

For any of the following reasons, forward error correction may be preferred over error detection:

(1) When reverse channel is unavailable or ARQ latency is severe.

(2) A retransmit is not convenient.

1.2 Channel Coding

Noise immunity is a fundamental property of information transmission systems. Errors in communication channels during data transfer require error-correcting codes to be used. Forward Error Correction (FEC) improves channel capacity by adding carefully planned redundant information to a data being carried. Channel coding is the technique of combining redundant data [1].

Digital communication system building pieces presented in figure 1.1 are channel encoder, decoder, binary modulator and detector.

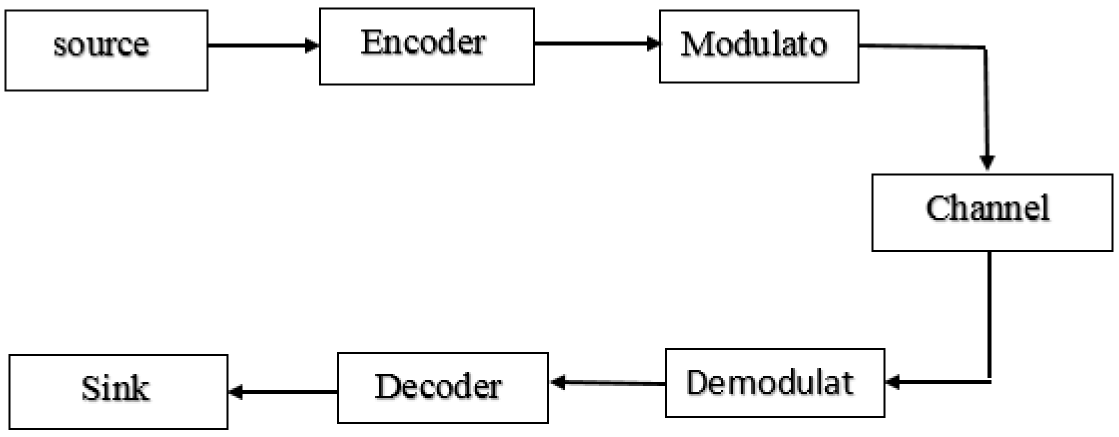


Figure 1.1: Digital Communication System

Transmitter has encoder and modulator. The encoder's job is to provide controlled redundancy to the binary data stream, which may be exploited by the receiver to counteract noise and interference. Encoding entails collecting k data bit at time and converting each k bit sequences into an n bit code word. An amount of redundancy introduced by this coding is assessed by the n/k ratio. Code rate is the reciprocal of this ratio. The modulator receives the binary sequence from the channel encoder and modifies one of the carrier parameters such as amplitude phase and frequency accordingly.

The demodulator decodes the contaminated waveform at the receiving end. Demodulator detector may decide if the transmitted bit is 0 or 1. This called hard decision. Soft decisions are made through quantization. The detector's quantized output is given to the channel decoder which uses redundancy to adjust for channel disruptions [2].

1.3 Motivations and Objectives

- A main objective of this work is to create send and receive laboratory boards that simulate convolutional coding using an Digilent FPGA device Artix-7 with a xilinx part number xc7a100tcsg324-1.
- The design is based on generating a schematic IP blocks using High-Level-Synthesis (HLS).

1.4 Literature Review

In January 2010, Dr. Dhafir A. Alneema and Yahya Taher Qassim [3], presented research about Implementation of a Convolutiona1 Encoder and a Viterbi Decoder on an FPGA that used a Multiple Booting Method. The FPGA is set up with the correct design based on the program that was loaded from the Intel flash memory into the FPGA Using this design, the FPGA may act as a Convolutional encoder or decoder, reusing the same hardware.
In 1999, Bupeish Panedita and Subier K Roy [4], presented a design and implemented of Viterbi Decoder based on FPGA by using HLS. The goal was to construct a 19.2kbps, 256-state Viterbi decoder that

was space efficient, but an architecture is flexible enough to accommodate larger input data rate.

In the year 2012. Suneha Gupta and Ms. Sakshi [1], indicated from the design and implementation of a Viterbi decoder that is more efficient. ACS recursion, or feedback loop, has been found to be most important factor in a speed at which pipelined VD can be decoded A latch has been added to feedback loop in order to stop the ACS recursion. This reduces a delay and thus increases frequency to 144.341MHz, which is even faster than before.

In 2012. Riham Ali Zbaid & Kasim K. Abdalla [5] presented the Design and Implementation of Convolutional Encoder - Viterbi Decoder Using FPGA. For development of a system in which a signal was sent after being encoded using Convolutional codes and received and decoded using the Viterbi a1gorithm the FPGA proved to be an effective mechanism for the implementation of the system. In the simulation result, it can be seen that the signal is received without any errors in the transmitted signa1 that also gives the FPGA a high level of integration, low development costs and doesn't use a lot of energy.

In 3.4 (2012), J. Tullasi, T. Venkatta Lakshmi and colleagues [6], present FPGA Implementation of Convolutional Encoder and Hard Decision Viterbi Decoder. A motivation of this paper is to realize the Viterbi decoder having K= 9 and R 1/2 by Xilinx 12.4i tools.

In 2007, Hema.S, Suresh Babu.V, and Ramesh [7], made an FPGA implementation of the Viterbi Decoder for department. A field-programmable gate array can be used to make a Viterbi Decoder that has a K of 11 and R of 1/3. This paper shows how to make this type of

Decoder. If you use more constraints in an encoding process like convolution, the more powerful your code will be.

In 21.1 (1999), M. Kivioja , and L. Vänskä & J. Isoaho [8] made Viterbi Decoder with FPGAs: Design and Implementation Viterbi decoder, K=7, was designed and simulated with VHDL in Synopsys and Mentor tool environments and further implemented on 4 Xilinx 4028EX devices using trace-back based architecture. Also parts of decoding algorithm are shown and looked at.

1.5 Organization of Thesis

Chapter 1: presents an introduction to error-correction coding.

Chapter 2: examines convolutional codes and the Viterbi algorithm.

Chapter 3: Design flow and pipelining using FPGA is covered in this chapter.

Chapter4: introduces different computer simulation design using High-Level-Synthesis (HLS) to generate IPs for convolution code (encoder, trellis and decoder) and comparisons between the result with and without pipelining are also presented.

Chapter 5: conclusions with a few suggestions for further work.

CHAPTER - 2

CONVOLUTIONAL CODES AND VITERBI DECODING

2.1 Introduction

The two most common types of channel coding are block Coding and convolution coding. On reasonably big message blocks, block codes work. Sequential data one or few bits at the time is used in convolution codes.

2.2 Block Codes

The first attempts at building error control technique depended on block codes. When there are a lot of information bits in a block there are a lot of redundant parity-check checks that are made from the information bits. They are sent with the information bit as codes that has the rate k/n bit per symbol. A linear feedback shift register encoder can produce these. When using any parity-check block code error detection is a simple process to accomplish. The incoming information bits are re encoded into parity check and Compared bit by bit with received duplicate parity check bit at the decoder. If any disparity happens a block error is reported [9] .

This approach dubbed syndrome decoding is easiest to apply with shift register encoders and decoders that use cyclic redundancy. Hamming Codes   Golay Codes and BCH. Codes are some of the most regularly utilized block codes. Convolutional codes have been the focus over the last decade. The output of a convolutional encoder is the convo1ution of input data. and filter impulse response. For the same encoder-decoder complexity convolutional codes outperform block codes [1].

2.3 Convolutional Codes

For similar encode/decode complexity convolutional codes outperform block codes and are frequently favored in practice. They were also among the first codes to benefit from good soft-decision decoding algorithms. To safeguard digital data transmission from accidental mistakes caused by noise sources convolutional codes are effective. The source symbols are suitably redundant to ensure error-free transfer [10].

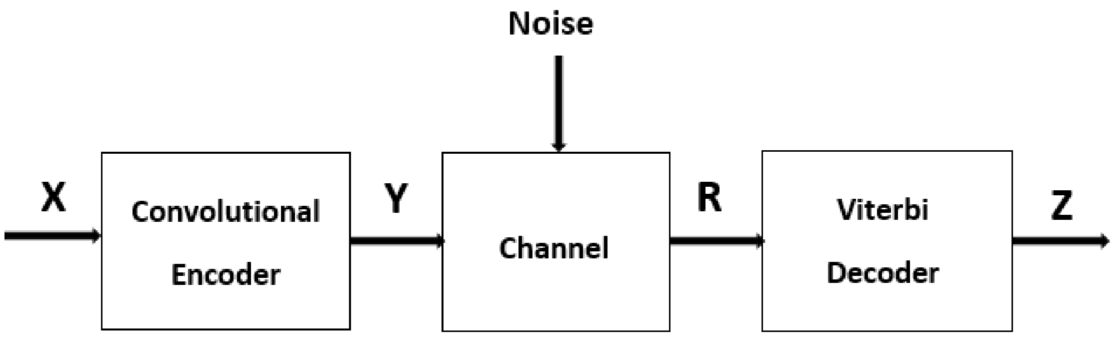


Figure 2.1 Convolutional encoder and Viterbi decoder [10].

Two parameters are often used to characterize convolutional codes the coding rate and the constraint length L. The coding rate is a ratio of input symbols (k) to output symbols (n) in a particular cycle. So r = k/n. bit/symbol is coding rate.

2.4 Convolution Encoder

The Convolution Encoder takes the input message stream and encodes it for transmission. The encoder creates several output bits for each input bit making the data more resistant to noise in the channel [11].

They assist determine and fix flaws in received patterns. In order to achieve uniformity the following nomenclature was developed:

(k) the convolution encoder's input bits.

(n) denotes the number of outputs bits (codeword length).

k/n = Code Rate for Convolution.

(M) In the encoder, the length of a shift register stage.

2^M = Count of states

(L) The constraint length of the encoder (L=M+1).

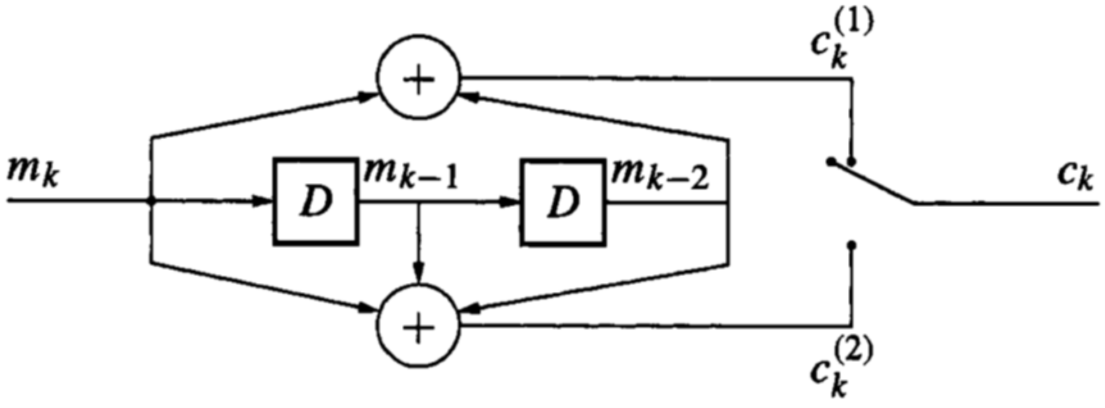


Figure 2.2: A convolution encoder's block diagram [11].

For example, the convolution encoder's block diagram is presented above in figure 2.2. It takes 3 input values 1 current and 2 pasts to create the output. A state is a collection of prior data values. The constraint length is the amount of input data values utilized to construct code. Each output is created by EX-ORing a present state with shifted input data. “Generator Polynomials” are binary strings that represent the pattern utilized to create the coded output value (GP). The input is represented by the MSB of the GP. The encoder is a linear non-systematic convolution encoder. Specifying the encoder connections is done using a generating polynomial. So the generator polynomial describes the convolution encoder mathematically. Each polynomial that makes up the generator polynomial has a maximum degree of L and describes the links between the shift registers and modulo-2 adders. There are two generating polynomials; $g_1(x) = (1+ x^2)$ and $g_2(x) = 1+x+ x^2$ that give $g_1(x) = (1\ 0\ 1)$ and $g_2(x) = (1\ 1\ 1)$ in binary representation or equivalently $g_1=5$, $g_2=7$ in octal form. Table 2.1 present the truth table of the encoder in figure 2.2

Table 2.1: The encoder's truth table

Input	PS	Out put	Ns
0	00	00	00
0	01	11	00
0	10	01	01
0	11	10	01
1	00	11	10
1	01	00	10
1	10	10	11
1	11	01	11

For both 0 and 1, the present state bits are the 2 bit pairs. The figure's expressions provide the output. Finally by right shifting the input and present state values by one bit the next state values are determined.

2.5 Representations of Convolutional Codes

Convolutional coding is commonly described using four different ways.

- Transfer Function Matrix
- State diagram
- Tree diagram
- Trellis diagram

a-Transfer Function Matrix

The input m(x) to output $C_1(x)$ transfer function is described as $g_1(x)=1+x^2$ with the m(x) to $C_2(x)$ transfer function as $g_2(x) =1+x+x^2$. It is possible to express the input stream m = 1 1 0 0 1 0 1. As $m(x) =1+x+ x^4+ x^6$.

The outputs are

$c_1(x)$=$m(x)g_1(x)$= (1+ x +x^4+x^6) (1+x^2) = 1+ x +x^2+x^3+x^4+x^8 . (2.1)

$c_2(x)$=$m(x)g_2(x)$=(1+ x +x^4+x^6)(1+ x +x^2) =1+x^3+x^4+x^5+x^7+x^8. (2.2)

It is necessary to link the rate R=k / n convo1utional code with encoder. which is defined as the k × n matrix transfer function G (x) also known as a transfer function matrix. In this case R = 1/2 code G (x) = [1+x^2 1+ x +x^2]. A feedforward (non-recursive) encoder or FIR encoder contains only po1ynomial elements in its transfer function matrix [12].

b - State Diagram

Transitions between states are represented in a state transition diagram by routes linking the states. Input bits are represented by 2^k pathways from each state. Paths to a state are also 2^k. When you look at a state transition diagram this is not able to pass from one state to another in the single transition. While the state transition diagram properly describes the encoder it falls short of monitoring transitions across time [13].

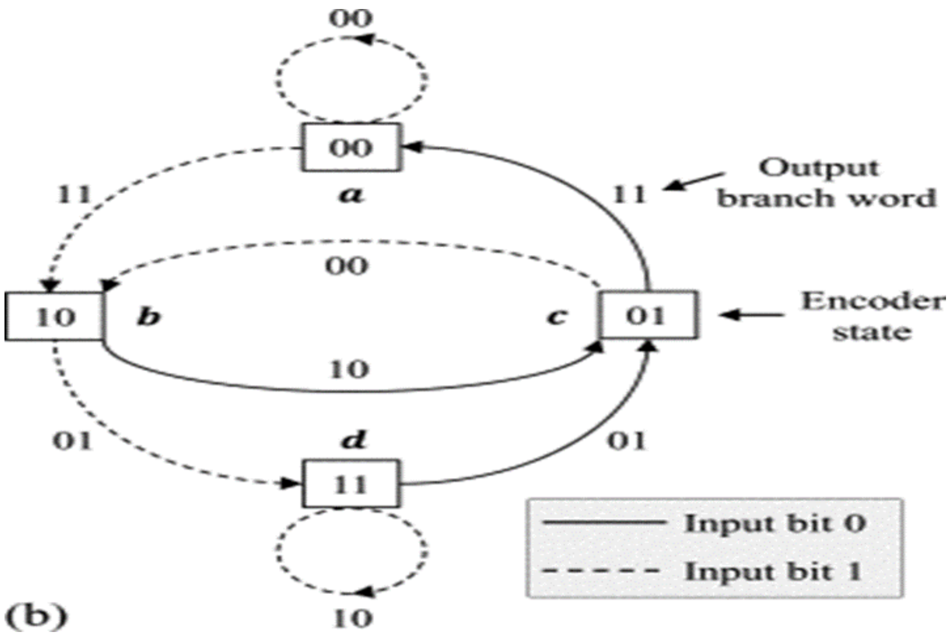


Figure 2.3 state diagram [13].

c -Tree Diagram

The tree diagram adds the time dimension allowing dynamic encoder description based on input sequence. Following the input bits, the encoding process is followed from left to right. If the input is a (0) users travel to the next right side in an upward position and if the input is a (1) users go to the next right side in a downward direction. The tree diagram's disadvantage is that the number of branches grows exponentially [13].

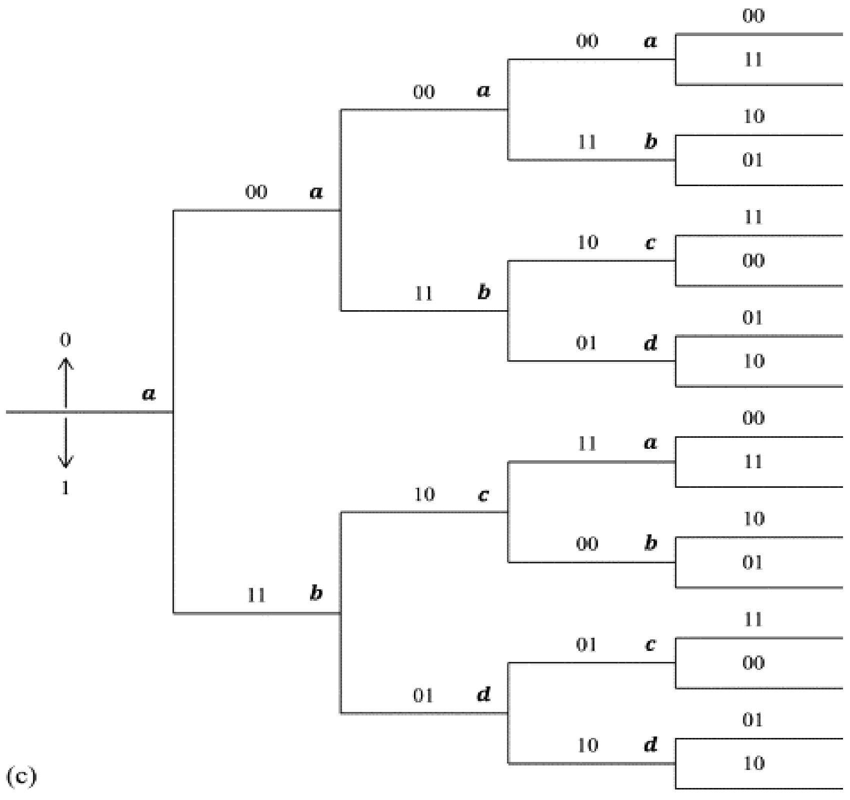
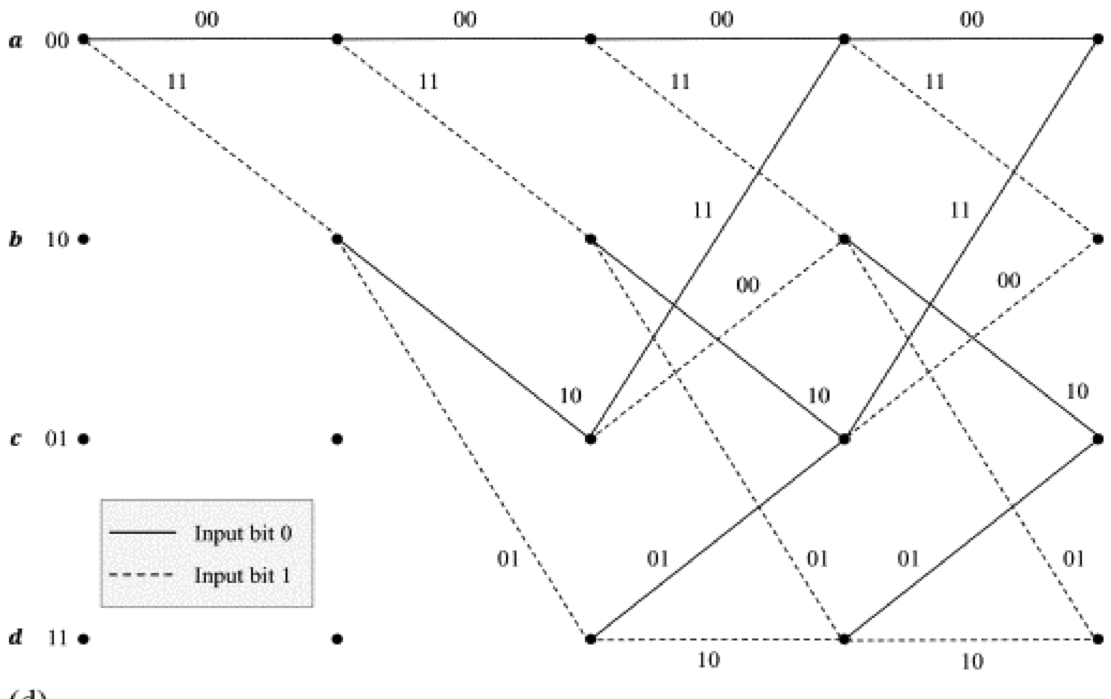


Figure 2.4 tree diagram [13].

d-Trellis Diagram

A trellis diagram by taking advantage of the repeating structure makes it easier to describe encoders. The trellis nodes represent encoder states. Transition diagram states trellis diagram states and tree diagram nodes all have a one-to-one relationship with each other in a way that is very easy to understand. To create a trellis diagram, place all states vertically. Paths on two vertical axes show how you move between two states. Each state has two trellis branches that branch out from it [13].



(d)

Figure 2.5 trellis diagram [13].

2.6 Recursive Encoder

When feed-back one of the outputs of a normal convolutional encoder a Recursive Systematic Convolutional (RSC) encoder can be created. Encoders with RSC have unlimited impulse response. Almost any input will provide a “good” (high weight) output. These can create “bad” (low weight) outputs [14].

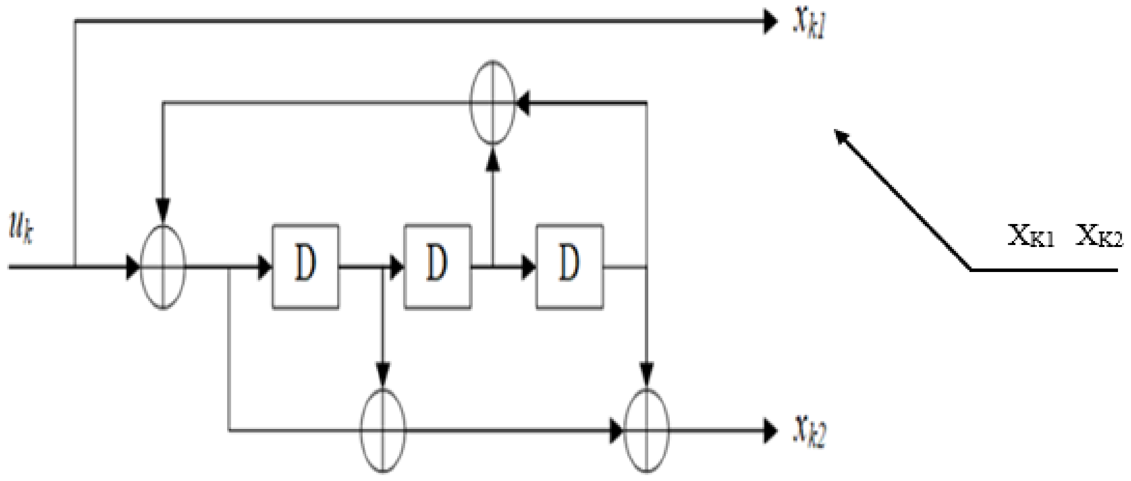
suppose an RSC code with the following generating matrix:

$$g_{RSC}(D) = \begin{bmatrix} 1 & \frac{g_{0(D)}}{g_{1(D)}} \end{bmatrix} \tag{2.3}$$

thus

$g_0(D) = 1 + D^2 + D^3$, and $g_1(D) = 1 + D + D^3$. Assuming the information sequence is $u(D)=1+D^7$, and its delayed form result in a self-terminating sequence with a finite weight when used as an input sequence. The code word generated as a result is $C(D)= (1+D^7 \quad 1+D+D^2+D^3+D^6+D^7)$.

A schematic diagram of this code is shown in Figure (2.6).



The RSC encoder is seen in Figure 2.6.

The trellis diagram of the RSC encoder shown in figure 2.6 is depicted in figure 2.7.

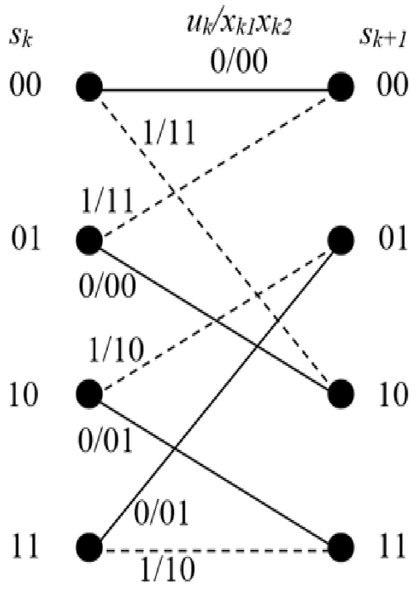


Figure 2.7: RSC encoder trellis diagram

2.7 Trellis Termination

After encoding all information bits, the tail bits are taken from the shift register's input. After encoding the data bits, the tail bits are padded. A basic solution that has been created in is presented in figure (2.8) to implement termination for RSC encoder. To encode the data input the switch is switched on to position A and the switch is switched on to position B to end the trellis

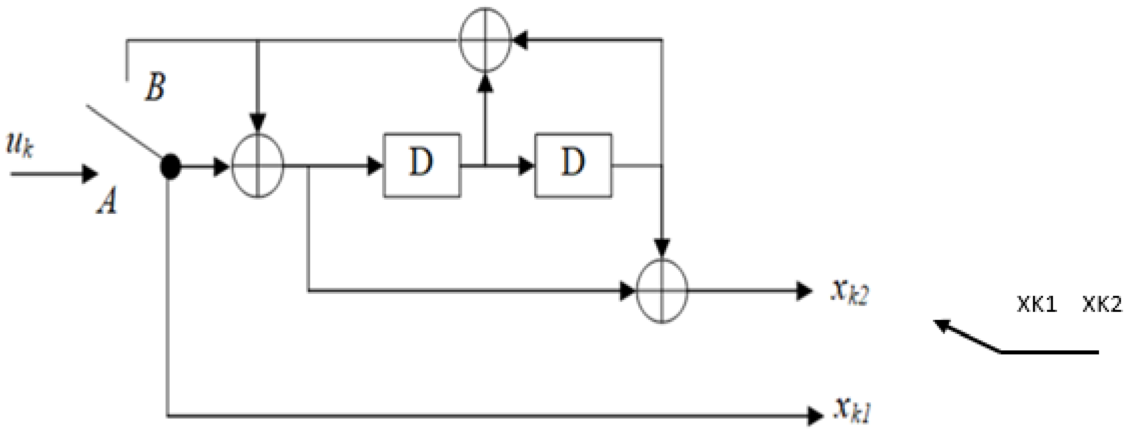


Figure (2.8): RSC encoder termination approach using a trellis [15].

2.8 Catastrophic Encoders

In addition to the inefficiency of the hardware there is another basic fault with G3 (x) of.

$$G3(x) = \begin{bmatrix} 1+x & 1+x^2 & x \\ x+x^2 & 1+x & 0 \end{bmatrix} \tag{2.4}$$

Take for example a value of

$$m(x) = [0 \quad \frac{1}{1+X}] \tag{2.5}$$

the case in which by using long division to increase the formal series

$$\frac{1}{(1+X)} = 1 + x + x^2 + \ldots \tag{2.6}$$

Consequently, the Hamming weight of the input sequence is unlimited. The following is the resulting output:

$$C(x) = m(x)\, Gg_3(x) = [\, x \quad 1 \quad 0] \tag{2.7}$$

The sequence with total Hamming Weight 2 Assume C(x) is channeled and two faults occur at the nonzero code elements. In such case the number of received sequences is exact1y zero which would decode (under any sensible decoding technique) to m (x) = [0 0 1]. A limited amount of faults in a channel results in an unlimited number of errors in the decoder and vice versa. Catastrophic encoders are the name given to such encoders [16].

2.9 Viterbi Decoder

The Viterbi decoding algorithm is the Convolutional code decoding technique. This approach is based on maximum Likelihood Hard decision decoding is employed when the incoming signal is sampled and quantized into two levels zero or one. In this type of decoding the path across trellis diagram is identified by utilizing hamming distance measure method. For example, the trellis is an extension of the State Diagram. That directly illustrates the state diagram as it evolves through time. Between the decoder's seen and encoder's transmitted symbols there is a hamming distance (number of bits). Input sequence consisting of information bits followed with reset sequence equals trellis length. In order to start tracing back from the original state the reset sequence is used. If you look at figure (2.9) you can see how a Viterbi decoder works [3].

The following are the primary tasks involved in the Viterbi decoding process:

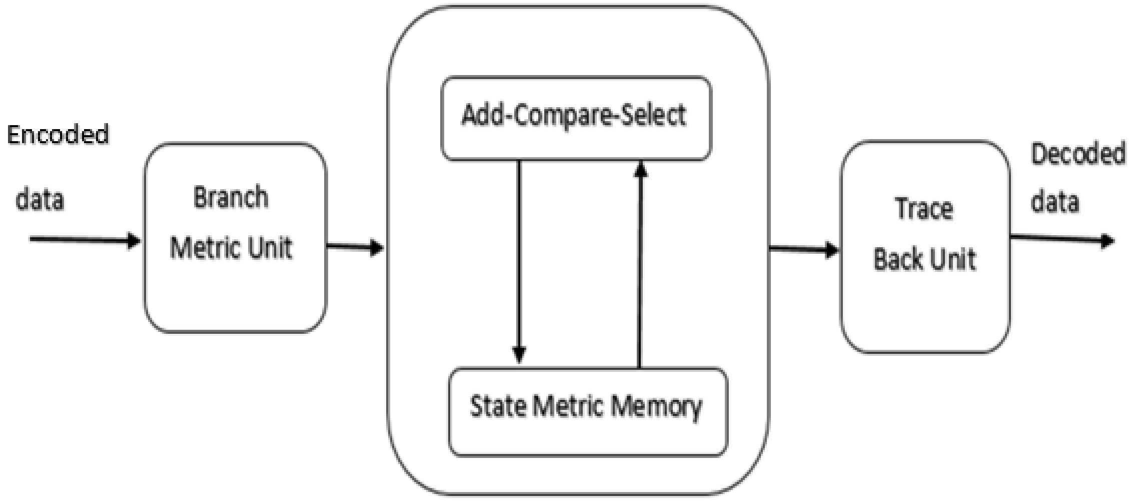


Figure (2.9) Viterbi decoder block diagram [3].

Partial path metrics can be made by adding up changes in one's own branch metric with changes in one's own state metric to get a new state metric. The state metric has two values since two branches enter each trellis node. As a result, the updated value of the state measure at each node will be the lowest possible value. To achieve this use, Add Compare Unit as indicated in figure (2-10).

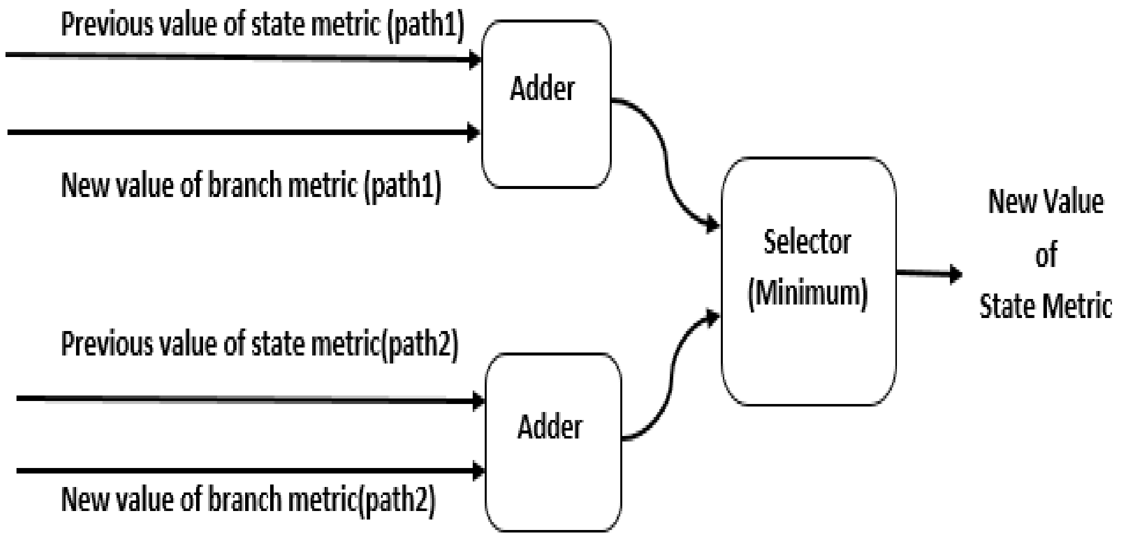

Previous value of state metric (path1)
New value of branch metric (path1)
Adder
Previous value of state metric(path2)
New value of branch metric(path2)
Adder
Selector
(Minimum)
New Value
of
State Metric

Figure 2.10 The Add Compare Unit [3].

The survivor path may be determined by passing through the minimal value of the state metric after updating a stat metric for each node in the trellis. Once this is done the partial path metric would be updated and the survivor data would be saved. The survivor management unit keeps track of the information bits linked with the surviving pathways. The trace back approach keeps track of which state's surviving branch was recorded. Each state has a flip-flop to record '0' or '1' depending on surviving path. The decoded output sequence will be formed by putting the decoded output bits together in reverse order of when they were decoded [17].

2.10 Types of Viterbi Decoding

2.10.1 Hard Viterbi Decoding

The Hamming distance metric is used in hard decision decoding process to identify a path via the trellis structure. As a result, the path across the trellis with the shortest Hamming distance is the most efficient path. When a symbol is seen at a decoder and when a symbol is delivered from an encoder a hamming distance may be defined as the number of bit that are different between symbols. Additionally, hard decision decoding algorithm quantize the received information one bit at a time [1].

Consider the following case study: assume that the transmitted information sequence is the all zero sequence i.e. m=[0 0 0 0 0]. The resulted codeword using the encoder given by figure 2.2 is given by c=[00 00 00 00 00]. Assuming that the sequence rH = [00 10 01 01 00] is received at the decoder. Implementing the Hard Viterbi decoding gives the decoded sequence $\hat{m} = [0\ 1\ 0\ 0\ 0]$.

Note: • That the decoded sequence has an error in the second bit.

• The x means the bath is cancelled

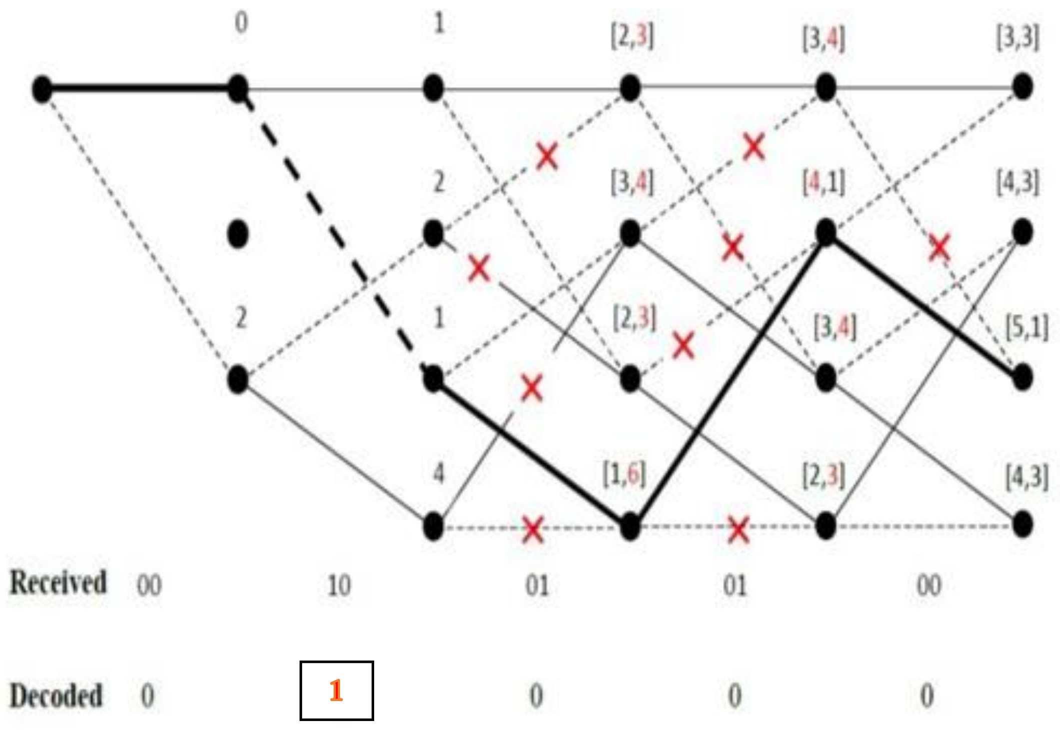


Figure 2.11 The trellis of hard decoding [12]

2.10.1 SOFT Viterbi Decoding

There are two techniques to determine distance in the Viterbi Algorithm to determine most probable path. A first is hamming distance that is associated with hard decision. The second one is the Euclidean distance which is associated with a soft choice. When data is broadcast across a Gaussian channel maximum likelihood decoding is performed via soft-decision decoding [1].

In contrast to hard decision decoding soft decision decoding employs multi-bit quantization. for received bit and Euclidean distance as the distance. measure rather than Hamming distance. The demodulator input is now analog waveform that is generally quantized into different levels to enable the decoder decide more readily. The quantization of 3 bits yields 8 output values.

Using Euclidean distance to compute straight line distance.

$$BM(x, r) = \sum_{i=1}^{n} (x_i - r_i)^2 \tag{2.8}$$

Case study:

Encode data sequence m = [0 0 0 0 0 …]. The coded sequence will be c =[-1 -1, -1 -1, -1 -1, -1 -1, -1 -1]. Given that the received sequence is r_s =[-1.5 -2.2,0.1 -3.1,-1.4 0.2,-2.5 0.1,-3.3 -4.2].

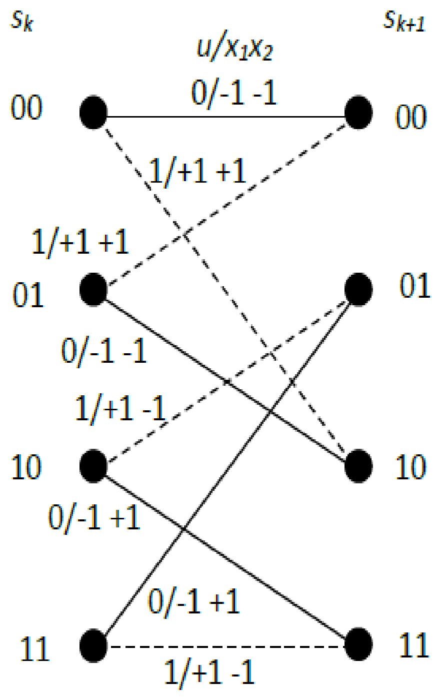


Figure 2.12 The Trellis with bipolar coded [12].

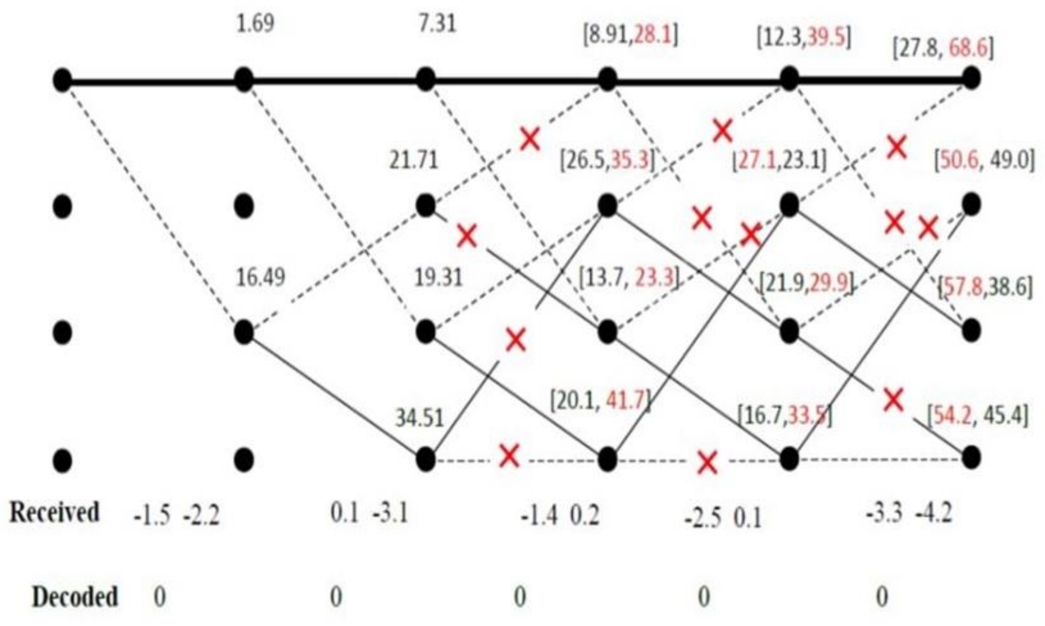


Figure 2.13 The trellis of soft decoding [12].

In the case of soft when calculating the metric, we calculate the Euclidean distance, which takes the correct difference between the sent code and the o/p on the trellis after converting the (0,1) to (-1,+1).

As for the hard, we calculate the Hamming distance after we compare the transmitted bits and the Threshold detection, and thus errors will occur.

2.11 Limitation of Viterbi Algorithm

VA decodes binary. convolutional code. with k=1 and constraint' length L into $2^{(L-1)}$ states. Convolutiona1 coding creates a trellis' with $2^{k\,(L-1)}$ states when k bits are moved into a shift register with L stages.

Decoding such a code using VA involves keeping. track. of $2^{k\,(L-1)}$ surviving paths and $2^{k\,(L-1)}$ metrics. During each step of the trellis there are 2^k paths that join at each node. The number of metrics for each node is 2^k because each path that comes to a common node needs to be computed. One of the2^k paths that merge at each node survives and it is a shortest path. A amount of decoding computations each step rises exponentially with k and L. Convolutional codes with large constraint lengths are unfeasible due to exponential increases in computation and storage [1].

2.12 Applications of Convolutional Codes

Convolutional codes are commonly used in satellite communications. The AWGN model provides a reasonable representation of the noise characteristic of communications to geostationary satellites provided that the earth station is well inside the antenna footprint. However, there are fluctuations with time dependent on air conditions. The major app1ication is digita1 speech and the BER is 10^{-5}. This makes it appropriate for convo1utiona1 codes with rate 1/2, K = 7 [18].

In cellular mobile communications the channel characteristics are less favorable due to burst errors caused by multipath (reflections) signal shadowing, and channel interference (reuse of the same frequency in different cells) yet the requirement for coding gain at reasonable target bit error rates again demands the use of convolutional codes. Constructed to perform well with bit error rates of 10^{-3} and higher voice coders (vocoders) are built for hostile channel environments.

The GSM standard for digital mobile communications uses a time division multiple access (TDMA) technology with a bit rate of 22 800 bits per second per channe1. The main application is digita1 voice using vocoders that can provide acceptab1e quality even with bit errors of 1% or higher. If you want to get a 1ot of coding gain at this level you need to use convolutiona1 codes with interleaving to protect against channel error bursts. The code is 1/2 rate, k = 5 [18].

Chapter three

FPGA ARCHITECTURE AND DESIGN FLOW

3.1 Introduction

FPGA stands for field-programmable gate array and refers to a semiconductor device with programmable logic blocks and interconnects. Logic blocks can be designed to execute simple logic operations like AND and XOR, or more complicated tasks like decoders or mathematical calculations. [19]

In most FPGA logic blocks, there is some kind of memory component. These could be simple flip-flops or bigger memory blocks. Programmable interconnects allow logic blocks to be linked together in any way the system designer wants. After production the client or designer can program logic blocks and interconnects to implement any logical function thus the name "field-programmable". [20]

3.2 FPGA Configuration Methods

The FPGA can be set up on different platforms. Among them are the following: [22]

(A) System generator (MATLAB)
(B) VHDL/VERELOG (ISE and VIVADO)
(C) C/C++/System C (HLS/VIVADO/SDK)

3.3 FPGA architecture

In comparison to other devices the primary characteristic that separates FPGAs is that their underlying fabric is made up of vast numbers of relatively basic programmable logic block "islands" located in a "sea" of programmable interconnects (see Figure; 3.3). [20]

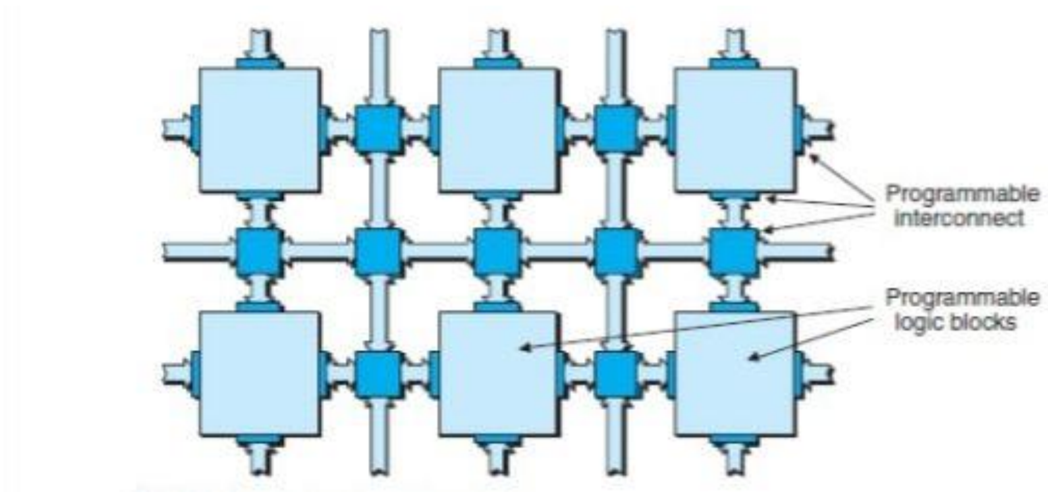


Figure 3.3. Underlying FPGA fabric [20].

A logic cell (LC) is the basic building piece of a contemporary Xilinx FPGA (a logic element (LE) in an Altera FPGA).

An LC consists of the following components (see Figure 3.4):

- 4-input. LUT (also known as a 16×1 RAM or a 16-bit shift register).
- Multiplexer.
- Register (It may be designed in accordance as a latch or a flip-flop). [23]

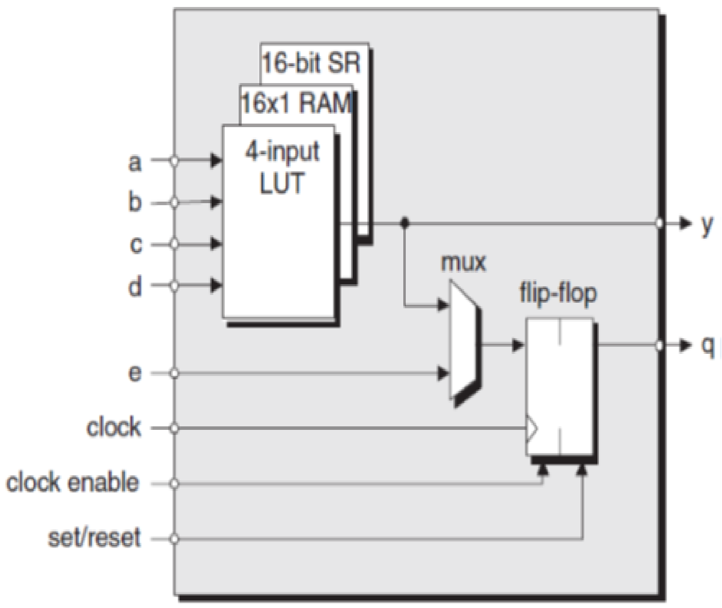


Figure 3.4: A simplified representation of a Xilinx LC.

A slice is the next Xilinx step up the hierarchy. Figure 3.5 shows two logic cells in a slice. Both logic cells' LUT, MUX, and register have its own data inputs and outputs, but the slice shares clock enables and set/reset signals. [24]

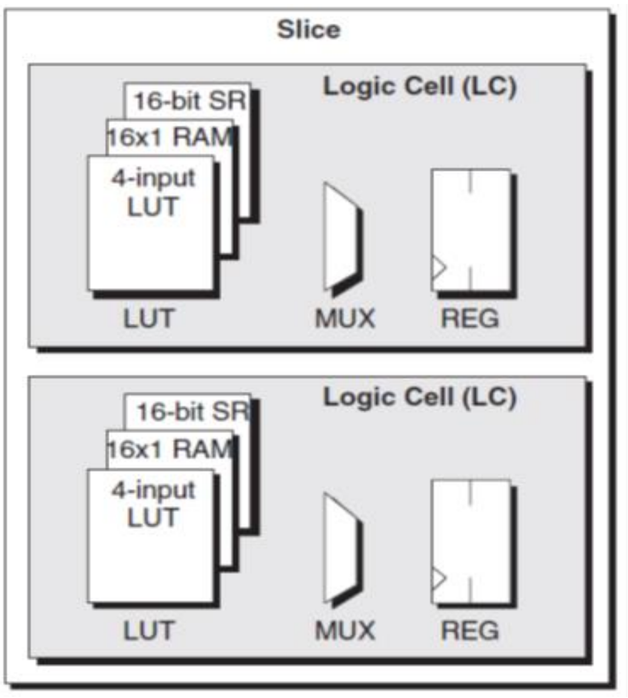


Figure 3.5: A slice with two logic cells.

Continuing up the hierarchy one step, we reach what Xilinx refers to as a customizable logic block (CLB) and Altera refers to as a logic array block (LAB).

Each CLB in certain Xi1inx FPGAs has two slices, whereas others have four. Figure 3.6 depicts a CLB as a single logic block in the "islands" of programmable logic in a "sea" of programmable interconnect.

The CLB also has some rapid programmable connections. This interconnect connects adjacent slices.

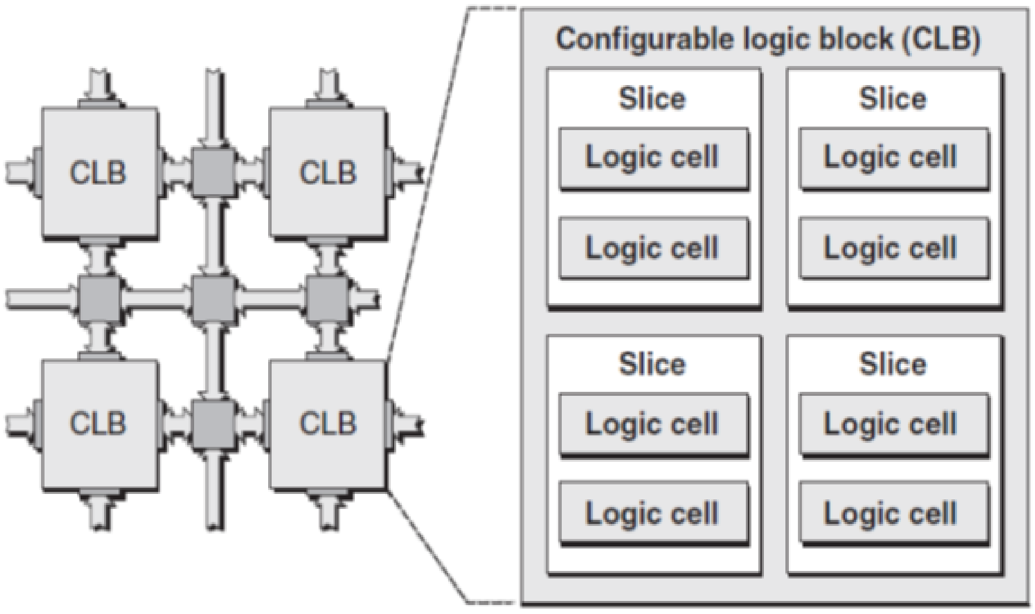


Figure 3.6: A CLB with 4 slices is depicted (The number of slices is determined by the FPGA family).

The purpose for using this form of logic-block hierarchy: LC S1ice (two LCs) and CLB (four slices) is to ensure that it is supplemented by an analogous hierarchy in the interconnect. The connection between LCs in a slice is quick then the interconnect between CLBs is slightly slower. The goal is to find the best balance between making interconnections simple and avoiding excessive interconnect delays. [22]

3.4 FPGA Implementation

The Xilinx Artix-7 (Family) FPGA with the Xilinx product number XC7A100T-1CSG324C, is utilized to implement the circuit in its final form. The Vivado HLS 2019 design environment and tool is used for the design process. The C++ code's FPGA Design flow is depicted by figure 3.7 [1].

3.4.1 Overview of FPGA Design Flow

The complexity of the FPGA architecture rises as it grows. FPGA suppliers now provide a comprehensive set of design tools that allow automated synthesis. and compilation of hardware specifications, in C/C++ system

C/Verilog or VHDL down to a bit stream for FPGA programming. Figure 3.7 depicts a typical FPGA design flow. The design flow normally takes as inputs the HDL definition, design restrictions, and target FPGA device specification [24].

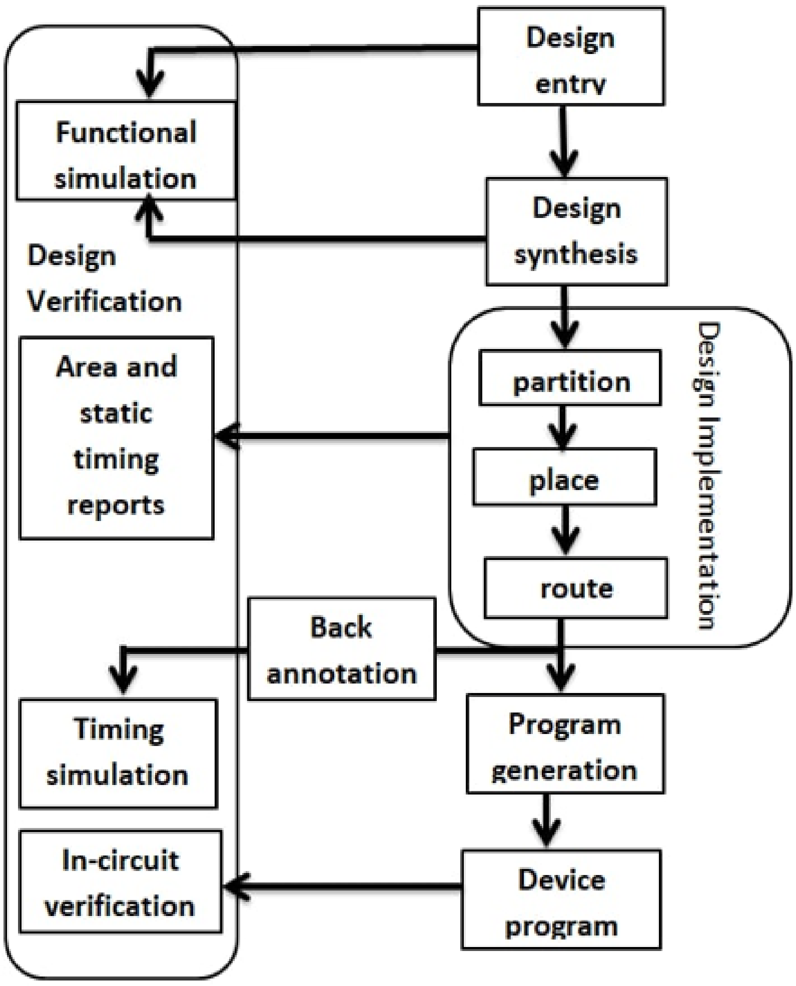
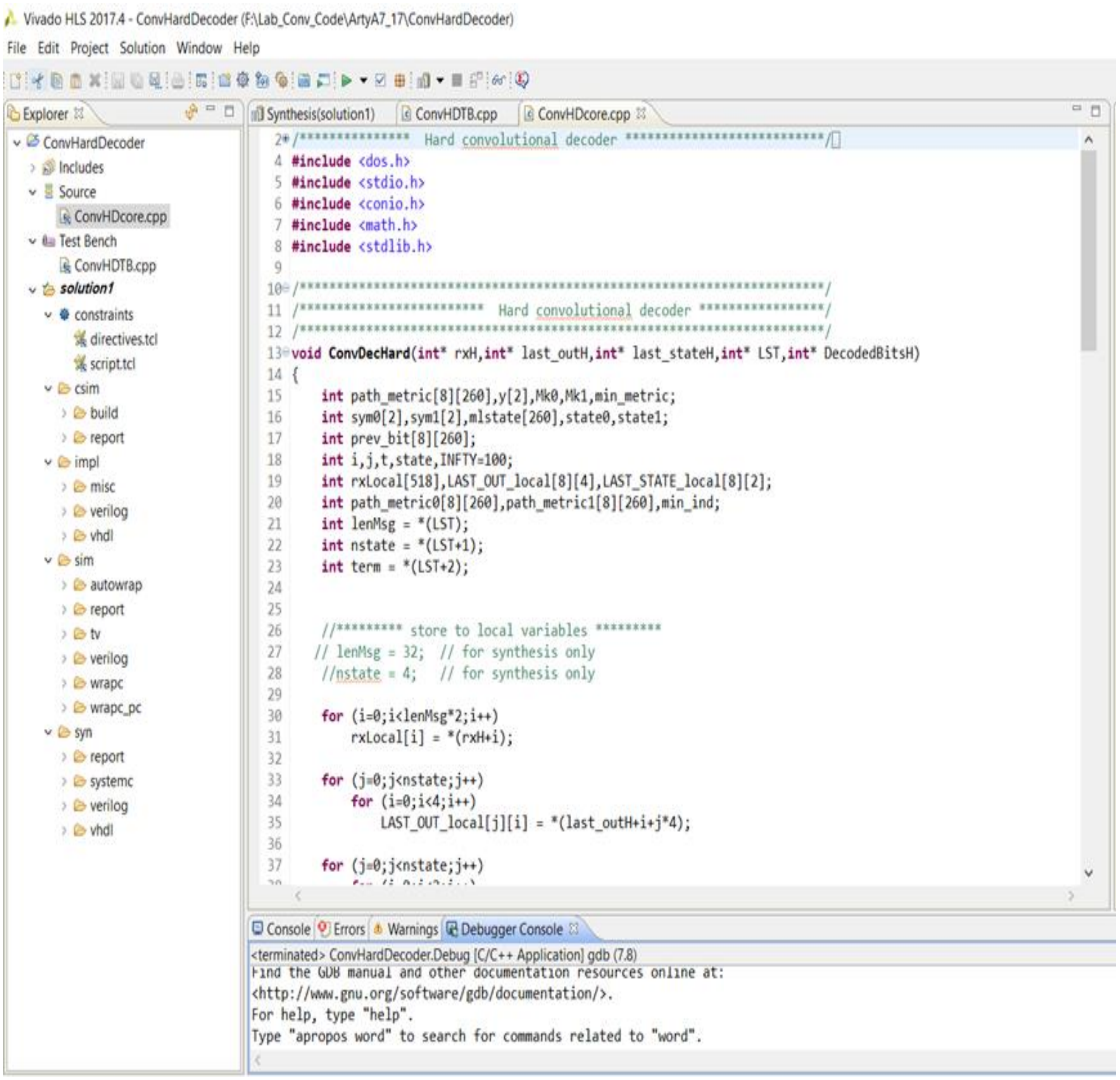


Figure 3.7: Flow for FPGA Design [24]

3.4.2 Design Entry

This part involves the design of the system's fundamental architecture, which is developed in C++ and implemented using Vivado HLS. The goal of using a high-level abstract language is to limit the time to market and make a complicated project easier to manage. The IP block with input/output ports represents the C++ function parameter. For the C++ entry, a Verilog or VHDL is also produced [25].

Figure 3.8 shows the HLS integrated development environment (IDE) with the code of the core function “ConvDecHard” is partially shown on. This function simulate the hard Viterbi decoder.



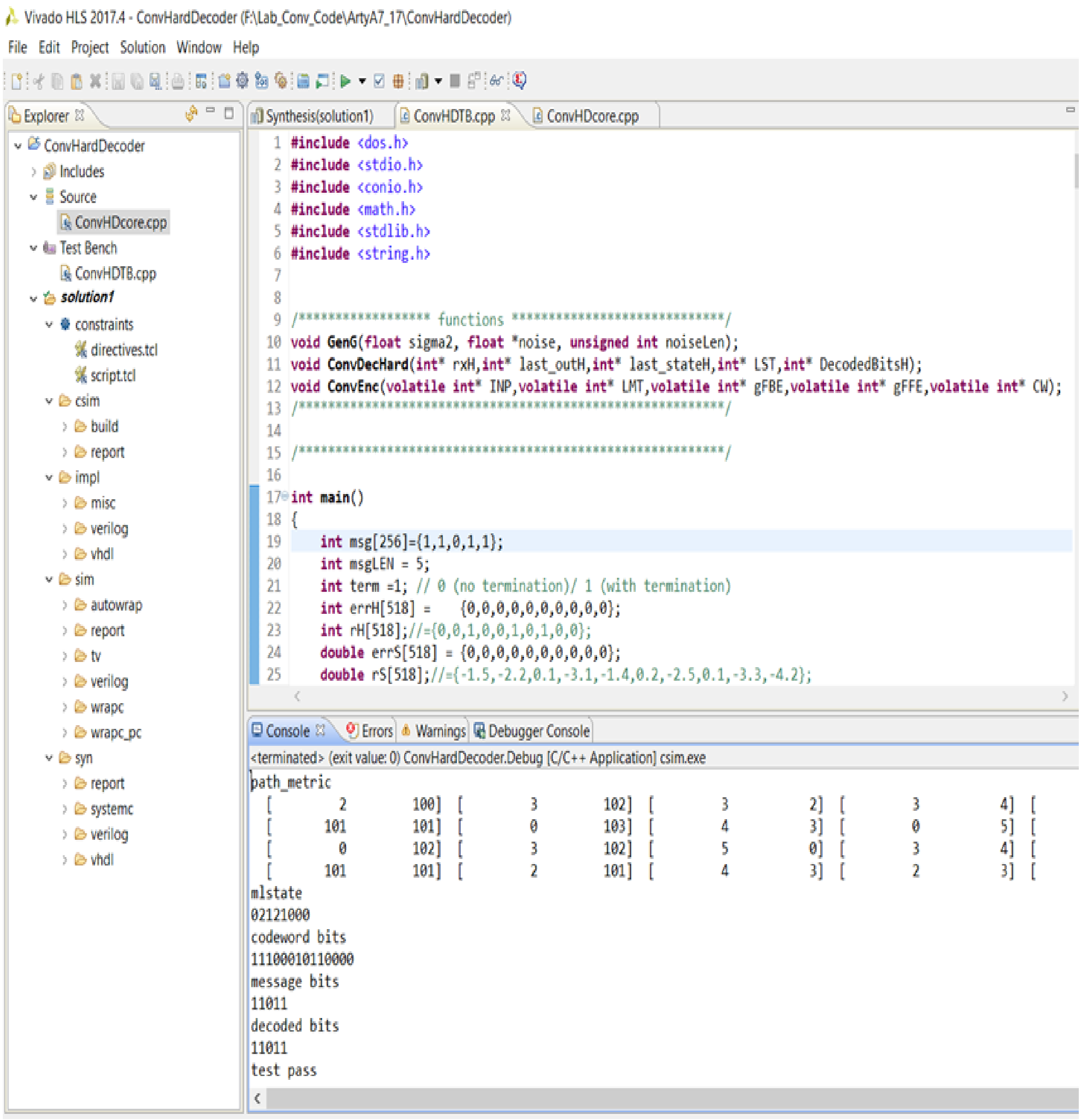
```
/***************  Hard convolutional decoder ***************************/
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>

/************************************************************************/
/************************  Hard convolutional decoder *****************/
/************************************************************************/
void ConvDecHard(int* rxH,int* last_outH,int* last_stateH,int* LST,int* DecodedBitsH)
{
    int path_metric[8][260],y[2],Mk0,Mk1,min_metric;
    int sym0[2],sym1[2],mlstate[260],state0,state1;
    int prev_bit[8][260];
    int i,j,t,state,INFTY=100;
    int rxLocal[518],LAST_OUT_local[8][4],LAST_STATE_local[8][2];
    int path_metric0[8][260],path_metric1[8][260],min_ind;
    int lenMsg = *(LST);
    int nstate = *(LST+1);
    int term = *(LST+2);


    //********* store to local variables *********
   // lenMsg = 32;  // for synthesis only
    //nstate = 4;   // for synthesis only

    for (i=0;i<lenMsg*2;i++)
        rxLocal[i] = *(rxH+i);

    for (j=0;j<nstate;j++)
        for (i=0;i<4;i++)
            LAST_OUT_local[j][i] = *(last_outH+i+j*4);

    for (j=0;j<nstate;j++)
```

```
<terminated> ConvHardDecoder.Debug [C/C++ Application] gdb (7.8)
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
```

Figure 3.8: the development of core function ConvDecHard in HLS IDE.

3.4.3 Behavioral Simulation

Upon completion of the design phase, build test bench code that contains input stimuli for the purpose of verifying the functioning of the C++ code. If everything checks out, you can go on; if not, updates and required fixes will be made to the C++ code. The behavioral simulation is the term used to describe this [1]. Figure 3.9 shows the HLS IDE with the test bench code “ConvHDTB” and the simulation results of the core function. The test bench calls the core function with the necessary data and return the output and compare it with golden data.

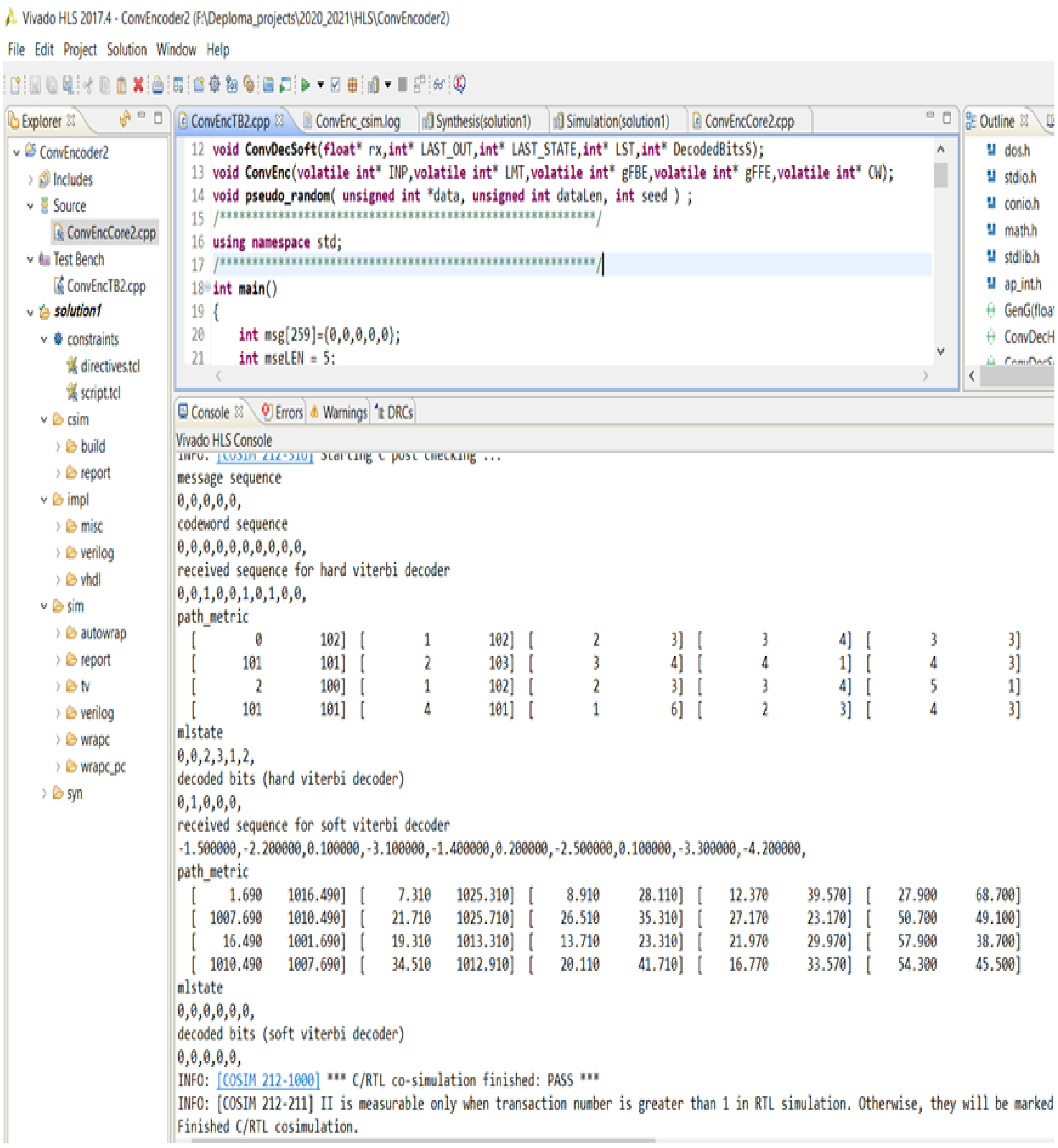


```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>


/***************** functions ***************************/
void GenG(float sigma2, float *noise, unsigned int noiseLen);
void ConvDecHard(int* rxH,int* last_outH,int* last_stateH,int* LST,int* DecodedBitsH);
void ConvEnc(volatile int* INP,volatile int* LMT,volatile int* gFBE,volatile int* gFFE,volatile int* CW);
/**********************************************************/

/**********************************************************/

int main()
{
    int msg[256]={1,1,0,1,1};
    int msgLEN = 5;
    int term =1; // 0 (no termination)/ 1 (with termination)
    int errH[518] =    {0,0,0,0,0,0,0,0,0,0};
    int rH[518];//={0,0,1,0,0,1,0,1,0,0};
    double errS[518] = {0,0,0,0,0,0,0,0,0,0};
    double rS[518];//={-1.5,-2.2,0.1,-3.1,-1.4,0.2,-2.5,0.1,-3.3,-4.2};
```

```
<terminated> (exit value: 0) ConvHardDecoder.Debug [C/C++ Application] csim.exe
path_metric
  [         2         100]  [         3         102]  [         3          2]  [         3          4]  [
  [       101         101]  [         0         103]  [         4          3]  [         0          5]  [
  [         0         102]  [         3         102]  [         5          0]  [         3          4]  [
  [       101         101]  [         2         101]  [         4          3]  [         2          3]  [
mlstate
02121000
codeword bits
11100010110000
message bits
11011
decoded bits
11011
test pass
```

Figure 3.9: the test bench function ConvHDTB and the resulted output.

Figure 3.10 present the RTL/Cosimulation results which represent the execution of the HDL code. Both the C simulation and RTL Cosimulation are matched, which means that the resulted IP will work well. The simulations are coinciding with the results obtained in section 2.10.1 and 2.10.2.



```
void ConvDecSoft(float* rx,int* LAST_OUT,int* LAST_STATE,int* LST,int* DecodedBitsS);
void ConvEnc(volatile int* INP,volatile int* LMT,volatile int* gFBE,volatile int* gFFE,volatile int* CW);
void pseudo_random( unsigned int *data, unsigned int dataLen, int seed ) ;
/************************************************************/
using namespace std;
/************************************************************/
int main()
{
    int msg[259]={0,0,0,0,0};
    int mseLEN = 5;
```

```
INFO: [COSIM 212-510] Starting C post checking ...
message sequence
0,0,0,0,0,
codeword sequence
0,0,0,0,0,0,0,0,0,0,
received sequence for hard viterbi decoder
0,0,1,0,0,1,0,1,0,0,
path_metric
  [        0        102] [        1        102] [        2          3] [        3          4] [        3          3]
  [      101        101] [        2        103] [        3          4] [        4          1] [        4          3]
  [        2        100] [        1        102] [        2          3] [        3          4] [        5          1]
  [      101        101] [        4        101] [        1          6] [        2          3] [        4          3]
mlstate
0,0,2,3,1,2,
decoded bits (hard viterbi decoder)
0,1,0,0,0,
received sequence for soft viterbi decoder
-1.500000,-2.200000,0.100000,-3.100000,-1.400000,0.200000,-2.500000,0.100000,-3.300000,-4.200000,
path_metric
  [    1.690   1016.490] [    7.310   1025.310] [    8.910     28.110] [   12.370     39.570] [   27.900     68.700]
  [ 1007.690   1010.490] [   21.710   1025.710] [   26.510     35.310] [   27.170     23.170] [   50.700     49.100]
  [   16.490   1001.690] [   19.310   1013.310] [   13.710     23.310] [   21.970     29.970] [   57.900     38.700]
  [ 1010.490   1007.690] [   34.510   1012.910] [   20.110     41.710] [   16.770     33.570] [   54.300     45.500]
mlstate
0,0,0,0,0,0,
decoded bits (soft viterbi decoder)
0,0,0,0,0,
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise, they will be marked
Finished C/RTL cosimulation.
```

Figure 3.10 RTL/ Cosimulation for the core function ConvEnc.

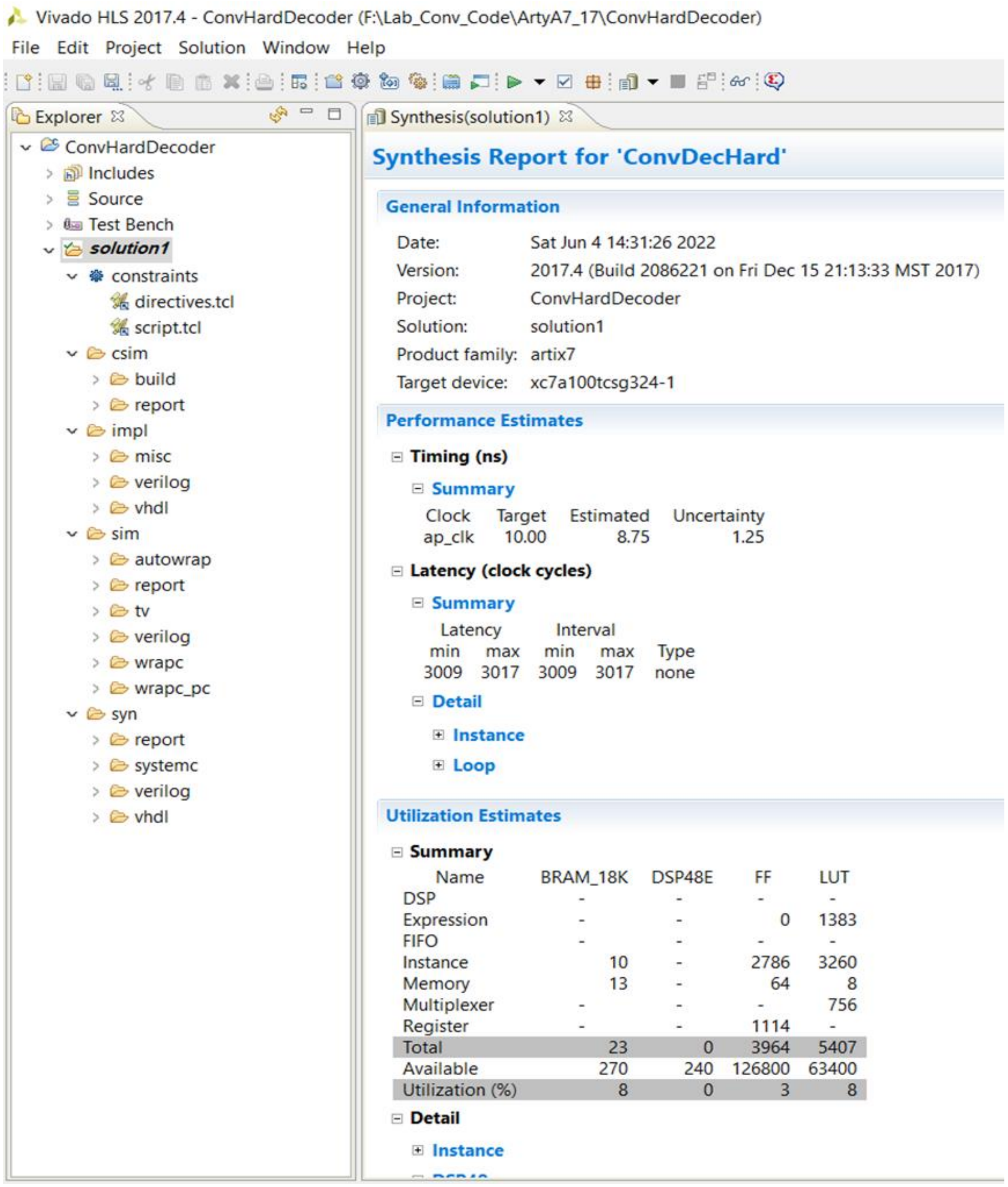
3.4.4 Design Synthesis

It is only after the correct simulations are done that the design is made. Xilinx vivado performs the following tasks during synthesis:

HDL Compilation: If there are several sub-functions in the main function, the tool compiles them all and then examines the syntax of the code created for the design.

Design Hierarchy Analysis: A look at how things are organized in the design.

HDL Synthesis: When you write VHDL or Verilog code, it is turned into a device netlist, which is a complete circuit with logical elements like Multiplexers, Adders, Subtractors, Counters, Flip-Flops and so on. Synthesis will examine code syntax and design hierarchy to ensure the design is optimum for designer's chosen design architecture. The generated netlist is NGC'(Native Generic Circuit) [20]. Figure 3.11 shows the synthesis report for the ConvDecHard function. The report also includes the latency and initiation interval in terms of clock cycles.



Synthesis Report for 'ConvDecHard'

General Information

Date:	Sat Jun 4 14:31:26 2022
Version:	2017.4 (Build 2086221 on Fri Dec 15 21:13:33 MST 2017)
Project:	ConvHardDecoder
Solution:	solution1
Product family:	artix7
Target device:	xc7a100tcsg324-1

Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.75	1.25

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
3009	3017	3009	3017	none

⊟ Detail

⊞ Instance

⊞ Loop

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1383
FIFO	-	-	-	-
Instance	10	-	2786	3260
Memory	13	-	64	8
Multiplexer	-	-	-	756
Register	-	-	1114	-
Total	23	0	3964	5407
Available	270	240	126800	63400
Utilization (%)	8	0	3	8

⊟ Detail

⊞ Instance

Figure 3.11 synthesis report for the ConvDecHard function.

3.4.5 Design Implementation

Sub-processes of the design implementation process include:

Translation: Input netlists and restrictions are combined into a logic design file. NGD (Native Generic Database) files include this information. In the NGD file, you can see how the logical design is reduced to Xilinx device primitive cells. Defining. restrictions mean allocating ports to physical elements (pins, switches, buttons, etc.) and providing design time requirements. This data is saved in UCF (User Constraints File). UCF is generated or edited using tools such as PACE, Constraint Editor, and others. [1]

Mapping: After Translate step, the Map process begins. The map procedure partitions the whole circuit containing logical elements to sub blocks; that could be accommodated in the FPGA logic blocks. That is the map operation converts the logic represented in the "NGD" fi1e to the specific FPGA elements, (Combinational Logic Blocks (CLB) and Input Output Block (IOB)) and genrate an NCD (Native Circuit, Description) file that actually represents a design translated to the. FPGA .components.[20]

Place and Route:A program called Place and Route. (PAR) is used to do this procedure. The place and route. procedure converts a map sub blocks into logic blocks and links them. For example placing a sub block near an IO pin in a logic block may reduce time but may effect other constraints. The place and route process considers the tradeoff between all limitations. When you use the PAR tool, you start with a mapped NCD file and end up with a completely routed NCD file.

Bitstream Generation: A "bitstream" is a set of binary data that is used to program. a reconfigurable logic device, however this is deceptive due to the fact that the data is not bit oriented and there is no "streaming." While configuration data are continually delivered into internal units of an instruction set processor, they are normally put into reconfigurable logic devices just once during initial setup.

Functional Simulation: Before you map the design, you can run a post-translate (functional) simulation. This simulation method lets users to check that the design has been synthesized successfully and to identify any changes that may have occurred as a result of the lower level of abstraction.

3.5 Pipelining

This method analyzes a sequential process into sub-processes, each running in a separate segment that runs parallel with all others. Pipelining divides a process into portions that may run in parallel. The process of pipelining is separated into segments, each of which may perform its operation simultaneously with the operations of the other parts. The outputs of each operation appear rapidly towards the pipeline's finish. [1]

The pipeline method is commonly used to increase digital circuit performance. It is shown that increasing the number of pipeline stages results in a reduction in the path de1ays of each. Stage and an improvement in the overall performance of' the circuit. An operation pipeline is a design method used to boost the throughput of computers and other digital electronic devices (the number of instructions that can be executed in a unit of time). [23]

The basic idea is to break down a computer operation into distinct parts with storage at each step. The delay before the logic delivers valid outputs is decreased by dividing it up into smaller chunks and putting flip flips between them. When the c1ock signa1 comes the f1ip f1ops update their values and a 1ogic must decode them. Afterwards, when the next c1ock pu1se occurs the f1ip f1ops are reset to their original values and etc [1].

Following is a graph showing how function pipelining improves throughput (3.12).

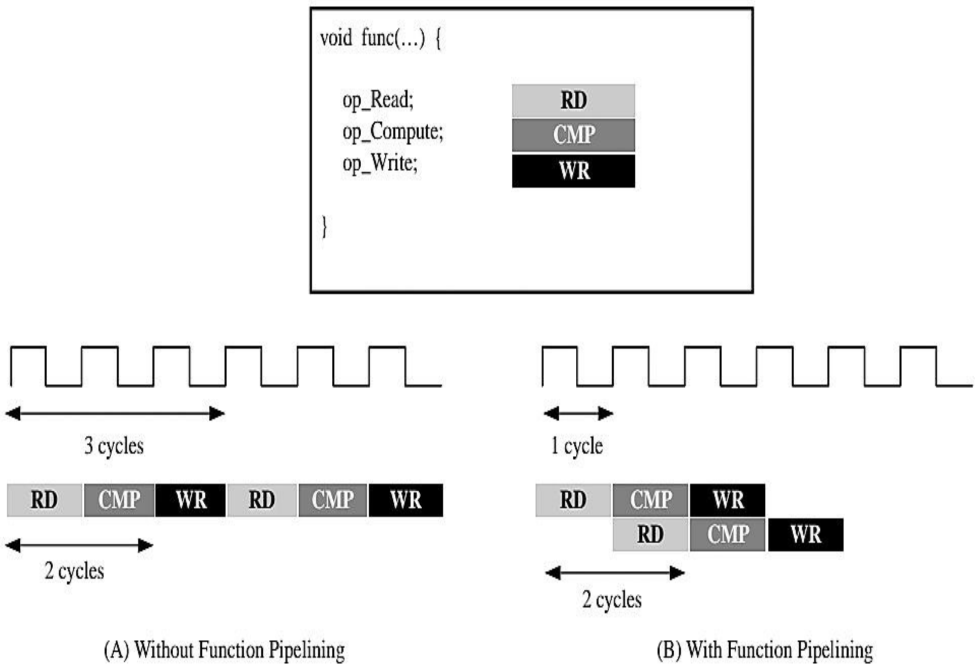
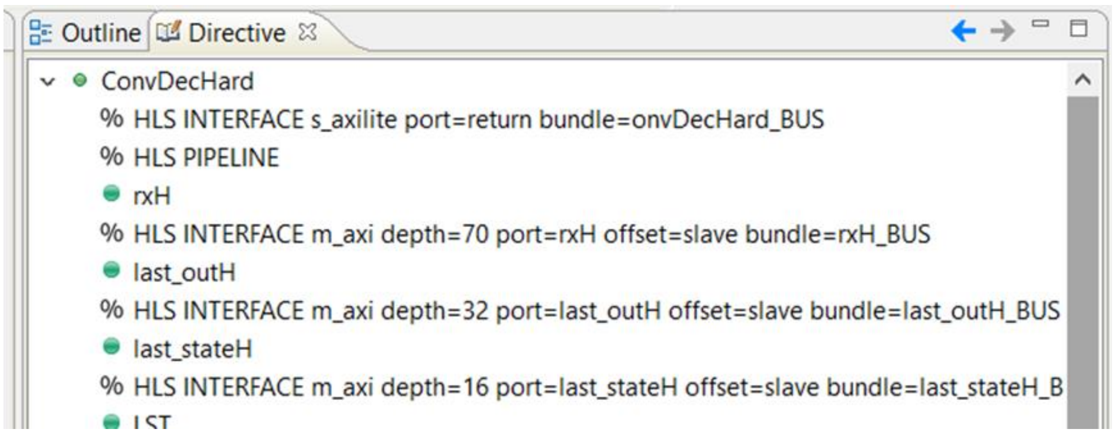


Figure 3.12: Behavior of Function Pipelining [23].

The function in the previous figure receives an input every 3 clock cycles and produces a va1ue every 2. Clock cycles without pipeline. The function has an II of 3 and a latency of 3. When using pipelining like in this figure each cycle (II=1) has a fresh input read, but there is no change in the latency of the output [23].

The pipeline can be added to the design as a directive to the core function as shown in figure 3.13.



```
ConvDecHard
    % HLS INTERFACE s_axilite port=return bundle=onvDecHard_BUS
    % HLS PIPELINE
    rxH
    % HLS INTERFACE m_axi depth=70 port=rxH offset=slave bundle=rxH_BUS
    last_outH
    % HLS INTERFACE m_axi depth=32 port=last_outH offset=slave bundle=last_outH_BUS
    last_stateH
    % HLS INTERFACE m_axi depth=16 port=last_stateH offset=slave bundle=last_stateH_B
    LST
```

Figure 3.13. Adding pipeline to the design.

3.6 Exporting the design as intellectual property (IP)

The last step in HLS work is to package the design as IP block and export it to vivado IP integrator where other blocks are added like microblaze, DDR RAM, UART and so on. Figure 3.14 shows the IP block for the ConvDecHard core function. The input and output arguments of the core function are implemented as physical ports.

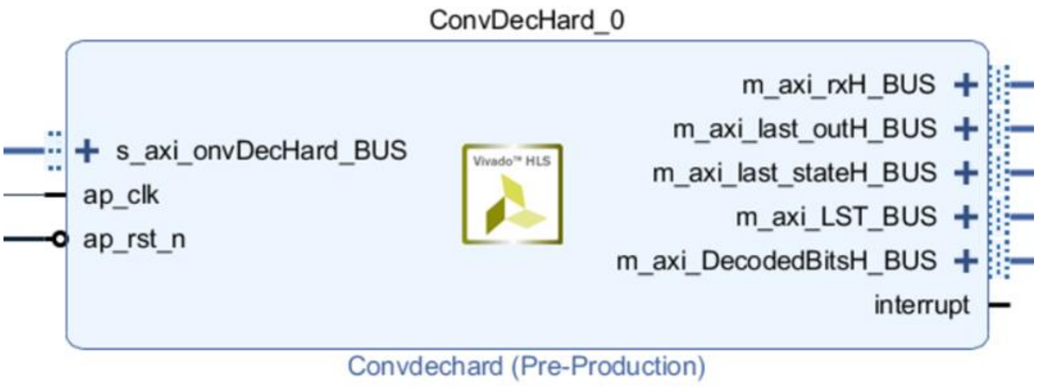


Figure 3.14 the IP block for the ConvDecHard core function.

3.7 FPGA Applications

There are several practical uses for the FPGA. A selection of them are listed below. [21]

3.7.1 Medical applications

(A) Diagnostic imaging systems

1- Ultrasound
2- X-ray
3- (CT) Computed tomography
4- (PET) Nuclear or positron emission tomography
5- (MRI) Magnetic resonance imaging

(B) Robotic surgical systems

Robotic assisted surgery with enhanced visual guiding and multi-axis control.

3.7.2 On-Board Artificial intelligent for Autonomous Space Exploration

NASA's Spirit and Opportunity rovers employed Xilinx's Virtex-4QV FPGA to examine Martian terrain. The V5QV gadget was also utilized for off-line processing on MARS2020. The FPGA's key tasks in these rovers are indicated by :

(i) Autonomous navigation
(ii) Reading telemetry data from sensors
(iii) Controlling actuators
(iv) Image processing

CHAPTER FOUR

HARDWARE AND SOFTWARE SIMULITION RESULTS

4.1 Implementation of convolutional encoder

The implementation of convolutional encoder starts by writing the function in C++ using the Vivado HLS. The input and output of the function as follows:

Inputs:

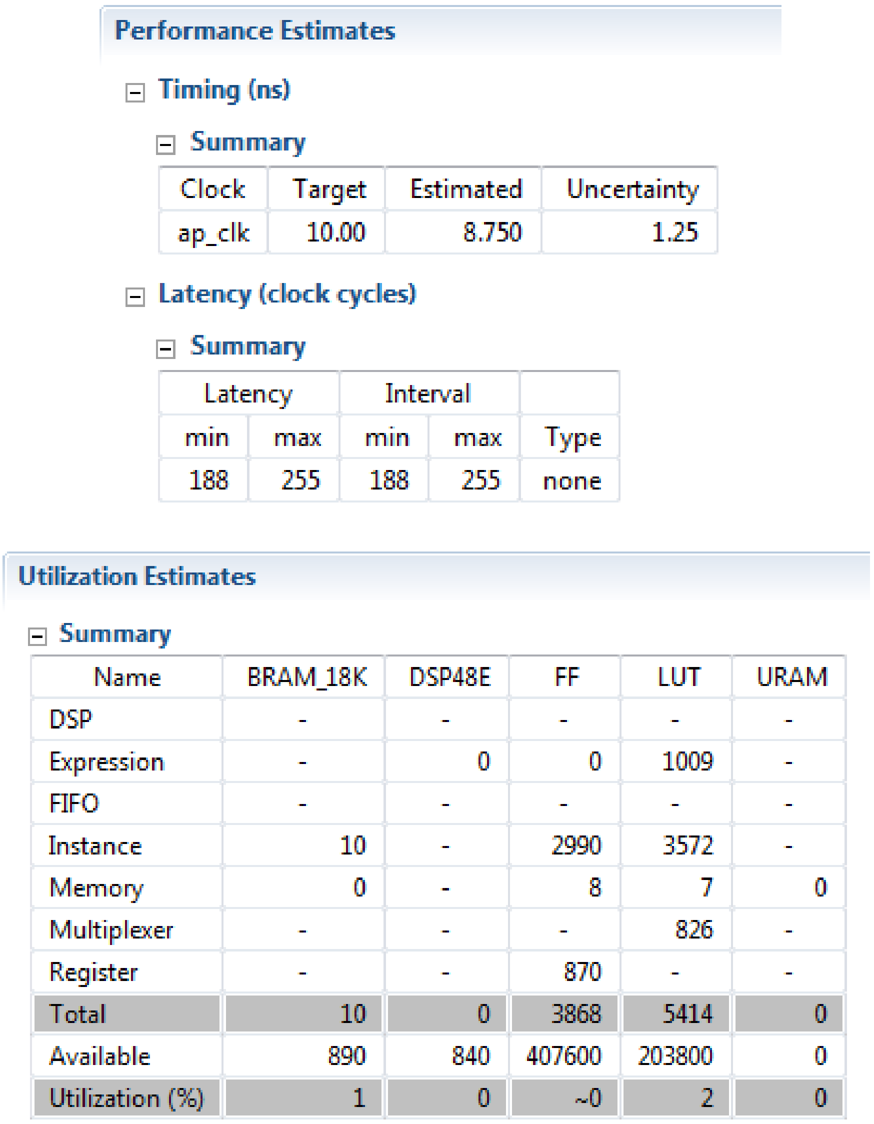
INP_BUS: the vector of input message.
gFBE_BUS: feedback generator polynomial.
gFFE_BUS: feedforward generator polynomial.
memE_BUS: memory length of convolutional encoder.
msgLen_BUS: message length.
TERM_BUS: trellis termination indicator.

Outputs:

CW_BUS: the vector of codeword

The function was examined inside the HLS program using the Test bench program. It was also written in the C++ language to call the function and execute it after entering the necessary data. Directives were used to direct the HLS program to implement the program to work in parallel to reduce the delay, which is the period required by the function to finish its mission and give results. The directive is also used to reduce the interval period, which is the time in which the function requests other inputs, as an indicator of the transmission speed. Table (4.1) gives an idea of the delay time before the use of directives and numbers represents the number of clocks, and each pulse has a delay time of 10 nanoseconds, as the frequency of the clocks for the processor is 100 megahertz. The table also shows the total storage capacity, Digital signal processor cores (DSP48E), Flip-Flop (FF), and Lockup Tables (LUT).

Table (4.1): delay time and required space from convolutional encoder circuit for solution 1.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
188	255	188	255	none

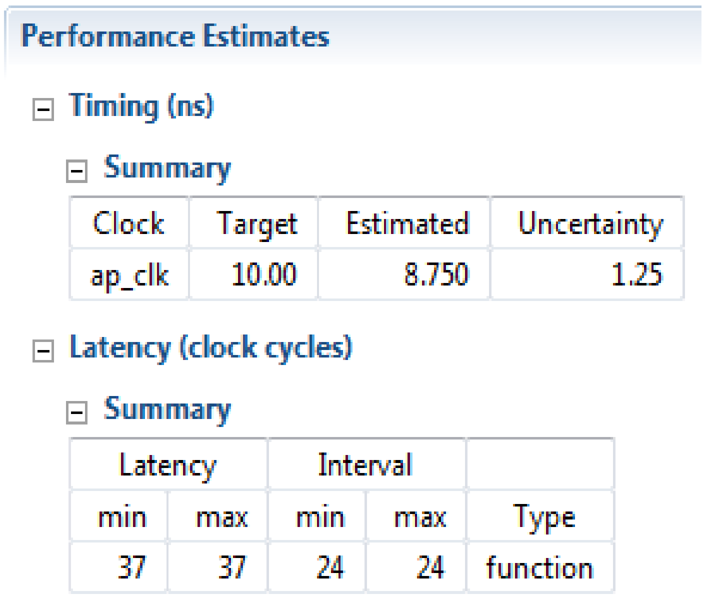
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	0	0	1009	-
FIFO	-	-	-	-	-
Instance	10	-	2990	3572	-
Memory	0	-	8	7	0
Multiplexer	-	-	-	826	-
Register	-	-	870	-	-
Total	10	0	3868	5414	0
Available	890	840	407600	203800	0
Utilization (%)	1	0	~0	2	0

With the implication of some directives like pipelining, one can achieve a significant improvement in delay and interval time as well as the need for total silicon area. Table (4.2) gives an idea about the delay and utilization after adding pipeline directives.

Table (4.2): delay time and required space with pipelining from convolutional encoder circuit for solution 2.



Performance Estimates

Timing (ns)

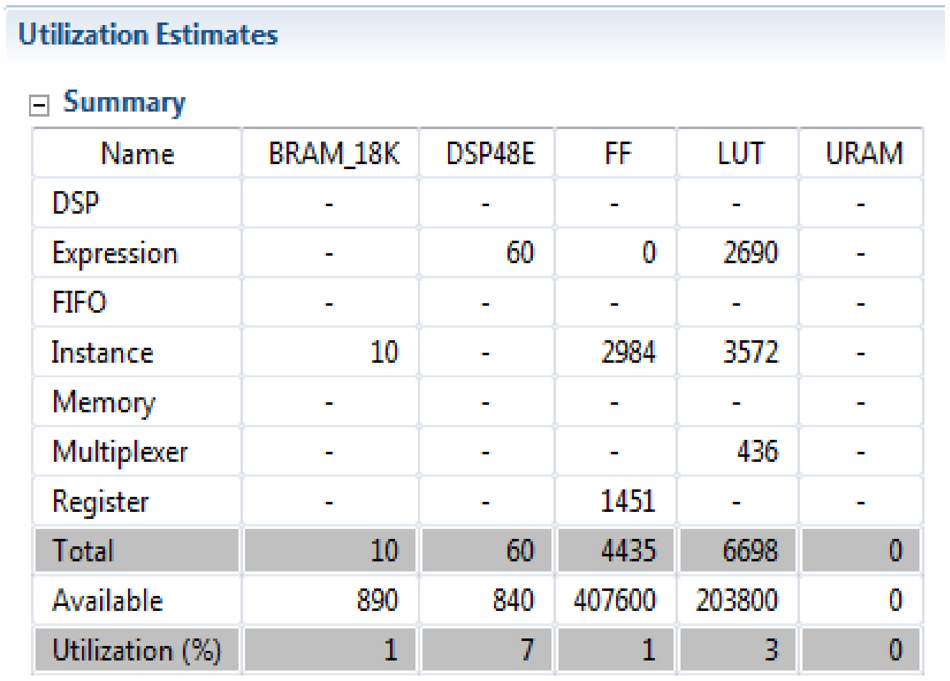
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
37	37	24	24	function

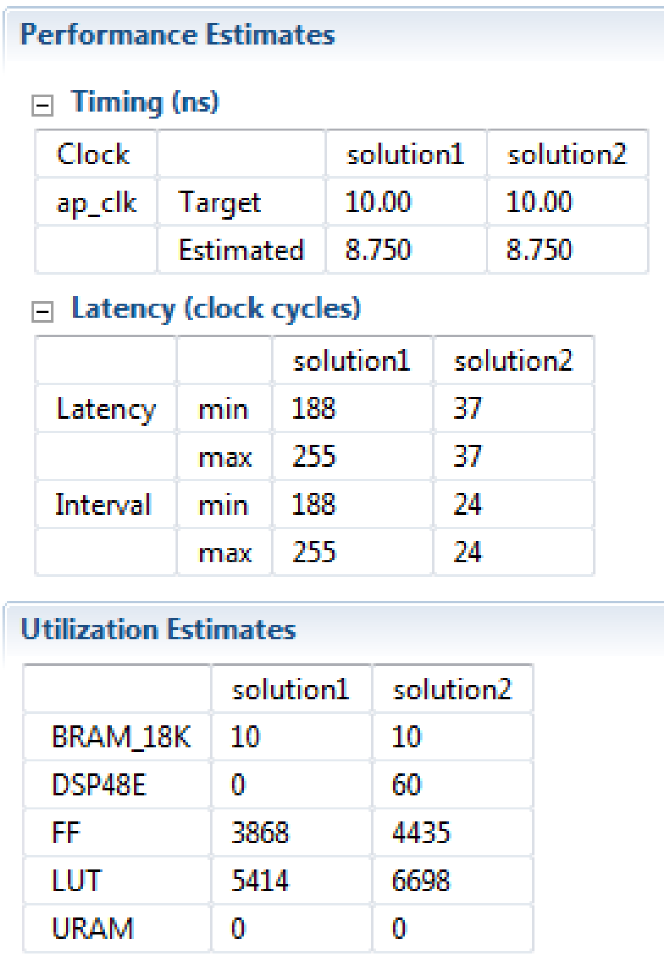


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	60	0	2690	-
FIFO	-	-	-	-	-
Instance	10	-	2984	3572	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	436	-
Register	-	-	1451	-	-
Total	10	60	4435	6698	0
Available	890	840	407600	203800	0
Utilization (%)	1	7	1	3	0

Table (4.3) comparison between solution 1 and 2.



Performance Estimates

Timing (ns)

Clock		solution1	solution2
ap_clk	Target	10.00	10.00
	Estimated	8.750	8.750

Latency (clock cycles)

		solution1	solution2
Latency	min	188	37
	max	255	37
Interval	min	188	24
	max	255	24

Utilization Estimates

	solution1	solution2
BRAM_18K	10	10
DSP48E	0	60
FF	3868	4435
LUT	5414	6698
URAM	0	0

The HLS can also produce intellectual property (IP) blocks to be used in the Vivado IP integrator. Figure 4.1 represents the IP generated from the convolutional encoder function process.

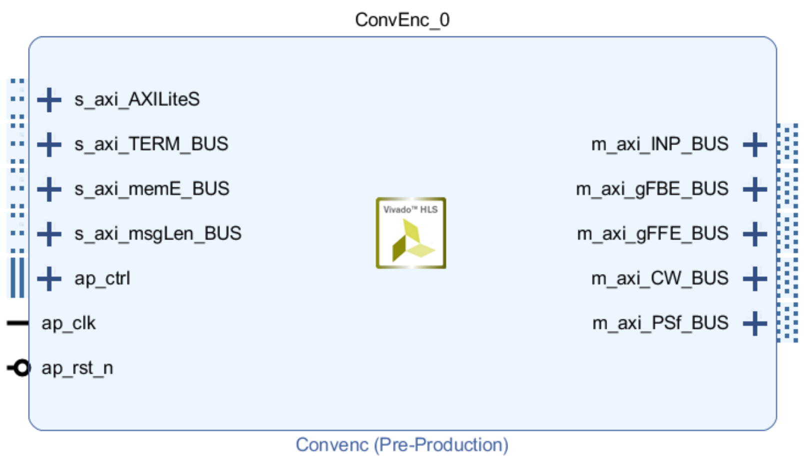


Figure 4.1 IP of the convolutional encoder

Figure (4.2) represent the flowchart of the convolutional encoder.

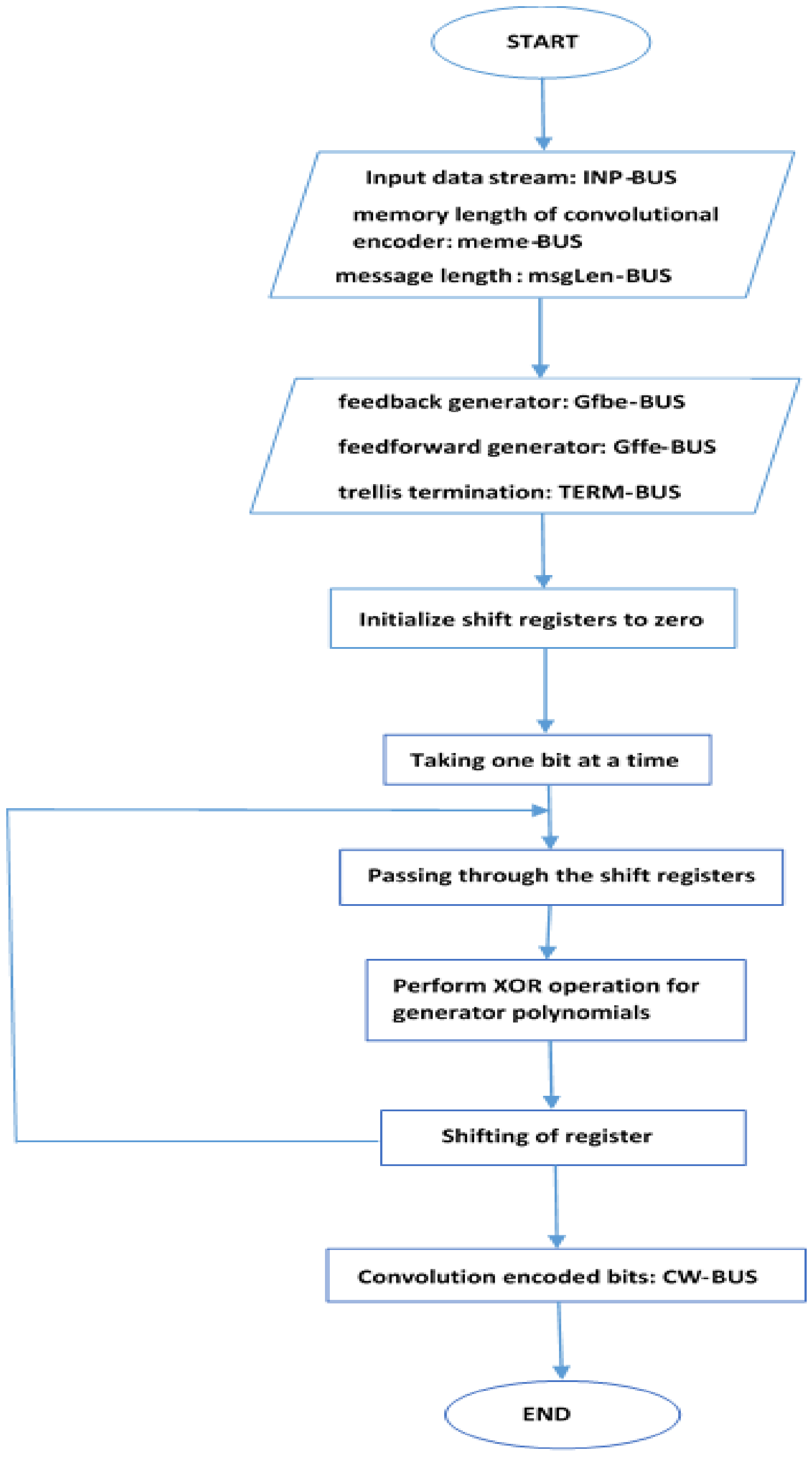

START
Input data stream: INP-BUS
memory length of convolutional encoder: meme-BUS
message length : msgLen-BUS
feedback generator: Gfbe-BUS
feedforward generator: Gffe-BUS
trellis termination: TERM-BUS
Initialize shift registers to zero
Taking one bit at a time
Passing through the shift registers
Perform XOR operation for generator polynomials
Shifting of register
Convolution encoded bits: CW-BUS
END

Figure 4.2 Flow Chart of convolutional encoder

4.2 Implementation of trellis

The trellis circuit was designed using C++ and utilizing the Vivado HLS platform. The input and output of the trellis function are:

Inputs:

gFBT_BUS : feedback generator polynomial.

gFFT_BUS : feedforward generator polynomial.

memT_BUS : memory length of convolutional encoder.

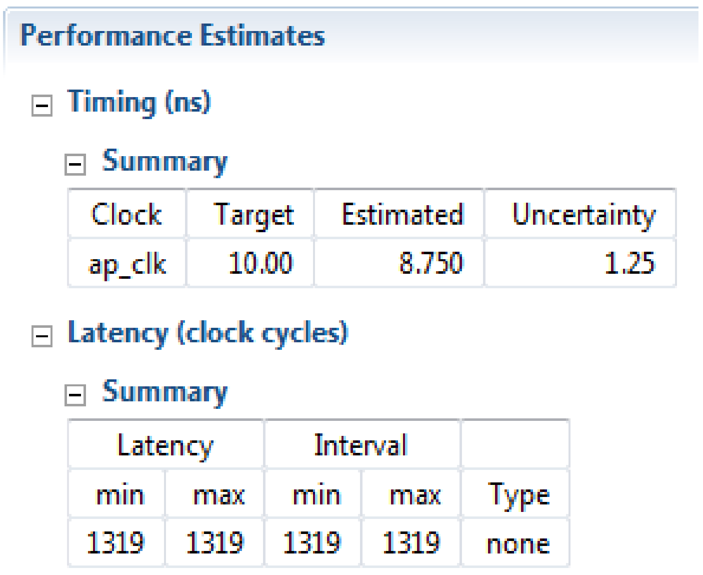
outputs:

last_outT_BUS : the last output

last_stateT_BUS : the last state

With the implication of some directives pipelining, one can achieve a significant improvement in delay and interval time as well as the need for total silicon area. Table (4.5) gives an idea about the delay and utilization after adding pipeline directives.

Table (4.4): delay time and required space from trellis circuit for solution1.



Performance Estimates

Timing (ns)

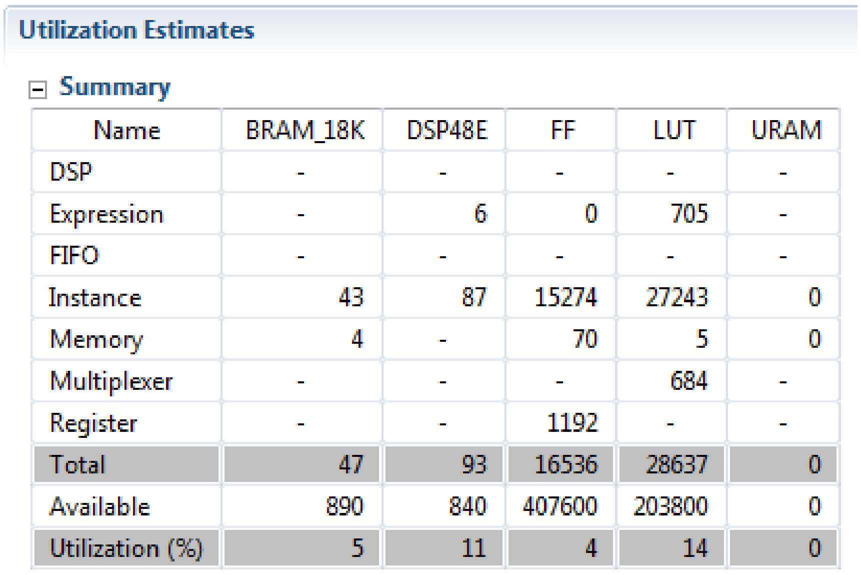
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1319	1319	1319	1319	none

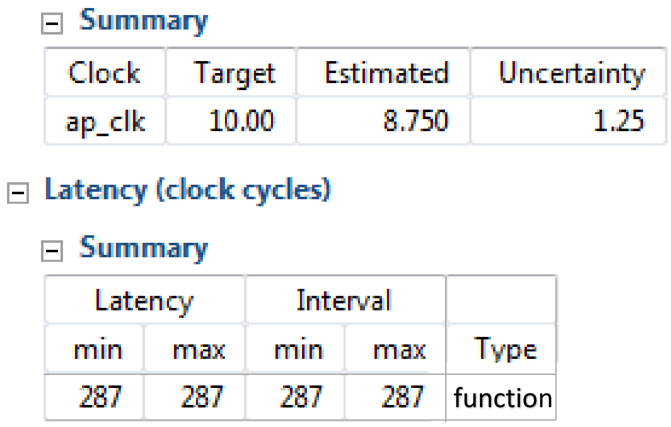


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	6	0	705	-
FIFO	-	-	-	-	-
Instance	43	87	15274	27243	0
Memory	4	-	70	5	0
Multiplexer	-	-	-	684	-
Register	-	-	1192	-	-
Total	47	93	16536	28637	0
Available	890	840	407600	203800	0
Utilization (%)	5	11	4	14	0

Table (4.5): delay time and required space with pipelining of trellis circuit for solution 2.



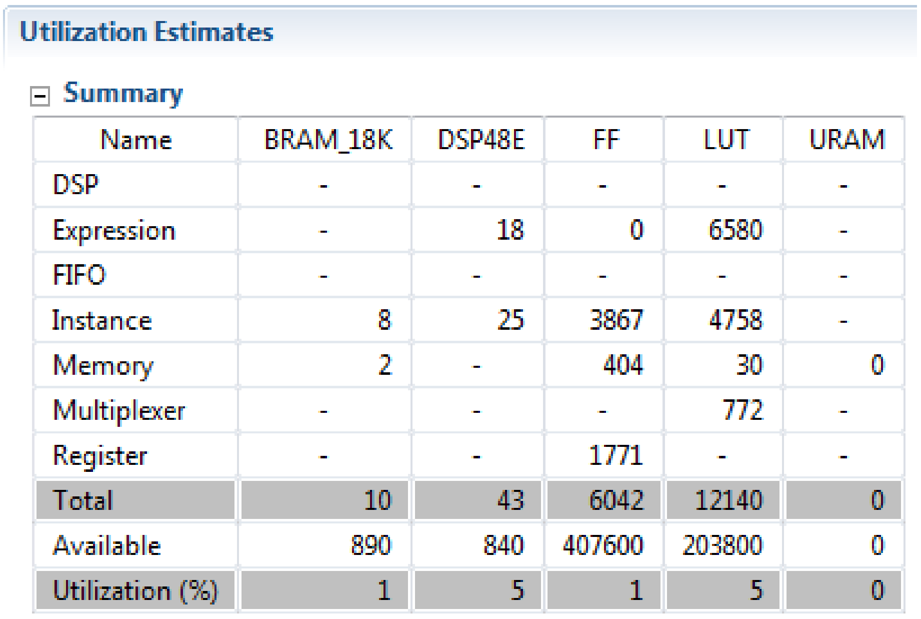
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
287	287	287	287	function

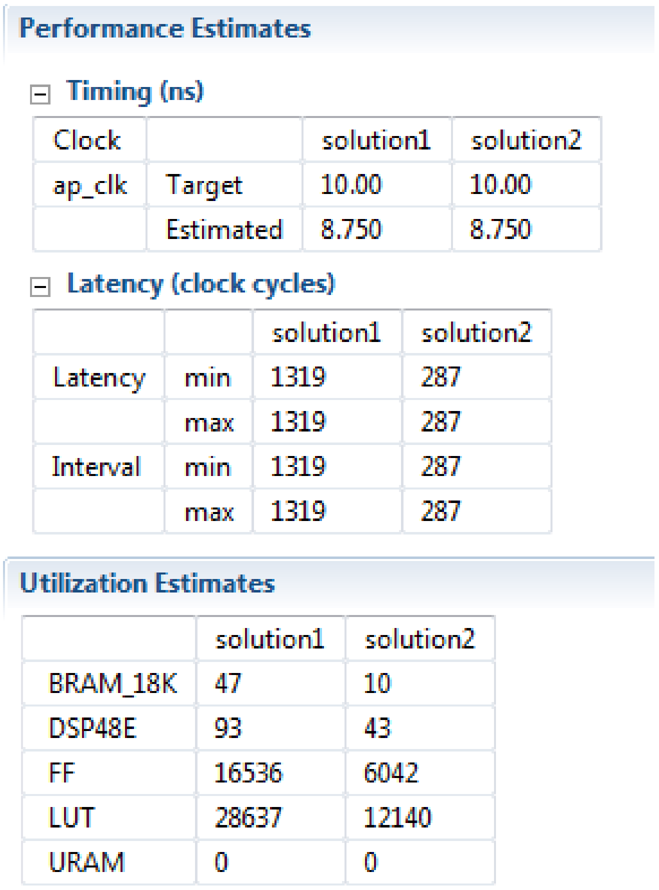


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	18	0	6580	-
FIFO	-	-	-	-	-
Instance	8	25	3867	4758	-
Memory	2	-	404	30	0
Multiplexer	-	-	-	772	-
Register	-	-	1771	-	-
Total	10	43	6042	12140	0
Available	890	840	407600	203800	0
Utilization (%)	1	5	1	5	0

Table (4.6) comparison between solution 1 and 2.



Performance Estimates

⊟ **Timing (ns)**

Clock		solution1	solution2
ap_clk	Target	10.00	10.00
	Estimated	8.750	8.750

⊟ **Latency (clock cycles)**

		solution1	solution2
Latency	min	1319	287
	max	1319	287
Interval	min	1319	287
	max	1319	287

Utilization Estimates

	solution1	solution2
BRAM_18K	47	10
DSP48E	93	43
FF	16536	6042
LUT	28637	12140
URAM	0	0

The HLS can also produce intellectual property (IP) block to be used in Vivado IP integrator. Figure 4.2 represents the IP generated from trellis function process.

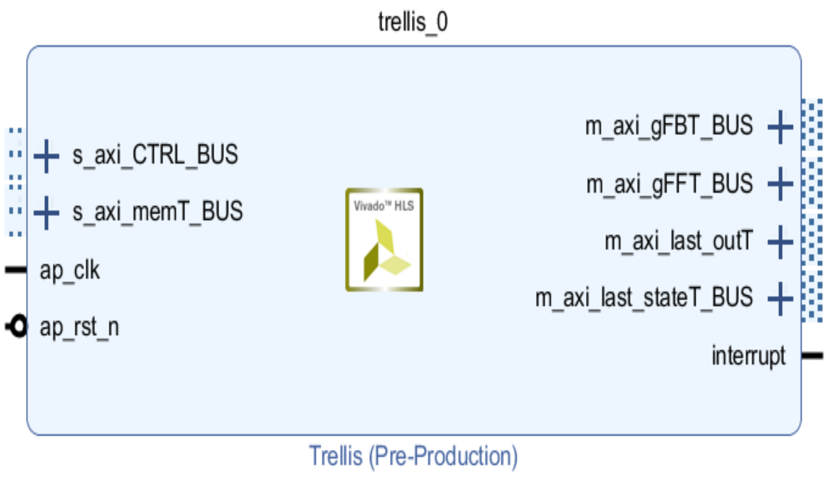


Figure 4.3 IP of the trellis

4.3 Implementation of convolutional hard Decoder Circuit:

Inputs:

last_stateH_BUS : the last state

last_outH_BUS : the last output

lenH_BUS : the length of the massage sent

nstateH_BUS : the number of state

rxH_BUS :the vector of the message sent

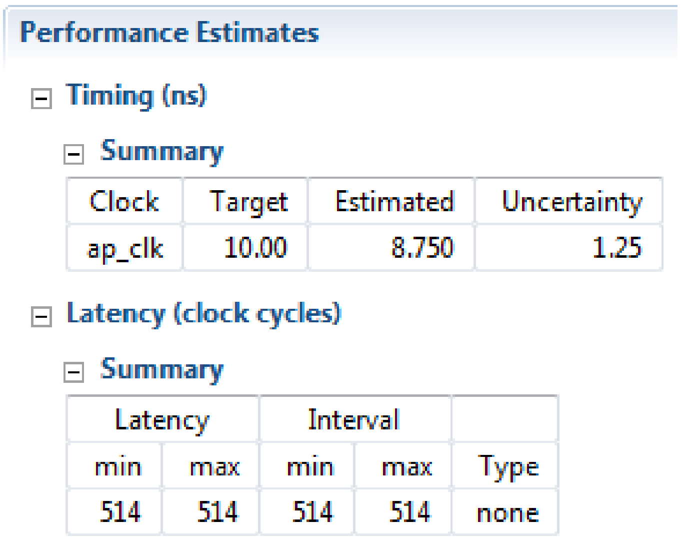
outputs:

DecodedBitsH_BUS :the vector of output bits from decoder

The function was checked inside the HLS program using the test bench program, it was also written in C ++ language to call the function and its implementation after entering the necessary data, and it was verified that it is working correctly after noting the outputs.

The performance and utilization estimates produced by HLS for the convolutional hard decoder circuit are given by Table (4.7).

Table (4.7) Performance and utilization estimates for convolutional hard decoder circuit for solution 1.



Performance Estimates

Timing (ns)

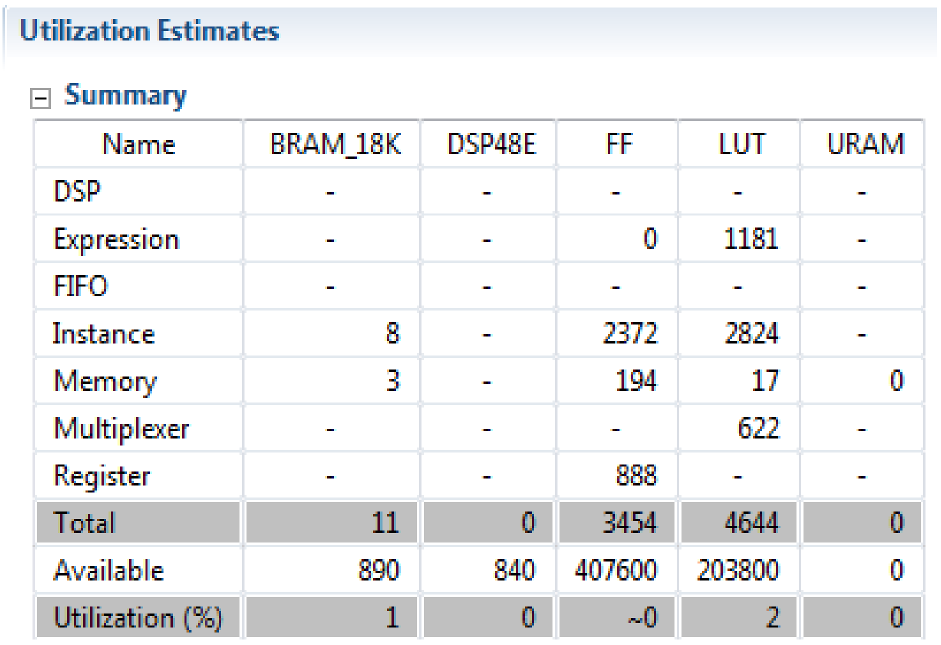
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
514	514	514	514	none

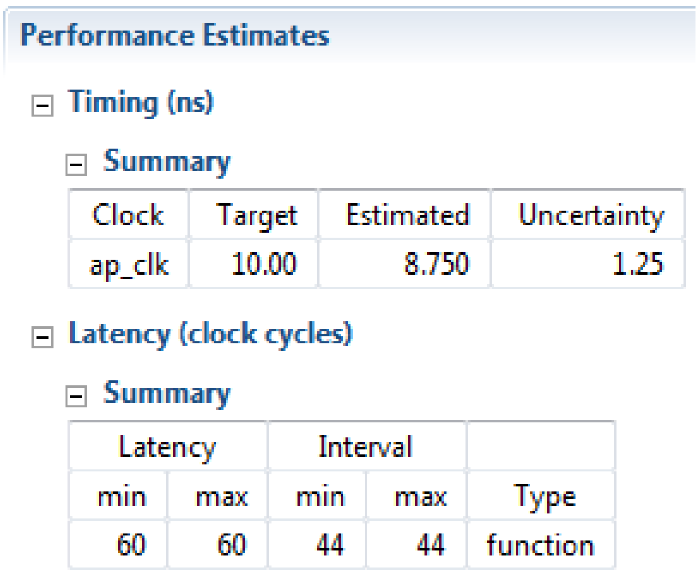

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1181	-
FIFO	-	-	-	-	-
Instance	8	-	2372	2824	-
Memory	3	-	194	17	0
Multiplexer	-	-	-	622	-
Register	-	-	888	-	-
Total	11	0	3454	4644	0
Available	890	840	407600	203800	0
Utilization (%)	1	0	~0	2	0

With the implication of some directives like pipelining, one can achieve a significant improvement in delay and interval time as well as the need for total silicon area. Table (4.8) gives an idea about the delay and utilization after adding pipeline directives.

Table (4.8): delay time and required space with pipelining from circuit for convolutional hard decoder solution 2.


Performance Estimates

Timing (ns)

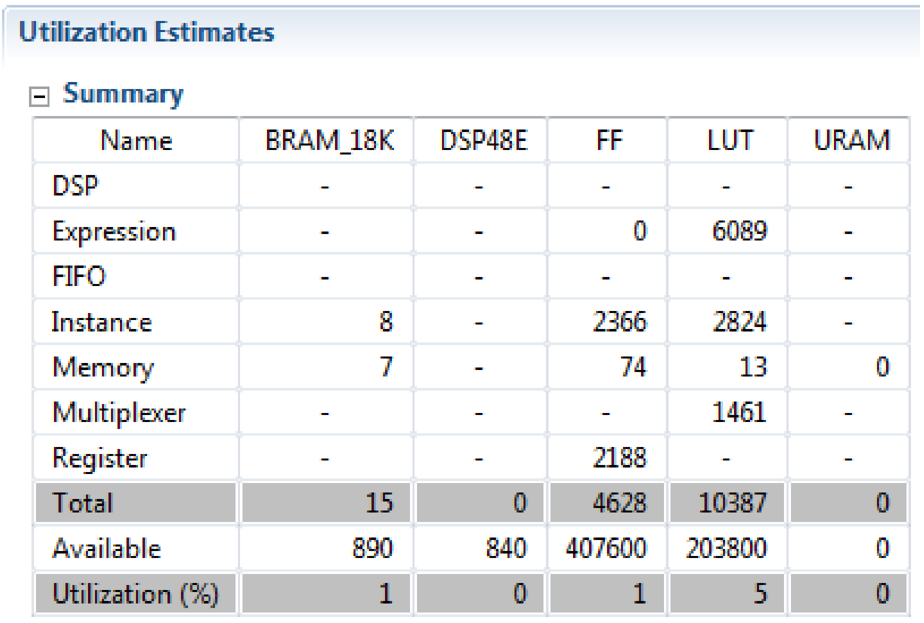
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
60	60	44	44	function

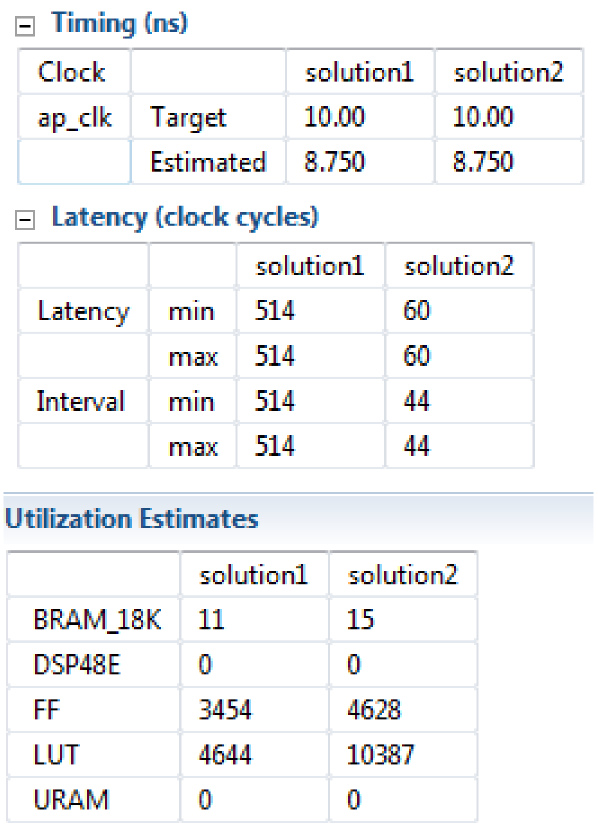


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6089	-
FIFO	-	-	-	-	-
Instance	8	-	2366	2824	-
Memory	7	-	74	13	0
Multiplexer	-	-	-	1461	-
Register	-	-	2188	-	-
Total	15	0	4628	10387	0
Available	890	840	407600	203800	0
Utilization (%)	1	0	1	5	0

Table (4.9) comparison between solution 1 and 2.



Timing (ns)

Clock		solution1	solution2
ap_clk	Target	10.00	10.00
	Estimated	8.750	8.750

Latency (clock cycles)

		solution1	solution2
Latency	min	514	60
	max	514	60
Interval	min	514	44
	max	514	44

Utilization Estimates

	solution1	solution2
BRAM_18K	11	15
DSP48E	0	0
FF	3454	4628
LUT	4644	10387
URAM	0	0

The HLS can also produce intellectual property (IP) block to be used in Vivado IP integrator. Figure 4.3 represents the IP generated from the hard decoder function process.

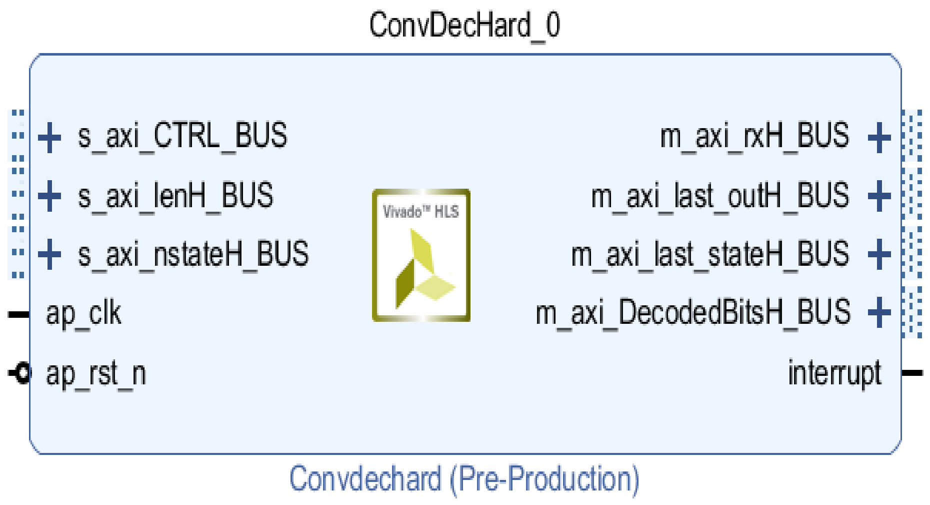


Figure 4.4 IP of convolutional hard decoder

4.4 Implementation of convolutional soft Decoder Circuit:

Inputs:

last_stateS_BUS : the last state

last_outS_BUS : the last output

lenS_BUS : the length of the massage sent

nstateS_BUS : the number of state

rxS_BUS :the vector of the message sent

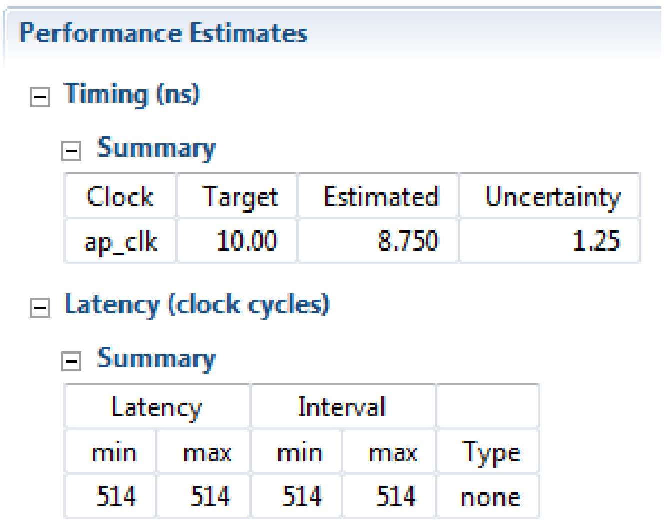
outputs:

DecodedBitsS_BUS :the vector of output bits from decoder

the HLS program using the test bench program, it was also written in C ++ to call the function and its implementation after entering the necessary data, and it was verified that it is working correctly after noting the outputs.

The performance and utilization estimates produced by HLS for the convolutional soft decoder circuit are given by Table (4.10).

Table (4.10) Performance and utilization estimates for convolutional soft decoder circuit for solution 1.



Performance Estimates

Timing (ns)

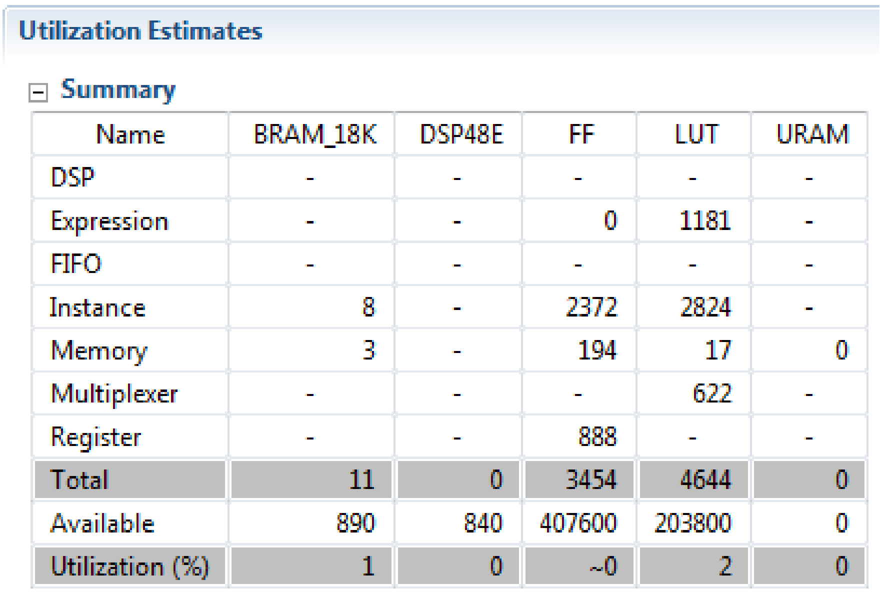
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
514	514	514	514	none



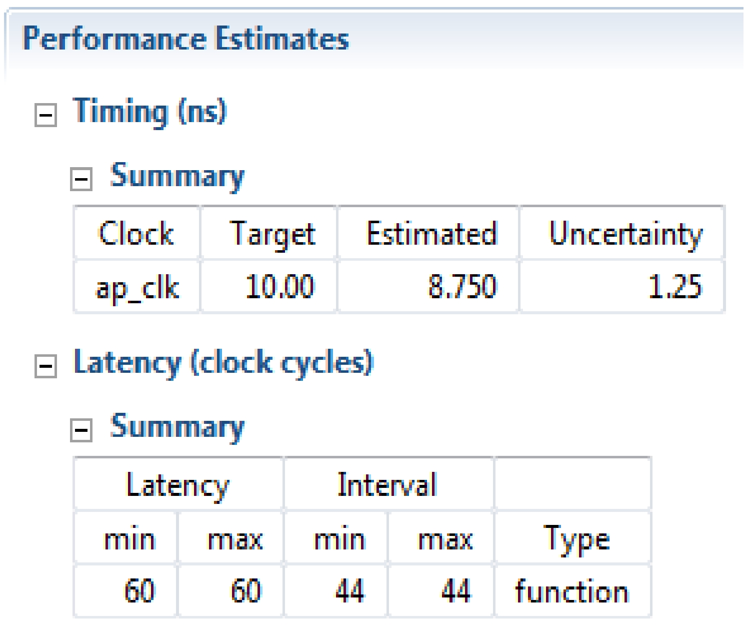
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1181	-
FIFO	-	-	-	-	-
Instance	8	-	2372	2824	-
Memory	3	-	194	17	0
Multiplexer	-	-	-	622	-
Register	-	-	888	-	-
Total	11	0	3454	4644	0
Available	890	840	407600	203800	0
Utilization (%)	1	0	~0	2	0

With the implication of some directives like pipelining, one can achieve a significant improvement in delay and interval time as well as the need for total silicon area. Table (4.11) gives an idea about the delay and utilization after adding pipeline directives.

Table (4.11): delay time and required space with pipelining from circuit for convolutional hard decoder solution 2.



Performance Estimates

Timing (ns)

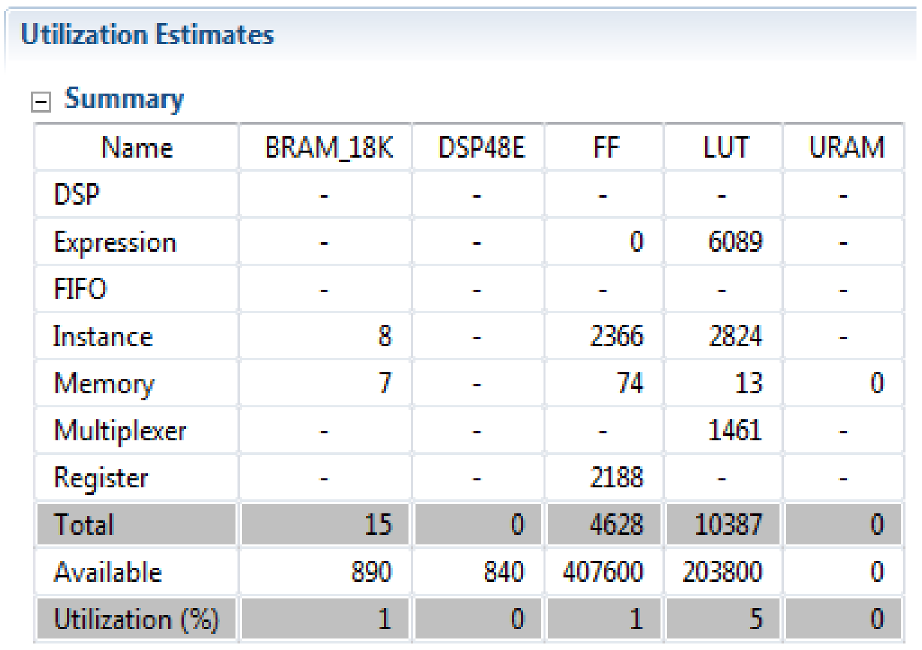
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
60	60	44	44	function

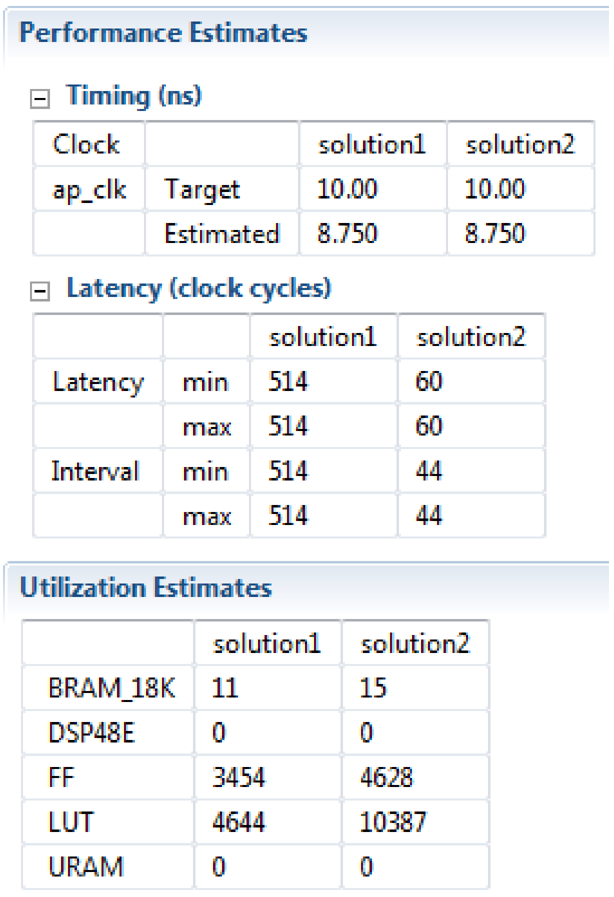


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6089	-
FIFO	-	-	-	-	-
Instance	8	-	2366	2824	-
Memory	7	-	74	13	0
Multiplexer	-	-	-	1461	-
Register	-	-	2188	-	-
Total	15	0	4628	10387	0
Available	890	840	407600	203800	0
Utilization (%)	1	0	1	5	0

Table (4.12) comparison between solution 1 and 2.



Performance Estimates

Timing (ns)

Clock		solution1	solution2
ap_clk	Target	10.00	10.00
	Estimated	8.750	8.750

Latency (clock cycles)

		solution1	solution2
Latency	min	514	60
	max	514	60
Interval	min	514	44
	max	514	44

Utilization Estimates

	solution1	solution2
BRAM_18K	11	15
DSP48E	0	0
FF	3454	4628
LUT	4644	10387
URAM	0	0

The HLS can also produce intellectual property (IP) block to be used in Vivado IP integrator. Figure 4.4 represents the IP generated from the soft decoder function process.

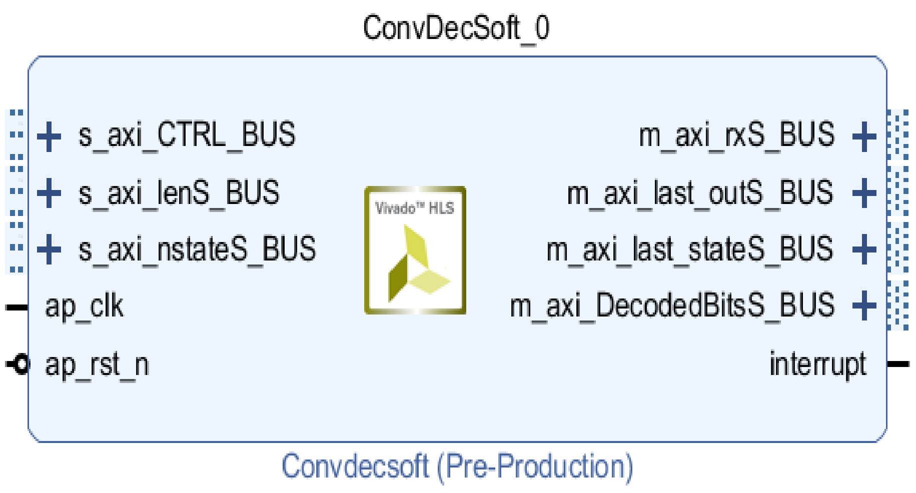


Figure 4.5 IP of convolutional soft decoder

Figure (4.6) represent the flowchart of the convolutional decoder.

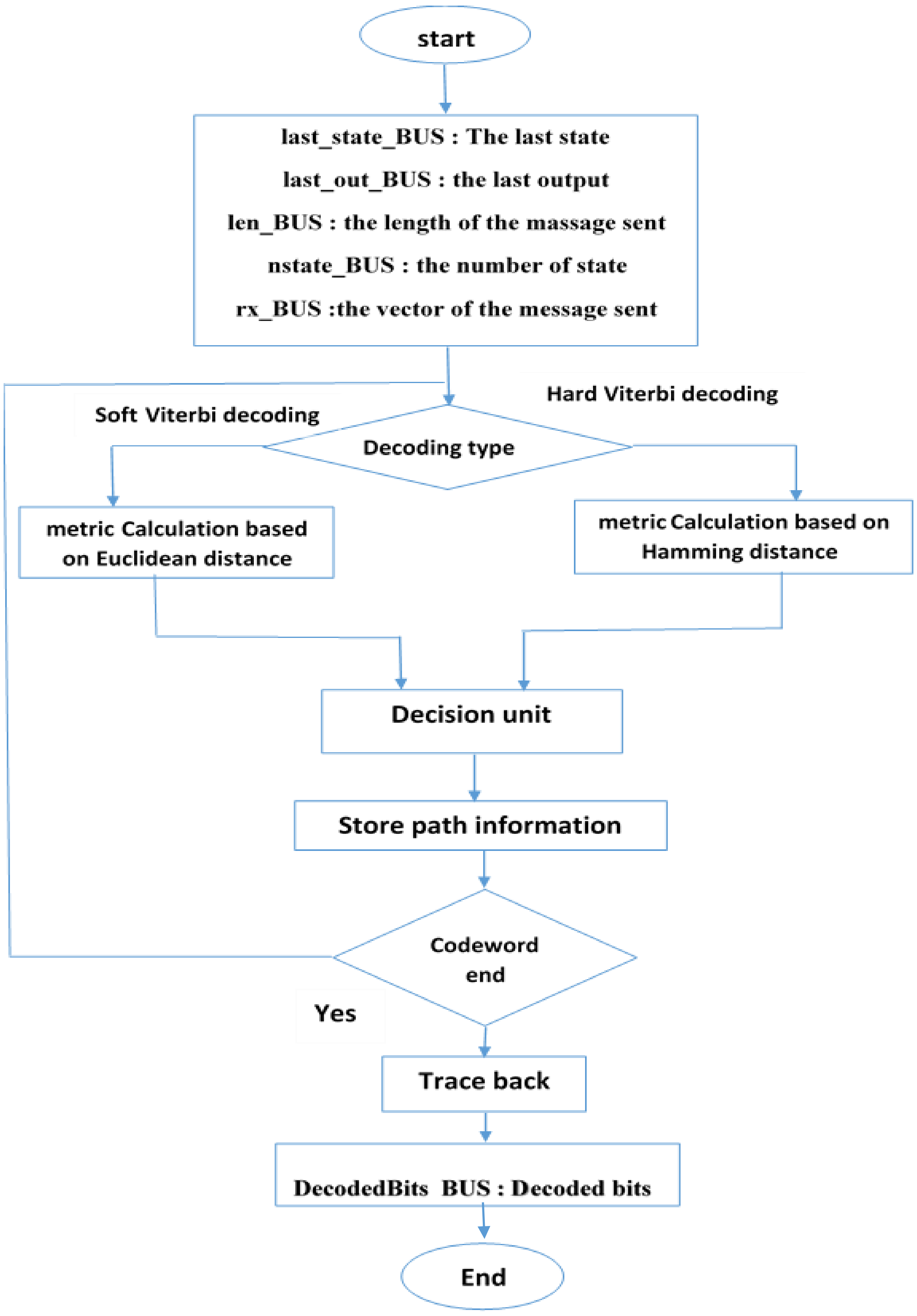

start
last_state_BUS : The last state
last_out_BUS : the last output
len_BUS : the length of the massage sent
nstate_BUS : the number of state
rx_BUS :the vector of the message sent
Hard Viterbi decoding
Soft Viterbi decoding
Decoding type
metric Calculation based on Euclidean distance
metric Calculation based on Hamming distance
Decision unit
Store path information
Codeword end
Yes
Trace back
DecodedBits BUS : Decoded bits
End

Figure 4.6 Flow Chart of convolutional decode

4.5 Bit-error rate (Pe) performance of Convolutional code

Figure (4.7) shows the BER and FER performance against E_b/N_o for massage length k=32, without trellis termination and generator of (7,5). It is obvious that the coded systems with soft decoding outperform the uncoded system by approximately 2 dB at BER of 10^{-4}. On the other hand, the hard Viterbi decoder present a degradation in performance relative to the uncoded system over all E_b/N_o range. This is mainly due to the loss of channel information by threshold detection and not using trellis termination which leads to error propagation in the trellis diagram. The soft decoder introduces an improvement by about 3.3 dB over hard decoder at BER of 10^{-4}.

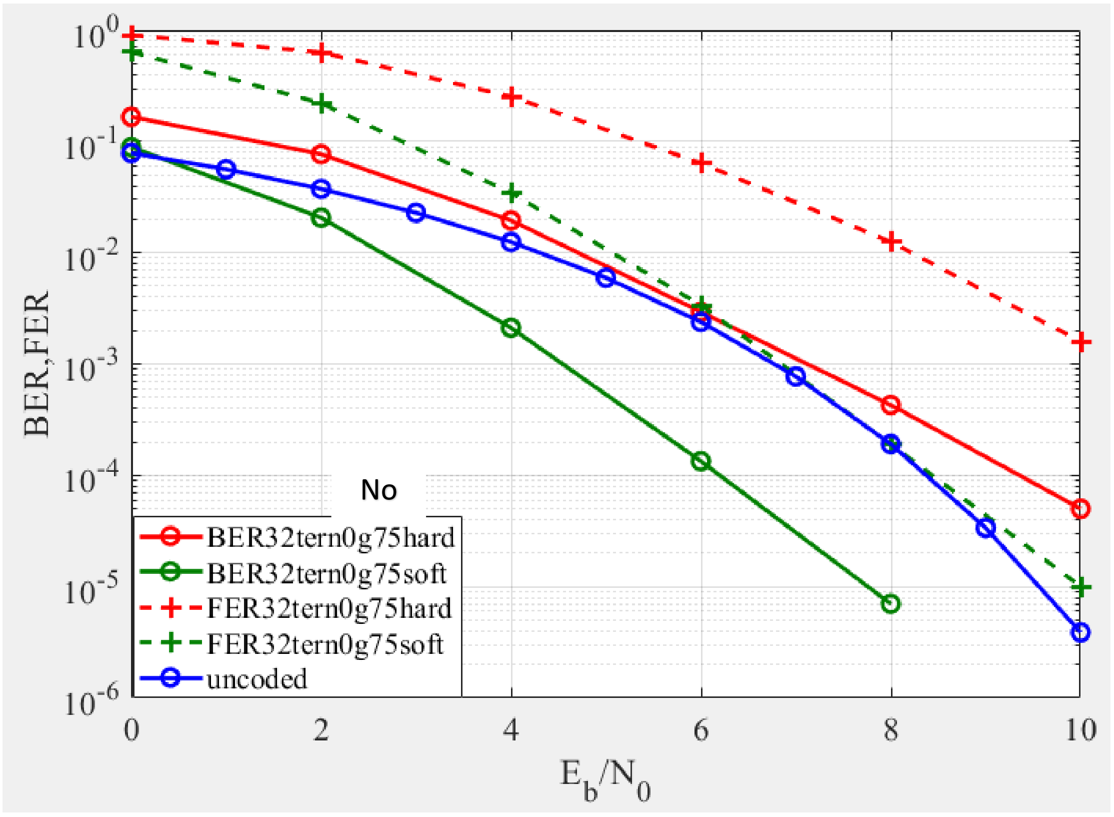


Figure (4.7) :BER and FER performance against E_b/N_o for message length k=32, without trellis termination

When the length k = 32, it is shown that without trellis termination the hard is worse than uncoded, because without trellis we choose the smallest path and it may not really be the last state you reach.

Also, for short lengths, the calculation of the maximum likelihood path has errors due to the propagation error.

Figure (4.8) shows the BER and FER performance against E_b/N_o for massage length k=32, with trellis termination and generator of (7,5). It is obvious that the coded systems outperform the uncoded system. The hard Viterbi decoder present a coding gain by about 1.5 dB over uncoded system at BER of 10^{-4}. Meanwhile, the soft decoder has coding gain of approximately 3.5 dB. The soft decoder introduces an improvement by about 2 dB over hard decoder at BER of 10^{-4}. The variation in performance between soft and hard decoders is mainly due to the usage of the channel output directly in soft decoding and the threshold decision by the hard decoder which result in loss of a lot information of the received signal. Also trellis termination aids in the decision of the maximum likelihood path.

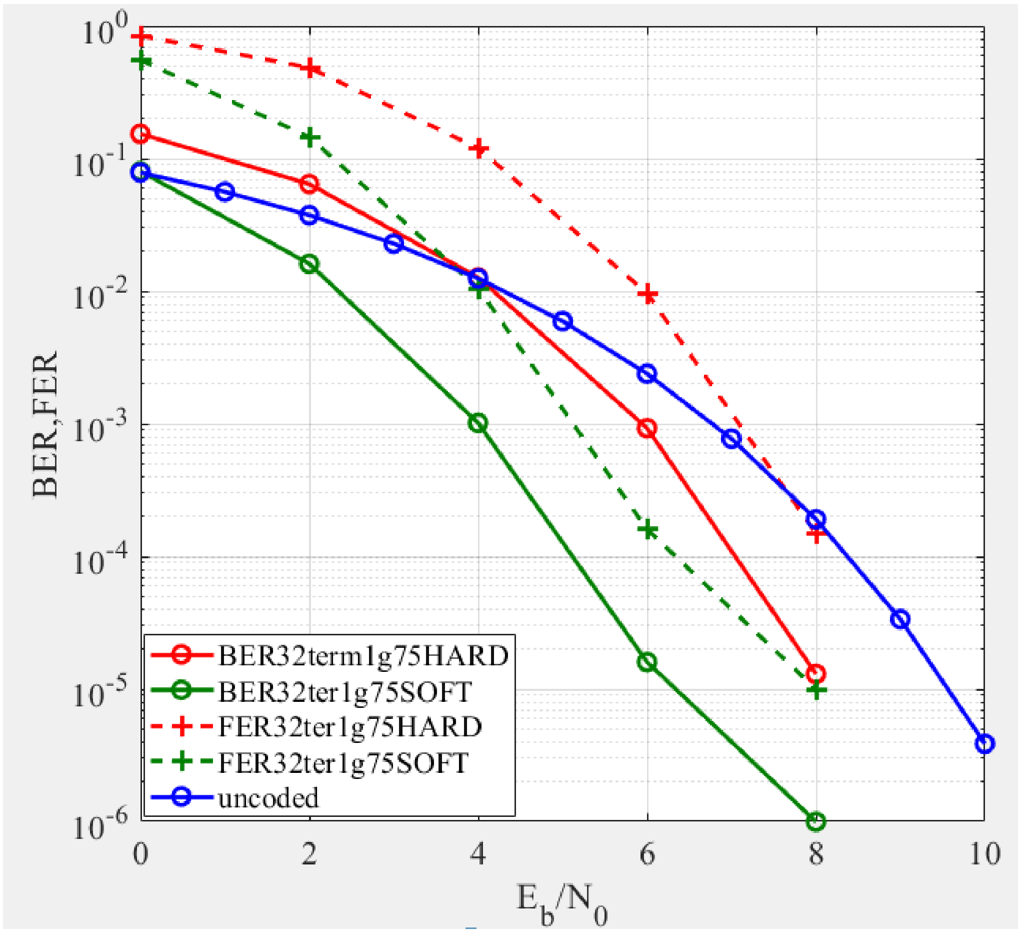


Figure (4.8) :BER and FER performance against E_b/N_o for message length k=32, with trellis termination

different tests are carried out for different lengths and the following are samples to them. Again for massage length of k=64, without trellis termination and generator of (7,5), the hard Viterbi decoder present deterioration relative uncoded system as shown in figure (4.9). The soft decoder has coding gain of approximately 2.2 dB. The soft decoder introduces an improvement by about 2.3 dB over hard decoder at BER of 10^{-4}.

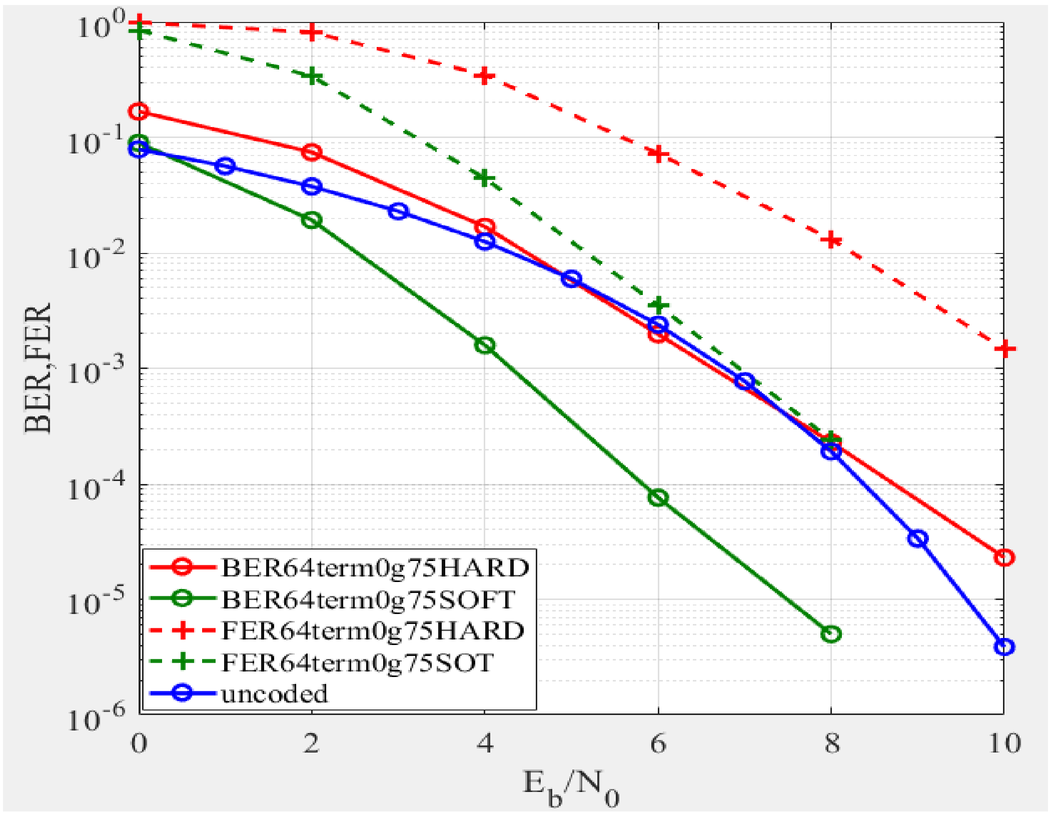


Figure (4.9) :BER and FER performance against E_b/N_o for message length k=64, without trellis termination

For massage length k=64, with trellis termination and generator of (7,5). The hard Viterbi decoder present a coding gain by about 1.2 dB over uncoded system at BER of 10^{-4} as depicted in figure (4.10). The soft decoder has coding gain of approximately 3.2 dB. The soft decoder introduces an improvement by about 2 dB over hard decoder at BER of 10^{-4}.

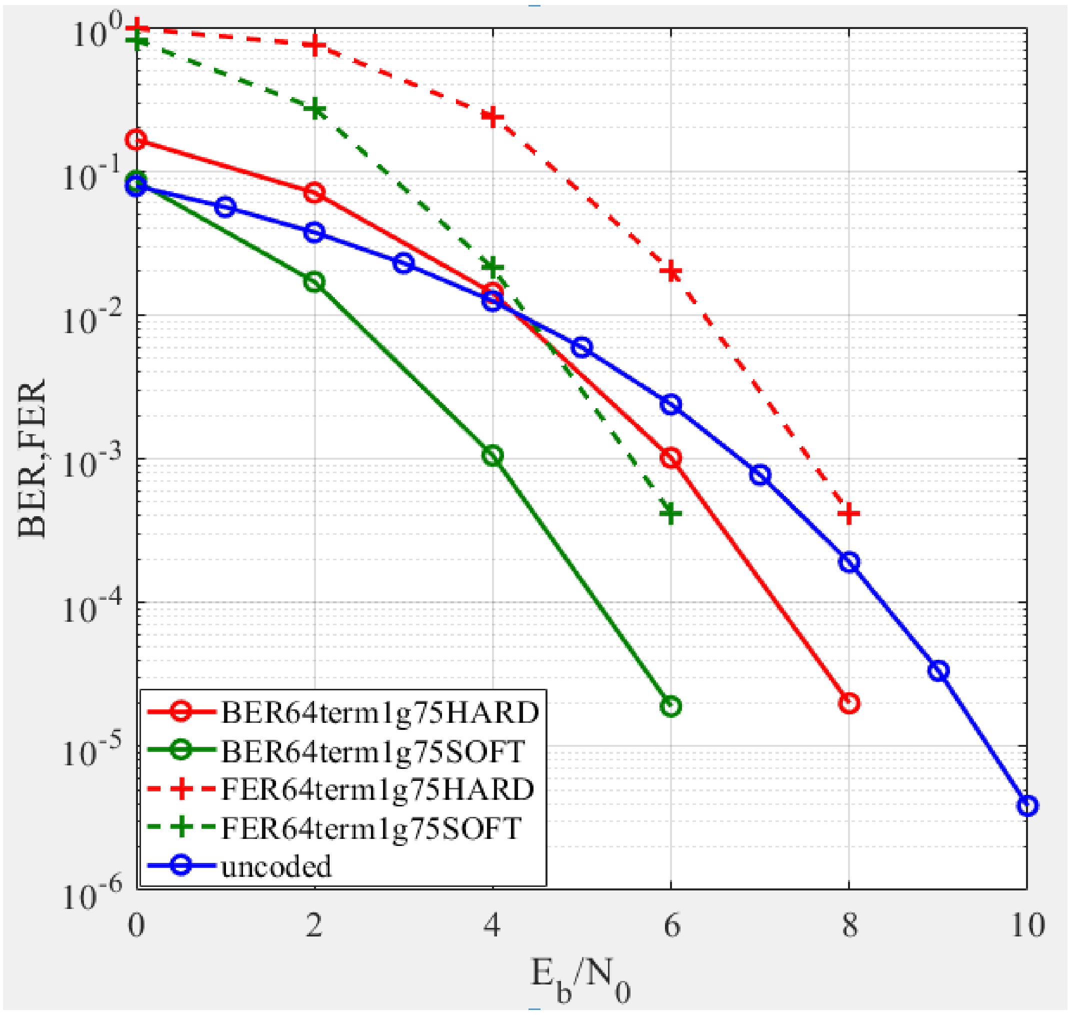


Figure (4.10) :BER and FER performance against E_b/N_o for message length k=64, with trellis termination

Figure (4.11) shows the performance of coded system with massage length of k=128, without trellis termination and generator of (7,5). The hard Viterbi decoder present a coding gain by about 1 dB over uncoded system at BER of 10^{-4}. The soft decoder introduces a coding gain of approximately 2.5 dB. The soft decoder introduces an improvement by about 1,5 dB over hard decoder at BER of 10^{-4}.

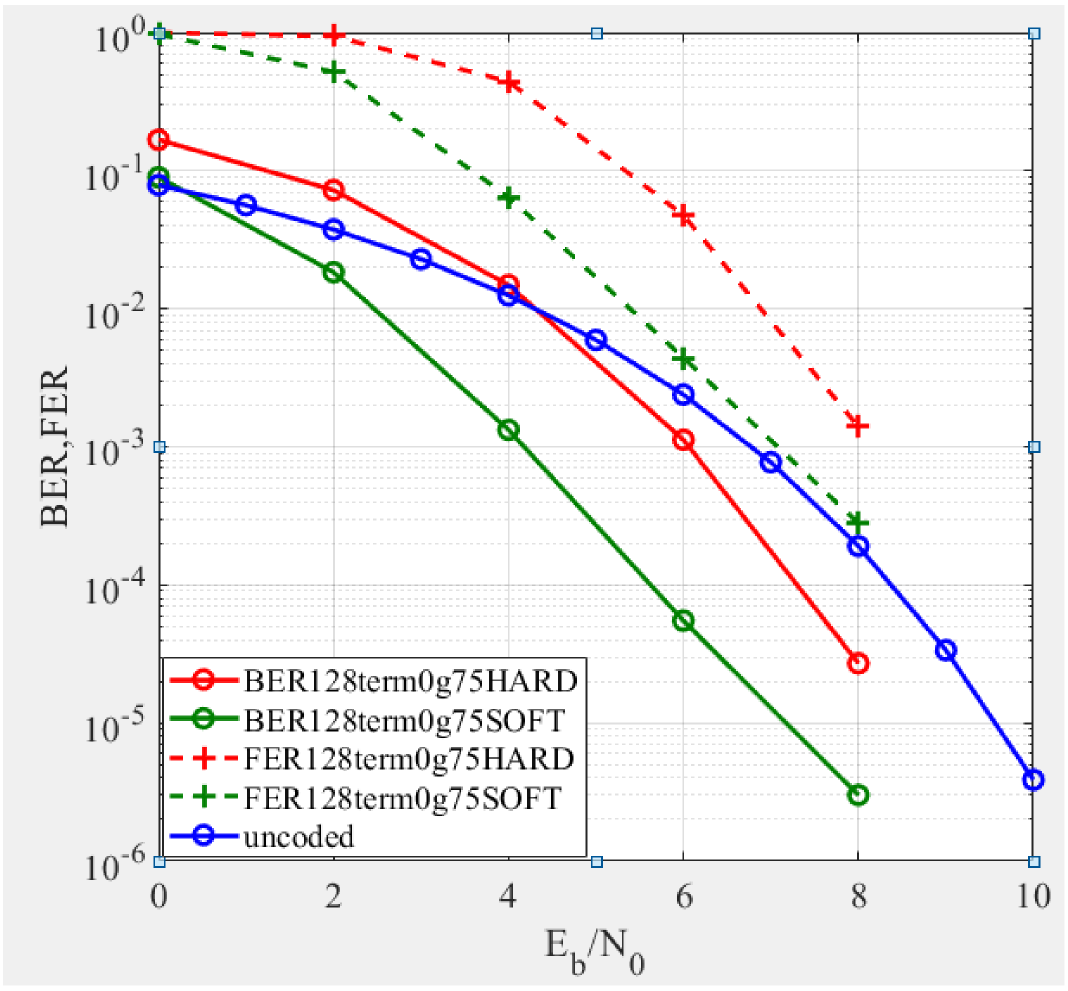


Figure (4.11) :BER and FER performance against E_b/N_o for message length k=128, without trellis termination

With massage length of k=128, and trellis termination and generator of (7,5). The hard Viterbi decoder present a coding gain by about 1.2 dB over uncoded system at BER of 10^{-4}. The soft decoder has coding gain of approximately 2.6 dB. The soft decoder introduces an improvement by about 1.4 dB over hard decoder at BER of 10^{-4} as presented in figure (4.12).

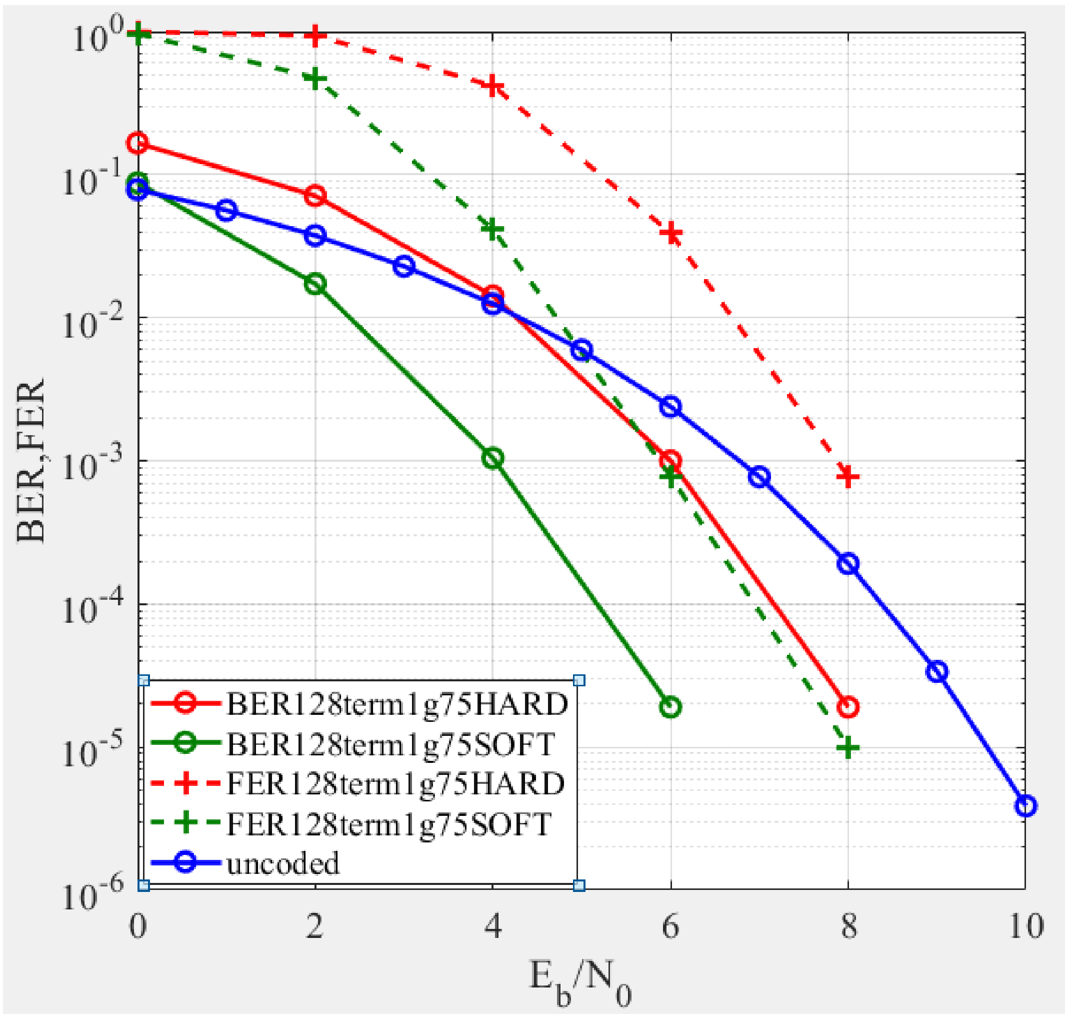


Figure (4.12) :BER and FER performance against E_b/N_o for message length k=128, with trellis termination

for the same massage length of k=256, without trellis termination and generator of (7,5), the hard Viterbi decoder present a coding gain by about 0.5 dB over uncoded system at BER of 10^{-4} as shown in figure (4.13). The soft decoder has coding gain of approximately 2.8 dB. The soft decoder introduces an improvement by about 2.3 dB over hard decoder at BER of 10^{-4}.

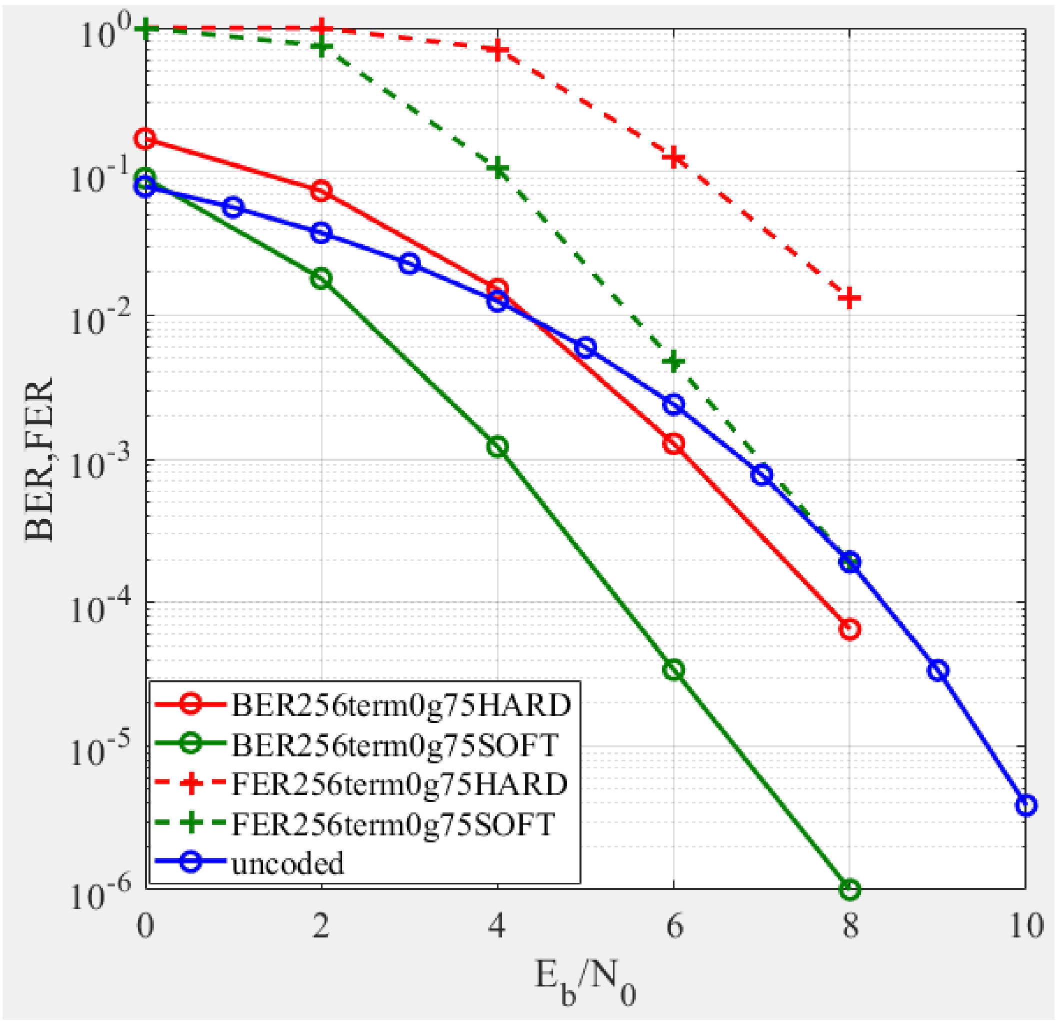


Figure (4.13) :BER and FER performance against E_b/N_o for message length k=256, without trellis termination

Figure (4.14) show the performance of coded systems that utilize trellis termination with massage length k=256 and generator of (7,5). The hard Viterbi decoder present a coding gain by about 1.2 dB over uncoded system at BER of 10^{-4}. The soft decoder has coding gain of approximately 3.2 dB. The soft decoder introduces an improvement by about 2 dB over hard decoder at BER of 10^{-4}.

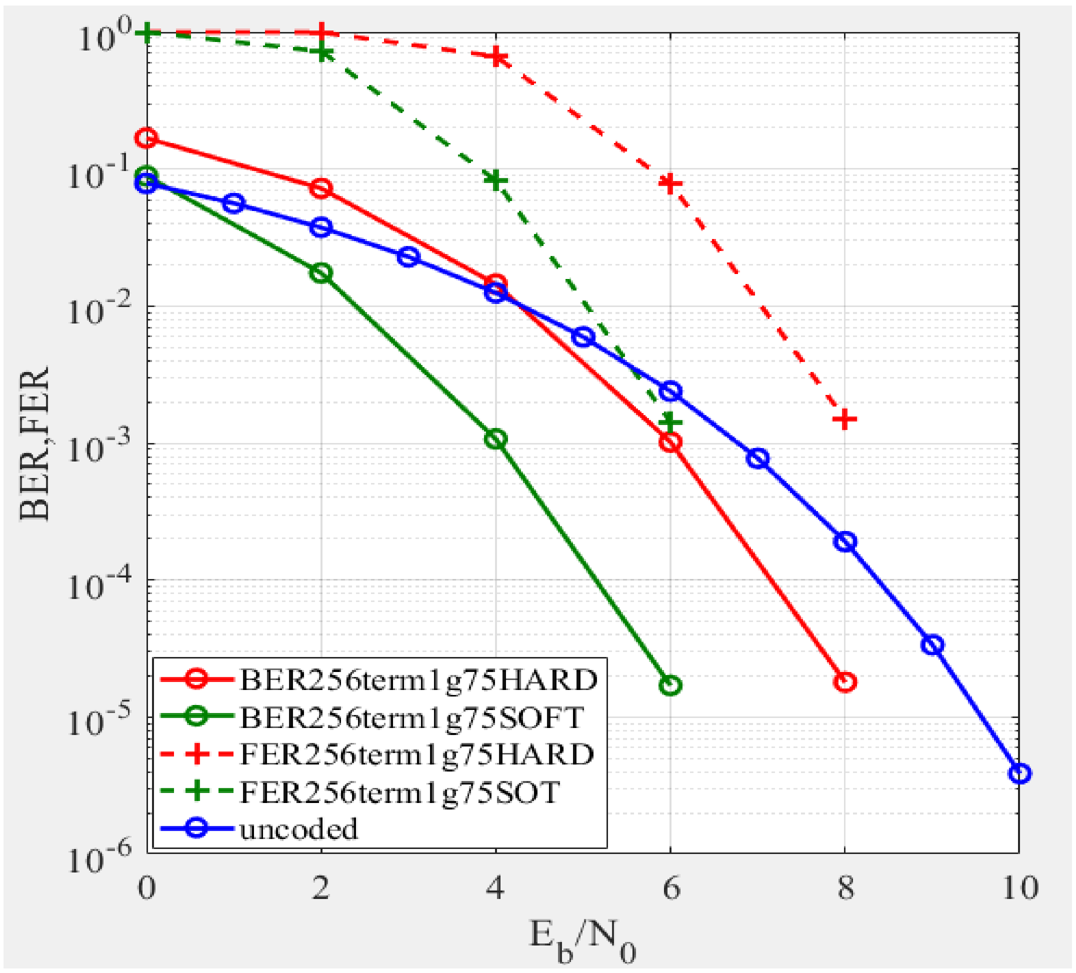


Figure (4.14) :BER and FER performance against E_b/N_o for message length k=256, with trellis termination

When comparing the different lengths of sing hard decoding without terminaion, over uncoded system at BER of 10^{-4}. We notice that the system with larger length k =256 present better performance comared with systems have smaller length. This is clearly shown in figure (4.15) presents that the systems with large lengths outperform the systems with short length. This mainly due to the better calculation of the maximum likelihood path.

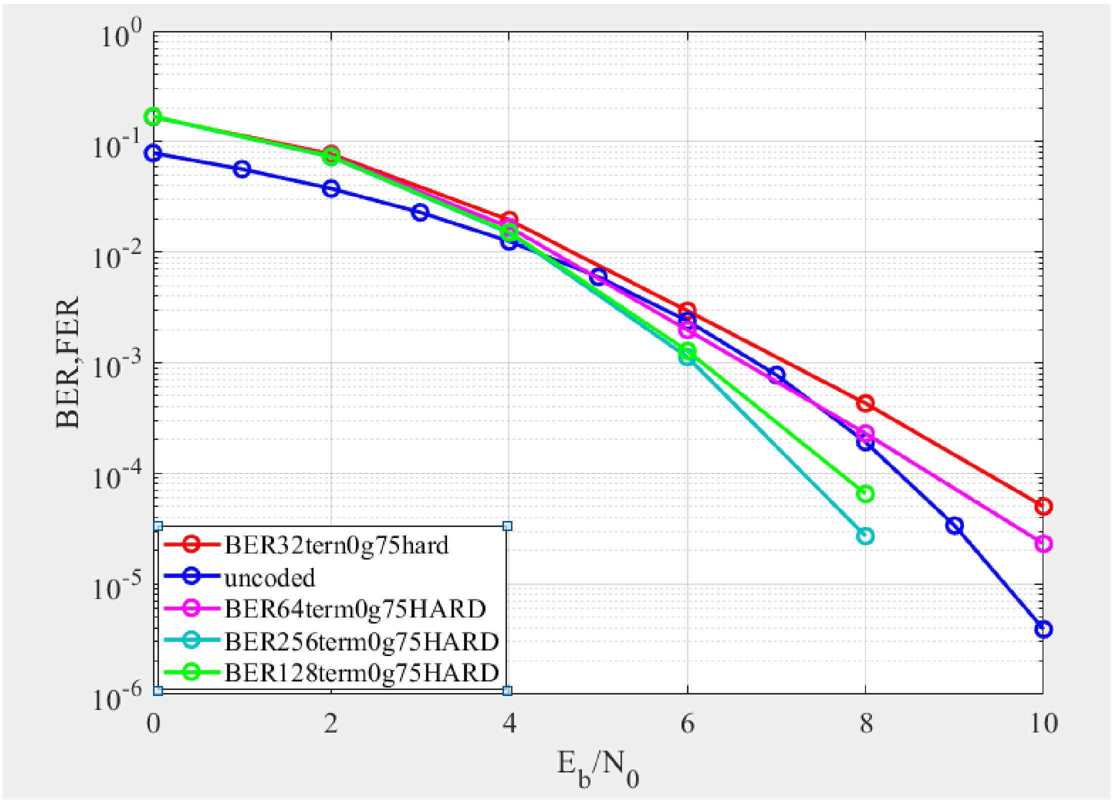


Figure (4.15) compare of different lengths

Chapter Five

Conclusions and Suggestions for Future Works

5.1 Conclusions

- IP configuration was achieved through the use of the Vivado High-Level Synthesis (HLS) platform. This platform allowed the creation of convolutional coded networks.
- The design of the different sections (ENCODER, TRELLIS, AND TWO TYPES OF DECODER) was done by using a high-level language (C++), which was found to be more efficient and faster than VHDL in terms of performance and time to market.
- We also illustrate the effectiveness of HLS directives such as pipelines in reducing latency and interval time by a substantial amount.
- In the case of soft when calculating the metric, we calculate the Euclidean distance, which takes the correct difference between the sent code and the o/p on the trellis after converting the (0,1) to (-1,+1).

As for the hard, we calculate the Hamming distance after we compare the transmitted bits and the Threshold detection, and thus errors will occur.

- The systems with large lengths outperform the systems with short length. This mainly due to the better calculation of the maximum likelihood path.
- When the length k = 32, it is shown that without trellis termination the hard is worse than uncoded, because without

trellis we choose the smallest path and it may not really be the last state you reach. Also, for short lengths, the calculation of the maximum likelihood path has errors due to the propagation error.

5.2 Future Works

The following points are suggestions for future work

1. The design procedure introduces in this work can be applied to different codes such as, BCH [27], Reed-Solomon [28], turbo codes [29] low-density parity check code (LDPC) [30] and polar code [31].
2. A modified version of FPGA, for instance, Virtix7, Ultra scale Zinc and more, can replace the Artix7 FPGA that used in this work.
3. The design and implementation of the control boards, interface circuits, and power unit that comprises the laboratory board is an interesting target.

References

[1] Gupta, Suneha. Design and Implementation of an Optimized Viterbi Decoder. Diss. 2012.

[2] John G. Proakis, “Digital Communication,” McGraw Hill, Singapore. Pp 502-507,471- 475 ,2001.

[3] Taher Qassim, Yahya, and Dhafir A. Alneema. "FPGA Based Implementation of Convolutional Encoder-Viterbi Decoder Using Multiple Booting Technique." Al-Rafidain Engineering Journal (AREJ) 18.6 (2010): 70-80.

[4] Pandita, Bupesh, and Subir K. Roy. "Design and implementation of a Viterbi decoder using FPGAs." Proceedings Twelfth International Conference on VLSI Design.(Cat. No. PR00013). IEEE, 1999.

[5] Zbaid, Riham Ali, and Kasim K. Abdalla. "Design and Implementation of Convolutional Encoder and Viterbi Decoder Using FPGA." Journal of University of Babylon for Pure and Applied Sciences 26.3 (2018): 22-29.

[6] Tulasi, J., T. Venkata Lakshmi, and M. Kamaraju. "FPGA Implementation of Convolutional Encoder And Hard Decision Viterbi Decoder." International Journal of Computer & Communication Technology (IJCCT) 3.4 (2012).

[7] Hema, S., V. Suresh Babu, and P. Ramesh. "FPGA implementation of Viterbi decoder." Proceedings of the 6th WSEAS International Conference on Electronics, Hardware, Wireless and Optical Communications. 2007.

[8] Kivioja, M., Jouni Isoaho, and L. Vänskä. "Design and implementation of Viterbi decoder with FPGAs." Journal of VLSI signal processing systems for signal, image and video technology 21.1 (1999): 5-14.

[9] Jacobs, I. "Practical applications of coding." IEEE Transactions on Information Theory 20.3 (1974): 305-310.

[10] V. et al. "Implementation of Convolutional encoder and Viterbi decoder using Verilog HDL." 2011 3rd International Conference on Electronics Computer Technology. Vol. 1. IEEE, 2011.

[11] Jabeen, Mahe, and Salma Khan. "Design of convolution encoder and reconfigurable Viterbi decoder." International Journal of Engineering and Science ISSN (2012): 2278-4721

[12] Alaus, Laurent, Dominique Noguet, and Jacques Palicot. "Extended reconfigurable linear feedback shift register operators for software defined radio." 2008 IEEE 10th International Symposium on Spread Spectrum Techniques and Applications. IEEE, 2008.

[13] Sweeney, Peter. Error control coding: from theory to practice. John Wiley & Sons, Inc., 2002.

[14] 3GPP Technical Specifications 36.212, "Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 15) ," 2019

[15] Moon, Todd K. Error correction coding: mathematical methods and algorithms. John Wiley & Sons, 2020.

[16] 3GPP Technical Specifications 36.212, "Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 15) ," 2019.

[17] Sklar, Bernard. Digital communications. Vol. 2. Upper Saddle River, NJ, USA: Prentice hall, 2001.

[18] Sweeney, Peter. Error control coding: from theory to practice. John Wiley & Sons, Inc., 2002.

[19] Clive Maxfield. 2004. The Design Warrior's Guide to FPGAs: Devices, Tools and Flows (1st. ed.). Newnes, USA.

[20] Jack, Keith. Digital video and DSP: Instant access. Newnes , USA, (1st. ed.) Copyright © 2008, Elsevier Ltd .

[21] Dr. Ahmed Al-Rekabi. " Configuration Methods and Practical Applications of FPGA". Nov. 25, 2021

[22] Maxfield, Clive. The design warrior's guide to FPGAs: devices, tools and flows. Elsevier, 2004.

[23] Vivado Design Suite User Guide. High-Level Synthesis. UG902 (v2019.1) July 12, 2019

[24] Chen, Deming, Jason Cong, and Peichen Pan. FPGA design automation: A survey. Now Publishers Inc, 2006.

[25] Qasaimeh, Murad, et al. "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels." 2019 IEEE international conference on embedded software and systems (ICESS). IEEE, 2019.

[26] Mano, M. Morris. Computer system architecture. Prentice-Hall, Inc., 1993.

[27] A. Hocquenghem, “Codes correcteurs d’erreurs,” Chiffers, vol. 2, pp. 147–156, 1959

[28] I. Reed and G. Solomon, “Polynomial Codes over Certain Finite Fields,” J. SOC. Indust. Appl. Math, vol. 8 , pp. 300-304, 1960.

[29] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in Proceedings of ICC’93-IEEE International Conference on Communications, 1993, pp. 1064–1070, doi: 10.1109/ICC.1993.397441.

[30] R. G. Gallager, “Low-Density Parity Check Codes,” IRE Transactions on Information Theory, Vol. 8, No. 1, 1962, pp. 21-28.http://dx.doi.org/10.1109/TIT.1962.1057683.

[31] E. Arikan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," in IEEE Transactions on Information Theory, vol. 55, no. 7, pp. 3051-3073, July 2009, doi: 10.1109/TIT.2009.2021379