P-3455

IMPLEMENTATION OF LDPC DECODER IN FPGA

By

MAHESH K R

Reg. No. 0920106008

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University of Technology, Coimbatore)

COIMBATORE - 641049

A PROJECT REPORT

Submitted to the

FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

for the award of the degree

of

MASTER OF ENGINEERING

IN

APPLIED ELECTRONICS

APRIL 2011

BONAFIDE CERTIFICATE

Certified that this project report entitled "**IMPLEMENTATION OF LDPC DECODER IN FPGA**" is the bonafide work of Mr. Mahesh K R [Reg. No. 0920106008] who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Project Guide

Ms.G.Amirtha Gowri

Head of the Department

Dr. (Ms.) Rajeswari Mariappan

The candidate with university Register no. 0920106008 is examined by us in the project viva-voce examination held on ...21-4-2011...

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

A project of this nature needs co-operation and support from many for successful completion. In this regard, we are fortunate to express our thanks to **Padmabhusan Arutselvar Dr.N.Mahalingam B.Sc.,F.I.E.,** Chairman and Co-Chairman **Dr.B.K.Krishnaraj Vanavarayar B.Com, B.L.,** for providing necessary facilities for the course.

First I would like to express my praise and gratitude to the Lord, who has showered his grace and blessing enabling me to complete this project in an excellent manner. I express my sincere thanks to our beloved Director **Dr.J.Shanmugam,** Kumaraguru College of Technology, for the kind support and necessary facilities to carry out the project work.

I express my sincere thanks to our beloved Principal **Dr.S.Ramachandran,** Kumaraguru College of Technology, who encouraged and supported me the project work.

I would like to express my sincere thanks and deep sense of gratitude to our HOD, **Dr.Rajeswari Mariappan**, Department of Electronics and Communication Engineering, for the valuable suggestions and encouragement which paved way for the successful completion of the project work.

In particular, I wish to thank and everlasting gratitude to the project coordinator **Ms.R.Latha (Ph.D).,** Associate Professor, Department of Electronics and Communication Engineering for her expert counseling and guidance to make this project to a great deal of success.

I am greatly privileged to express deep sense of gratitude to my guide **Ms.G.Amirtha Gowri (Ph.D).,** Associate Professor , Department of Electronics and Communication Engineering, for the valuable suggestions and support throughout the course of this project work. I wish to convey my deep sense of gratitude to all the teaching and non-teaching staff of ECE Department for their help and cooperation.

Finally, I thank my parents and my family members for giving me the moral support and abundant blessings in all of my activities and my dear friends who helped me to endure my difficult times with their unfailing support and warm wishes.

Low Density Parity Check (LDPC) codes are promising error correcting codes. The LDPC H matrix is a sparse matrix. So the memory requirement for this code is very less. This will make the decoding of the LDPC codes very efficient and simple compared to the other error correcting methods. The decoding can be easily done with the help of the tanner graph and message passing algorithm. In this project, the two different decoding methods of LDPC codes are studied. The two methods are hard decision decoding and the soft decision decoding. These two decoding method are analyzed and simulation results are taken from model sim soft ware. For hard decision decoding 6 bit code word is given to the decoder and the performance is analyzed. For hard decision decoding 32 bit code word performance is also analyzed. For soft decision 6 bit code performance is analyzed. LDPC codes give good error correction performance approaching Shannon`s limit. Applications of this error correcting codes are in Digital Video Broadcasting (DVB-2), Gigabit Ethernet, and Wireless broadband communication etc.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

BCH	-------	Bose_ Chaudhuri_Hocquenghen
DVB	-------	Digital Video Broadcasting
ECC	-------	Error Correcting Codes
FPGA	-------	Field Programmable Gate Array
LDPC	-------	Low Density Parity Check Codes
MPA	-------	Message Passing Algorithm
RS	-------	Reed Solomon

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Communication systems transmit data from source to destination through a channel or medium such as air, wire line and optical fiber. Reliability of the received data depends on the channel and external noises that could interface or distort the signal representing the data. Noise introduces errors in the received data. Error detection and correction is achieved by adding redundant symbols to the original data. Forward Error Correction Code (FEC) is used for error correction easily without data need to be retransmitted. Retransmission will result in delay, cost and wastes system throughput. Several error correction codes have been developed to improve the reliability of the data transfer. Forward Error Correction Codes (FEC) includes Viterbi, convolution codes, Bose Chaudhuri Hocquenghen (BCH) codes, Reed Solomon (RS) codes, turbo codes and low density parity check codes (LDPC).

Low Density Parity Check (LDPC) codes are a class of linear block codes, shows good error correcting performance approaching Shannon's limit. Good error correcting performance enables efficient and reliable communication. They were first introduced by Gallager in his Ph.D. thesis in 1960. But due to the computational complexity in implementing encoder and decoder for such codes and the introduction of Reed-Solomon codes, they were mostly ignored until about ten years ago. They remained largely neglected for over 35 years. In the mean time the field of forward error correction was dominated by highly structured algebraic block and convolutional codes. Despite the enormous practical success of these codes, their performance fell well short of the theoretically achievable limits set down by Shannon in his seminal 1948 paper. The relative quiescence of the coding field was utterly transformed by the introduction of turbo codes, proposed by Berrou, Glavieux and Thitimajshima, wherein all the key ingredients of successful error correction codes were replaced: turbo codes involve very little algebra, employ iterative, distributed algorithms, focus on average (rather than worst-case) performance, and rely on soft (or probabilistic) information extracted from the channel.

New generalizations of Gallager's LDPC codes by a number of researchers including Luby, Mitzenmacher, Shokrollahi, Spielman, Richardson and Urbanke, produced new irregular LDPC codes which offer certain practical advantages and an arguably cleaner setup for theoretical results. Today, design techniques for LDPC codes exist which enable the construction of codes which approach the Shannon's capacity to within hundredths of a decibel. The main research interests are low complexity encoding and efficient decoding schemes.

The future wireless standards need different scalable properties like multiple code rates, multiple code lengths, fixed code lengths, different throughputs depending on the applications. LDPC codes can be designed to meet the above requirements. In addition to the strong theoretical interest in LDPC codes, such codes have already been adopted in satellite-based digital video broadcasting and long-haul optical communication standards, are highly likely to be adopted in the IEEE wireless local area network standard, and are under consideration for the long-term evolution of third generation mobile telephony. The idea behind these codes dates back to the sixties, but recently such coding schemes has been given a fresh analysis and it has been shown that they can approach the information theoretical limits at unprecedented low complexity. The name Low Density comes from the characteristic of their parity-check matrix which contains only a few 1's in comparison to the number of 0's. Their main advantage is that they provide a performance which is very close to the capacity for a lot of different channels and linear time algorithms for decoding. Furthermore they are suited for implementations that make heavy use of parallelism. They use parallel decoding and the simple computation operations are the main advantage of the LDPC codes.

1.2 PROJECT GOAL

The project aim is to implement the LDPC decoder in FPGA. In this project, hard decision decoding and the Soft decision decoding algorithms are analyzed in detail. Both algorithms are simulated using model sim soft ware. Hard decision decoding algorithm for 6 bit codeword and 32 bit code word are implemented in FPGA

1.3 SOFTWARES USED

- ModelSim PE 5.4e
- Xilinx ISE 9.2i
- Matlab R2008b

1.4 ORGANIZATION OF THE REPORT

- **Chapter 2** discusses about the LDPC codes.
- **Chapter 3** deals with the encoding and decoding of LDPC codes
- **Chapter 4** gives the introduction to VHDL language
- **Chapter 5** presents the simulation results
- **Chapter 6** gives the conclusion and future scope

CHAPTER 2

LOW DENSITY PARITY CHECK CODES

2.1 PARITY CHECK CODES

In communication systems the noise are added when the messages are passed over the channel. So different error correcting methods are introduced. Parity check method of error correction is one of the simplest methods. In parity check method we will only consider binary messages and so the transmitted messages consist of strings of 0's and 1's. The essential idea of forward error control coding is to augment these message bits with deliberately introduced redundancy in the form of extra check bits to produce a codeword for the message. These check bits are added in such a way that code words are sufficiently distinct from one another that the transmitted message can be correctly inferred at the receiver, even when some bits in the codeword are corrupted during transmission over the channel.

The simplest possible coding scheme is the single parity check code (SPC). The SPC involves the addition of a single extra bit to the binary message, the value of which depends on the bits in the message. In an even parity code, the additional bit added to each message ensures an even number of 1s in every codeword. More formally, for the 7-bit ASCII plus even parity code we define a codeword c to have the following structure:

$$c = [c1\ c2\ c3\ c4\ c5\ c6\ c7\ c8] \tag{2.1}$$

Where each ci is either 0 or 1, and every codeword satisfies the constraint

$$c1 \oplus c2 \oplus c3 \oplus c4 \oplus c5 \oplus c6 \oplus c7 \oplus c8 = 0 \tag{2.2}$$

Equation (2.2) is called a parity-check equation, in which the symbol $\oplus$ represents modulo-2 addition.

While the inversion of a single bit due to channel noise can easily be detected with a single parity check code, this code is not sufficiently powerful to indicate which bit, or indeed bits, were inverted. Moreover, since any even number of bit inversions produces a string

satisfying the constraint (2.2), patterns of even numbers of errors go undetected by this simple code. Detecting more than a single bit error calls for increased redundancy in the form of additional parity bits and more sophisticated codes contain multiple parity-check equations and each codeword must satisfy every one of them.

2.2 LDPC CODES

Low Density Parity Check (LDPC) codes are error checking and correcting codes, which show good error correcting performance approaching Shannon's limit. The matrix 'H' is called a parity-check matrix. In LDPC codes the H matrix is sparse. The number of ones in the matrix is lesser compared to the number of zeroes. Each row of H corresponds to a parity-check equation and each column of H corresponds to a bit in the codeword. Thus for a binary code with m parity-check constraints and length n codeword the parity-check matrix is an $m \times n$ binary matrix

2.3 REPRESENTATION OF LDPC CODES

Basically there are two different possibilities to represent LDPC codes. Like all linear block codes they can be described in two ways

- Matrix representation.
- Graphical representation.

2.3.1 Matrix Representation

The parity check constrain can be represented in the form of matrix with 1's and 0's. The matrix given in eqn (2.3) is a parity check matrix with dimension $n \times m$ for a (6, 2) code. We can now define two numbers describing these matrixes. W_r (row weight) represent the number of 1's in each row and W_c (column weight) represent number of ones in each column. For a matrix to be called low-density the two conditions $W_c << n$ and $W_r << m$ must be satisfied.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \qquad (2.3)$$

The matrix H is called a parity-check matrix. Each row of H corresponds to a parity-check equation and each column of H corresponds to a bit in the codeword. Thus for a binary code with 'm' parity-check constraints and length 'n' codeword's the parity-check matrix is an $m \times n$ binary matrix. The matrix is called sparse since the number of ones in the matrix is less compared to the number of zeroes. In matrix form a string y = [c1 .....cn] is a valid codeword for the code with parity-check matrix H if and only if it satisfies the matrix equation $Hy^T = 0$.

Matrix G is called the generator matrix of the code. The message bits are conventionally labeled by u = [u1, u2, ... ,uk], where the vector 'u' holds the 'k' message bits. Thus the codeword c corresponding to the binary message u = [u1u2u3] can be found using the matrix equation c = uG. For a binary code with 'k' message bits and length 'n' code words the generator matrix, G, is a k×n binary matrix. The ratio k/n is called the rate of the code. A code with k message bits contains 2^k code words. These code words are a subset of the total possible 2^n binary vectors of length n.

2.3.2 Graphical Representation

Tanner introduced an effective graphical representation for LDPC codes. This way of representing the codes is called the Tanner graph. Tanner graph methods are very easy in implementing the message passing between the nodes and the LDPC decoding

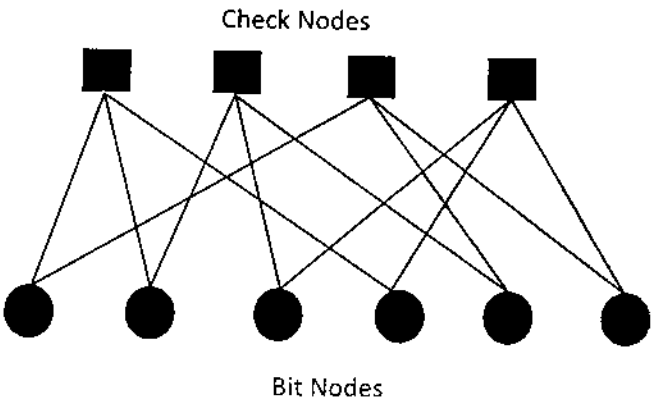


Figure 2.1: Tanner graph representation of parity check matrix

Tanner graphs are bipartite graphs. That means that the nodes of the graph are separated into two distinctive sets and edges are only connecting nodes of two different types. The two types of nodes in a Tanner graph are called variable nodes (v-nodes) and check nodes (C-nodes). Figure 2.1 is an example for such a Tanner graph and represents the same code as the matrix in (2.3). The creation of such a graph is straight forward. It consists of m check nodes (the number of parity bits) and n variable nodes (the number of bits in a codeword). Check node fi is connected to variable node c_j if the element h_{ij} of H is a 1.

The graph representation is analogous to a matrix representation by looking at the adjacency matrix of the graph, let H be a binary m x n matrix in which the entry (i; j) is 1 if and only if the ith check node is connected to the jth message node in the graph. Then the LDPC code defined by the graph is the set of vectors c = (c1.... cn) such that $H*c^T = 0$. The matrix H is called a parity check matrix for the code. Conversely, any binary m x n matrix gives rise to a bipartite graph between 'n' message and 'm' check nodes, and the code defined as the null space of H is precisely the code associated to this graph. Therefore, any linear code has a representation as a code associated to a bipartite graph (note that this graph is not uniquely defined by the code). However, not every binary linear code has a representation by a sparse bipartite graph if it does, then the code is called a low-density parity-check (LDPC) code. The sparsity of the graph structure is key property that allows for the algorithmic efficiency of LDPC codes

2.4 REGULAR AND IRREGULAR LDPC CODES

An LDPC code is called regular if W_c is constant for every column and $Wr = W_c * (n/m)$ is constant for every row. The example matrix from equation (2.3) is regular with $W_c = 2$ and $W_r = 3$. It's also possible to see the regularity of this code while looking at the graphical representation. There is the same number of incoming edges for every v-node and also for all the c-nodes. If the numbers of 1's in each row or column aren't constant, then the code is called an irregular LDPC code. The parity check matrix of LDPC codes is either regular or irregular. But the irregular LDPC parity check matrix gives better performance.

2.5 CONSTRUCTION OF LDPC CODES

Several different algorithms exist to construct suitable LDPC codes. Gallager introduced one. Furthermore MacKay proposed one to semi-randomly generate sparse parity check matrix. This is quite interesting since it indicates that constructing good performing LDPC codes is not a problem. In fact, completely randomly chosen codes are good with a high probability. The problem that will arise is that the encoding complexity of such codes is usually rather high.

LDPC codes can be constructed in two ways they are

- Random construction
    - MacKay Constructions
    - Bit filling Algorithm
    - Progressive Edge-Growth Algorithm
- Structured construction
    - Combinatorial Designs
    - Finite Geometry Designs
    - Algebraic Methods

These are the methods to construct the LDPC codes. In random construction, the H matrix is generated randomly. Therefore the number of ones in the row and the column need not be same. But in the structured construction the numbers of ones are arranged in a structured way. Irregular constructed LDPC codes will give better performance than the regular constructed LDPC codes

CHAPTER 3

ENCODING AND DECODING OF LDPC CODES

.1 ENCODING PROCESS

The LDPC encoder transforms each input message block 'u' into a distinct N-tuple (N-bit equence) code word 'c'. The codeword length N, where N > K, is then referred to as the block-ength. And, there are 2^k distinct code words corresponding to the 2^K message blocks. This set of ie 2^K code words is termed as a C(N,K) linear block code. The word linear signifies that the nodulo-2 sum of any two or more code words in the code C(N,K) is another valid codeword. he number of non-zero symbols of a codeword 'c' is called the weight, while the number of bit-ositions in which two code words differ is termed as the distance. The minimum distance of a near code is denoted by d_{min}, and determined by the weight of that codeword in the code (N,K), which has the minimum weight.

The unique and distinctive nature of the code words implies that there is a one-to-one napping between a K-bit information sequence 'u' and the corresponding N-bit codeword 'c' lescribed by the set of rules of the encoder.

A generator matrix 'G' is determined by performing Gauss–Jordan elimination on 'H' to btain it in the form:

$$H' = [A, I_{N-K}] \tag{3.1}$$

Where 'A' is a (N-K) × K binary matrix and I_{N-K} is the size N-K identity matrix. The generator natrix is then:

$$G = [A, I_{N-K}] \tag{3.2}$$

Since LDPC codes are linear, a codeword is generated by multiplying the input vector with the generator matrix,

$$c = uG \tag{3.3}$$

Where 'c' is the code word and 'u' is the input vector bits. Since 'G' matrix is not sparse, he matrix multiplication at the encoder will have complexity in the order of n^2 operations.

.2 DECODING OF LDPC CODES

The class of decoding algorithms used to decode LDPC codes is collectively termed nessage-passing algorithms (MPA) since their operation can be explained by the passing of nessages along the edges of a Tanner graph. Each Tanner graph node works in isolation, only having access to the information contained in the messages on the edges connected to it. The nessage-passing algorithms are also known as iterative decoding algorithms as the messages pass back and forward between the bit and check nodes iteratively until a result is achieved (or he process halted). Different message-passing algorithms are named for the type of messages passed or for the type of operation performed at the nodes. In some algorithms, such as bit-flipping decoding, the messages are binary and in others, such as belief propagation decoding, the messages are probabilities which represent a level of belief about the value of the code word bits.

It is often convenient to represent probability values as log likelihood ratios, and when this is done belief propagation decoding is often called sum-product decoding since the use of log likelihood ratios allows the calculations at the bit and check nodes to be computed using sum and product operations. The decoding algorithms are normally classified in to two they are hard decision algorithm and Soft decision algorithms. Soft decision algorithm which is based on the concept of belief propagation gives better decoding performance and therefore is a preferred method. The decoding can be done iteratively since the parity check matrix is sparse the LDPC codes have less complexity compared with Turbo codes

3.3 MESSAGE PASSING ALGORITHM

The messages passed along the Tanner graph edges are straightforward: a bit node sends the same outgoing message M to each of its connected check nodes. This message, labeled Mi. for the i-th bit node, declares the value of the bit '1' or '0'. The check nodes send back different messages to each of their connected bit nodes. This message, labeled $E_{j,i}$ for the message from

he j-th check node to the i-th bit node, declares the value of the i-bit '1' or '0' as determined by he j-th check node. If the bit node of an erased bit receives an incoming message which is '1' or 0' the bit node changes its value to the value of the incoming message. This process is until some maximum number of decoder iterations has passed and the decoder gives up.

The advantages of LDPC decoding algorithms are that they will use tanner graph and iterative decoding methods. They consist of two sets of nodes check nodes and bit nodes. They both are in different levels, Connected each other. Within one level there is no connection so the parallel processing can be done easily. This will speed up the decoding process

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

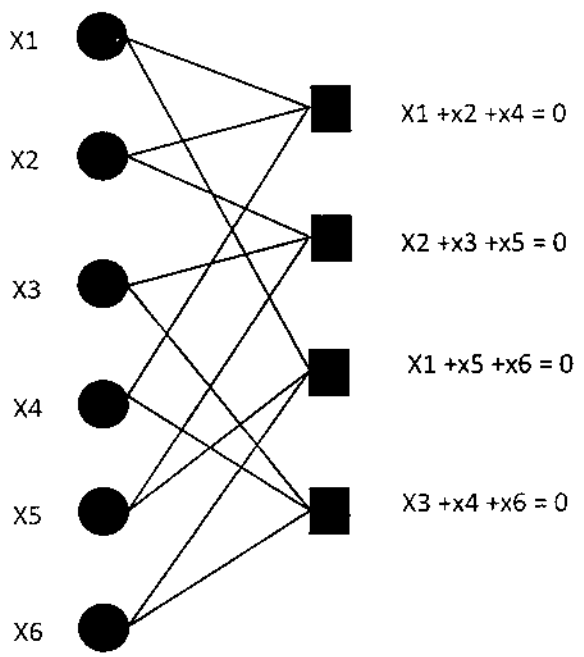


Figure 3.1: Representation of parity check constrain of LDPC codes

The notation B_j is used to represent the set of bits in the jth parity-check equation of the code. So for the parity check constrain shown in figure 3.1 we have

$B_1 = \{1, 2, 4\}$ $B_2 = \{2, 3, 5\}$ $B_3 = \{1, 5, 6\}$ $B_4 = \{3, 4, 6\}$.

Similarly, we use the notation A_i to represent the parity-check equations which check on the ith bit of the code. So for the parity check constrain shown in figure 3.1 we have

$A_1 = \{1, 3\}$ $A_2 = \{1, 2\}$ $A_3 = \{2, 4\}$
$A_5 = \{1, 4\}$ $A_5 = \{2, 3\}$ $A_6 = \{3, 4\}$.

Algorithm outlines message-passing decoding on the BEC. Input is the received values from the detector, y = [y1, . . . , yn] which can be '1', '0', and output is M = [M1, . . . ,Mn] which can also take the values '1', '0'.

The datas are sometime send over the erasure channel. The MPA algorithm will help to find the erased bit. Thus the message passing algorithm (MAP) helps to find out the reassured bits at the decoder. This message passing algorithm can be used in erasure channel where the received bits can be '0','1' or x the unknown bit. The specialty of the channel is that it will receive either a receive either a true bit or bit x. It will not produce an error bit. The unknown bit x can be found out by the message passing algorithm by passing of messages between bit nodes and check nodes

3.4 HARD DECISION DECODING

The bit-flipping algorithm is a hard-decision message-passing algorithm for LDPC codes. A binary (hard) decision about each received bit is made by the detector and this is passed to the decoder. For the bit-flipping algorithm the messages passed along the Tanner graph edges are also binary. A bit node sends a message declaring if it is a one or a zero, and each check node sends a message to each connected bit node, declaring what value the bit is based on the information available to the check node. The check node determines that its parity-check

quation is satisfied if the modulo-2 sum of the incoming bit values is zero. If the majority of the nessages received by a bit node are different from its received value the bit node changes (flips) ts current value. This process is repeated until all of the parity-check equations are satisfied, or ntil some maximum number of decoder iterations has passed and the decoder gives up.

The bit-flipping decoder can be immediately terminated whenever a valid code word has been found by checking if all of the parity-check equations are satisfied. This is true of all nessage-passing decoding of LDPC codes and has two important benefits; firstly additional terations are avoided once a solution has been found, and secondly a failure to converge to a code word is always detected. The bit-flipping algorithm is based on the principle that a code word bit involved in a large number of incorrect check equations is likely to be incorrect itself. The sparseness of H helps spread out the bits into checks so that parity-check equations are unlikely to contain the same set of code word bits. The bit-flipping algorithm applies the hard decision on the received vector, y = [y1, . . . , yn], and output is M = [M1, . . . , Mn].

The steps of the message passing algorithm is given below

Step 1: Initialization

Step 2: Check-node update

Step 3: Variable-node update

Step 4: Decision

Step 1: initialization

This is the first phase of MPA. In this phase in tanner graph the bit nodes are assigned the value of the received code word, this can or cannot be true. Then the bit nodes will send the information in to the corresponding check nodes to which they are connected .at the check node exor operations are performed. If all the result of the exor operation is zero then what ever code word we got is the actual code word or else there is an error in the code word which have to be corrected. So messages are passed between the bit nodes and the check nodes

Initialization

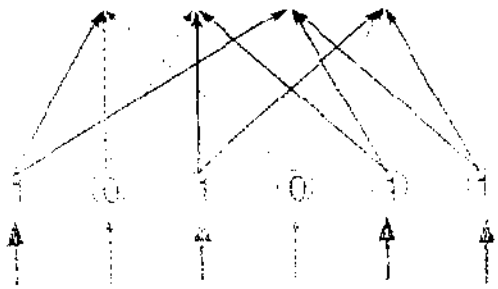


Figure 3.2: Initialization of the bit node

n the case of received bits {1 0 1 0 1 1} the value of the check bits are

$B_0 = 1$ $B_1 = 0$ $B_2 = 1$ $B_3 = 0$

Since all the bits in this case is not zero this is not satisfying the paritycheckequations and this is not the actual code word

Step 2: Check-node update

This is the next step in decoding. The check nodes will send the values they hold to all the bit nodes to which they have connected. E_{ij} is the value passed from the jth check node to the ith bit node. Since one check node is connected to three bit nodes .it will take the incoming value from any two of the bit node and exor in and passed to the third one .this can be summarised in terms of all E_{ij}'s

$E_{11} = 0$ E31 = 0

$E_{22} = 0$ E12 = 0

$E_{23} = 1$ E43 = 1

$E_{14} = 1$ E44 = 0

$E_{25} = 1$ E35 = 0

$E_{36} = 0$ E46 = 1

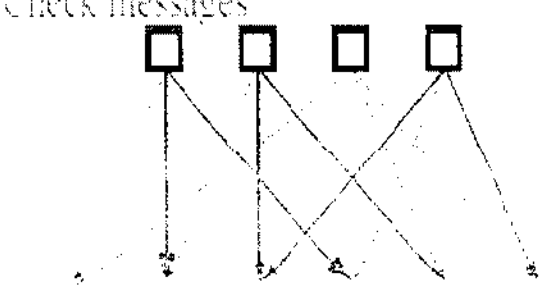


Figure 3.3: Check node message updates

Step 3: Variable-node update

The variable node values are up dated by looking the message from the check nodes .this will look maximum polling algorithm. That means each bit node will get messages from the two check nodes .That is two bits it can be of four different combinations they are {0,0} {0,1} {1,0} {1,1}.thus if the update from the check nodes are {1,0} or {0,1} whatever we received at the receiver is taken as the correct. But when the received information from the check node are {1, 1} by maximum polling algorithm we will take the correct bit as '1' whatever we received. Similarly in the case of {0,0} we will take the error free received bit as '0' for whatever we received

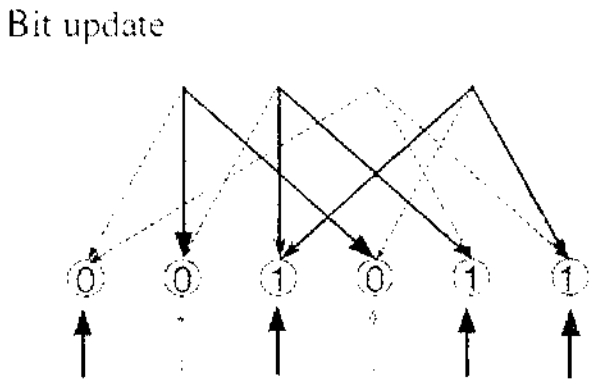


Figure 3.4: Variable-node update

Step 4: Decision

In this step the decisions will take. This is that by the new updated value of the received code word will be sending to check nodes again for the checking of correction .here we got that all the B values are zero so we represent it by the tic mark. Thus the error correction of the code is done

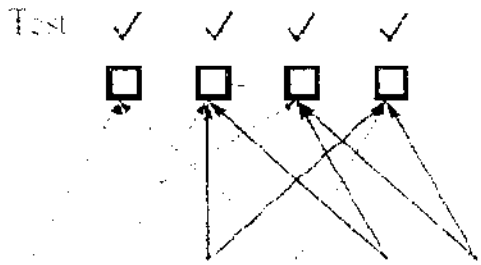


Figure 3.5: Decision making

Bit-flipping decoding of the received string y = [1 0 1 0 1 1]. Each sub-figure indicates the decision made at each step of the decoding algorithm based on the messages from the previous step. A cross represents that the parity check is not satisfied while a tick indicates that it is satisfied. For the messages, a dashed arrow corresponds to the messages "bit = 0" while a solid arrow corresponds to "bit = 1".Thus by repeated message passing between the check nodes and the bit nodes we can finally able to tell the received code word is correct or not. If there is any error in the code word then the algorithm will correct the errors. Since there is no connection with in the bit nodes and the check nodes and only connection between them the iterative decoding is easy in this case

In the previous case the hard decision algorithm is done with 4 x6 parity check matrix. The code word length is 6 bits. Hard decision decoding is extended up to 32 bit code word; the corresponding parity check matrix dimension is 16 x 32. In the case of LDPC codes as the dimension of the H, the parity check matrix increases, the performance shown by the code is better

.5 SOFT DECISION DECODING

It is convenient to represent probability values as log likelihood ratios, and when this is one belief propagation decoding is often called sum-product decoding since the use of log kelihood ratios allows the calculations at the bit and check nodes to be computed using sum and roduct operations. Soft decision algorithm which is based on the concept of belief propagation ives better decoding performance and therefore is a preferred method.

The sum-product algorithm is a soft decision message-passing algorithm. It is similar to he bit-flipping algorithm. But with the messages representing each decision (check met, or bit value equal to 1) now probabilities. The sum-product algorithm is a soft decision algorithm which accepts the probability of each received bit as input. The input bit probabilities are called he a priori probabilities for the received bits because they were known in advance before unning the LDPC decoder. The bit probabilities returned by the decoder are called the posterior robabilities. In the case of sum-product decoding these probabilities are expressed as log-ikelihood ratios.

For a binary variable x it is easy to find p(x=1) given p(x=0), since p(x=1) = 1−p(x=0) and so we only need to store one probability value for x. Log likelihood ratios are used to represent the metrics for a binary variable by a single value

$$p(x=0)=\frac{p(x=1)/p(x=0)}{1+p(x=1)/p(x=0)}=\frac{e^{-L(x)}}{1+e^{-L(x)}} \tag{3.4}$$

And

$$p(x=1)=\frac{p(x=0)/p(x=1)}{1+p(x=0)/p(x=1)}=\frac{e^{L(x)}}{1+e^{L(x)}} \tag{3.5}$$

The benefit of the logarithmic representation of probabilities is that when probabilities ced to be multiplied log-likelihood ratios need only be added, reducing implementation omplexity .The sum-product algorithm iteratively computes an approximation of the MAP alue for each code bit. Input is the log likelihood ratios for the a priori message probabilities. he a priori probabilities for the Binary Symmetric channel (BSC) are:

$$ri = \log p_i/(1-p_i) \text{ if } y_i = 1 \qquad \text{Or}$$

$$ri = \log(1-p)/(p) \text{ if } y_i = 0 \tag{3.6}$$

In sum-product decoding the extrinsic message from check node j to bit node i, E_{ji}, is the LR of the probability that bit i causes parity-check j to be satisfied.

$$E_{j,i} = \log\left(\frac{1+\prod_{i' \in B_j, i' \neq i} \tanh(Mj,i'/2)}{1-\prod_{i' \in B_j, i' \neq i} \tanh(Mj,i'/2)}\right) \tag{3.7}$$

The intrinsic message from check node j to bit node i, M_{ji}, is given by,

$$M_{j,i} = \sum_{j' \in Ai, j' \neq j} E_{j',i} + r_i \tag{3.8}$$

The total LLR of the bit stream is

$$Li = \sum_{j \in A_i} E_{j,i} + r_i \tag{3.9}$$

The total LLR can be either positive or negative number. The hard decision is taken. When total LLR is positive the decision is '0' else '1'. The code word is z. Then syndrome calculation is done by s=zH'. When s is zero then z is a valid codeword, and the decoding stops, returning z as the decoded word. For an AWGN channel the a priori LLRs are given by

$$r_i = 4y_i(E_s/N_o) \tag{3.10}$$

The extrinsic LLR and the total LLR calculation are done to find the codeword .

Various steps of Sum Product Decoding algorithm is given below

ocedure DECODE(r)

= 0

r i = 1 : n do

for j = 1 : m do

$M_{j,i} = r_i$

end for

nd for

epeat

or j = 1 : m do

for i ∈ Bj do

$$E_{j,i} = \log\left(\frac{1+\prod_{i' \in B_j, i' \neq i} \tanh\left(M_{j,i'}/2\right)}{1-\prod_{i' \in B_j, i' \neq i} \tanh\left(M_{j,i'}/2\right)}\right)$$

end for

end for

for i = 1 : n do

$L_i = \sum_{j \in A_i} E_{j,i} + r_i$

$$z_i = \begin{cases} 1, L_i \leq 0 \\ 0, L_i > 0 \end{cases}$$

end for

for i = 1 : n do

for $j \in A_i$ do

$M_{j,i} = \sum j' \in A_i, j' \neq j \; E_{j',i} + r_i$

end for

end for

$I = I + 1$

end if

until Finished

end procedure

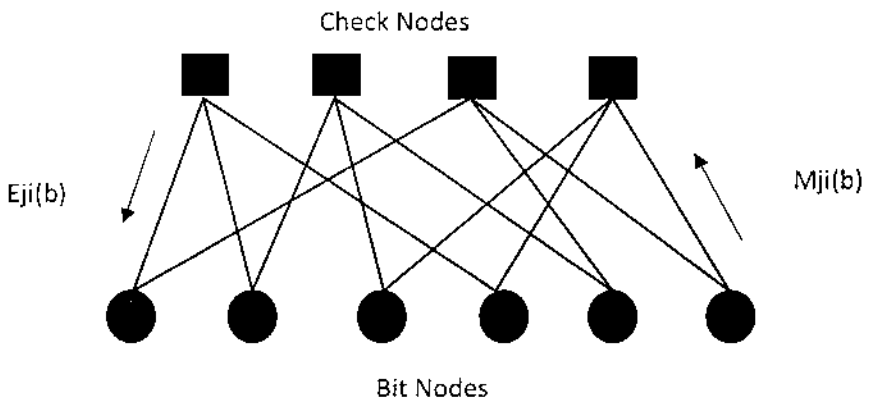


Figure 3.6: Intrinsic and Extrinsic information transfer between bit nodes and the check nodes

The figure 3.6 shows the intrinsic and extrinsic information transfer between bit nodes and the check nodes. Finally the total LLR is calculated and the decision is made.

CHAPTER 4

INTRODUCTION TO VHDL

VHDL is an acronym which stands for VHSIC Hardware Description Language. VHSIC s yet another acronym which stands for Very High Speed Integrated Circuits. It is being used for documentation, verification, and synthesis of large digital designs. VHDL is a standard (VHDL-1076) developed by IEEE. The different approaches in VHDL are structural, data flow, and behavioral methods of hardware description.

4.1 STRUCTURAL DESCRIPTIONS

The structural descriptions are explained below with examples.

4.1.1 Building Blocks

An entity declaration, or entity, combined with architecture or body constitutes a VHDL model. VHDL calls the entity-architecture pair a design entity. By describing alternative architectures for an entity, we can configure a VHDL model for a specific level of investigation. The entity contains the interface description common to the alternative architectures. It communicates with other entities and the environment through ports and generics. Generic information particularizes an entity by specifying environment constants such as register size or delay value. For example,

```
entity A is
        port (x, y: in real; z: out real);
        generic (delay: time);
end A;
```

The architecture contains declarative and statement sections. Declarations form the region before the reserved word begin and can declare local elements such as signals and components. Statements appear after begin and can contain concurrent statements. For instance,

```
architecture B of A is
  component M
```

```
signal a,b,c real := 0.0;
begin
"concurrent statements"
end B;
```

The variety of concurrent statement types gives VHDL the descriptive power to create and combine models at the structural, dataflow, and behavioral levels into one simulation model. The structural type of description makes use of component instantiation statements to invoke models described elsewhere. After declaring components, we use them in the component instantiation statement, assigning ports to local signals or other ports and giving values to generics. Invert: M port map (j => a ; k => c); We can then bind the components to other design entities through configuration specifications in VHDL's architecture declarative section or through separate configuration declarations. The dataflow style makes wide use of a number of types of concurrent signal assignment statements, which associate a target signal with an expression and a delay. The list of signals appearing in the expression is the sensitivity list; the expression must be evaluated for any change on any of these signals. The target signals obtain new values after the delay specified in the signal assignment statement. If no delay is specified, the signal assignment occurs during the next simulation cycle:

```
c <= a + b after delay;
```

VHDL also includes conditional and selected signal assignment statements. It uses block statements to group signal assignment statements and makes them synchronous with a guarded condition. Block statements can also contain ports and generics to provide more modularity in the descriptions. We commonly use concurrent process statements when we wish to describe hardware at the behavioral level of abstraction. The process statement consists of declarations and procedural types of statements that make up the sequential program. Wait and assert statements add to the descriptive power of the process statements for modeling concurrent actions:

```
process
begin
variable i : real := 1.0;
wait on a;
i = b * 3.0;
c <= i after delay;
end process;
```

Other concurrent statements include the concurrent assertion statement, concurrent procedure call, and generate statement. Packages are design units that permit types and objects to be shared.

4.2 DATA FLOW DESCRIPTIONS

In the data flow approach, circuits are described by indicating how the inputs and outputs of built-in primitive components are connected together.

4.2.1 An Example For Data Flow Approach

Suppose we were to describe the following SR latch using VHDL as in the following schematic.

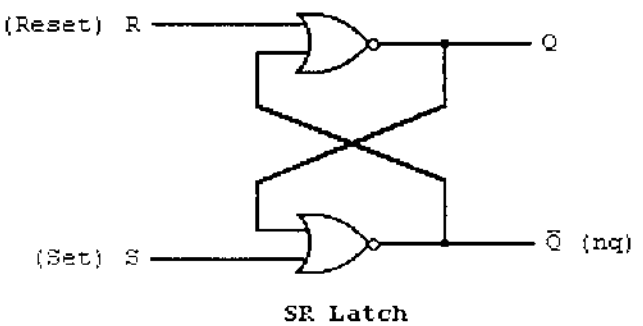


Figure 4.1: Data flow approach in SR Latch

```
entity latch is
```

```
    port (s,r : in bit;
    q,nq : out bit);
  end latch;
architecture dataflow of latch is
    begin
    q<=r nor nq;
    nq<=s nor q;
end dataflow;
```

The signal assignment operator in VHDL specifies a relationship between signals, not a transfer of data as in programming languages. The architecture part describes the internal operation of the design.The scheme used to model a VHDL design is called discrete event time simulation. In this the values of signals are only updated when certain events occur and events occur at discrete instances of time. The above mentioned SR latch works with this type of simulation.

4.2.2 The Delay Model

This section refers to the delay model. The two models of delay are used in VHDL. The first is called the inertial delay model. The inertial delay model is specified by adding an after clause to the signal assignment statement. The second is called transport delay model.

4.3 BEHAVIOURAL DESCRIPTIONS

The behavioural approach to modelling hardware components is different from the other two methods in that it does not necessarily in any way reflect how the design is implemented.

4.3.1 The Process Statement

It is basically the black box approach to modeling. It accurately models what happens on the inputs and outputs of the black box, but what is inside the box (how it works) is irrelevant. The behavioral description is usually used in two ways in VHDL. First, it can be used to model complex components.

Behavioral descriptions are supported with the process statement. The process statement can appear in the body of an architecture declaration just as the signal assignment statement

does. The process statement can also contain signal assignments in order to specify the outputs of the process.

4.3.2 Using Variables

A variable is used to hold data and also it behaves like you would expect in a software programming language, which is much different than the behavior of a signal. Although variables represent data like the signal, they do not have or cause events and are modified differently. Variables are modified with the variable assignment

4.3.3 Sequential Statements

There are several statements that may only be used in the body of a process. These statements are called sequential statements because they are executed sequentially. The types of statements used here are if, if else, for and loop.

4.3.4 Signals And Processes

This section is short, but contains important information about the use of signals in the process statement. A signal assignment, if anything, merely schedules an event to occur on a signal and does not have an immediate effect. When a process is resumed, it executes from top to bottom and no events are processed until after the process is complete.

4.3.5 Program Output

In most programming languages there is a mechanism for printing text on the monitor and getting input from the user through the keyboard. It can able to give output certain information during simulation. A standard library that comes with every VHDL language system. In VHDL, common code can be put in a separate file to be used by many designs. This common code is called a library. In order to use the library that provides input and output capabilities you must add the statement use textio.all; immediately before every architecture that uses input and output. The write statement can be used to append constant values and the value of variables and signals of the types bit, bit vector, time, integer, and real.

CHAPTER 5

SIMULATION RESULTS

The message bits are encoded and transmitted over the noisy channel. And at the input of the receiver, the received bits are decoded checked whether they are corrupted or not. The parity check matrix used here is

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Simulation result for received code word [1 0 1 0 1 1] is shown in figure 5.1.The result shows that this received code word is not correct .It shows that the check node update is initially [1 0 1 0]. And the algorithm will correct the result as [0 0 1 0 1 1], which is the actual code word

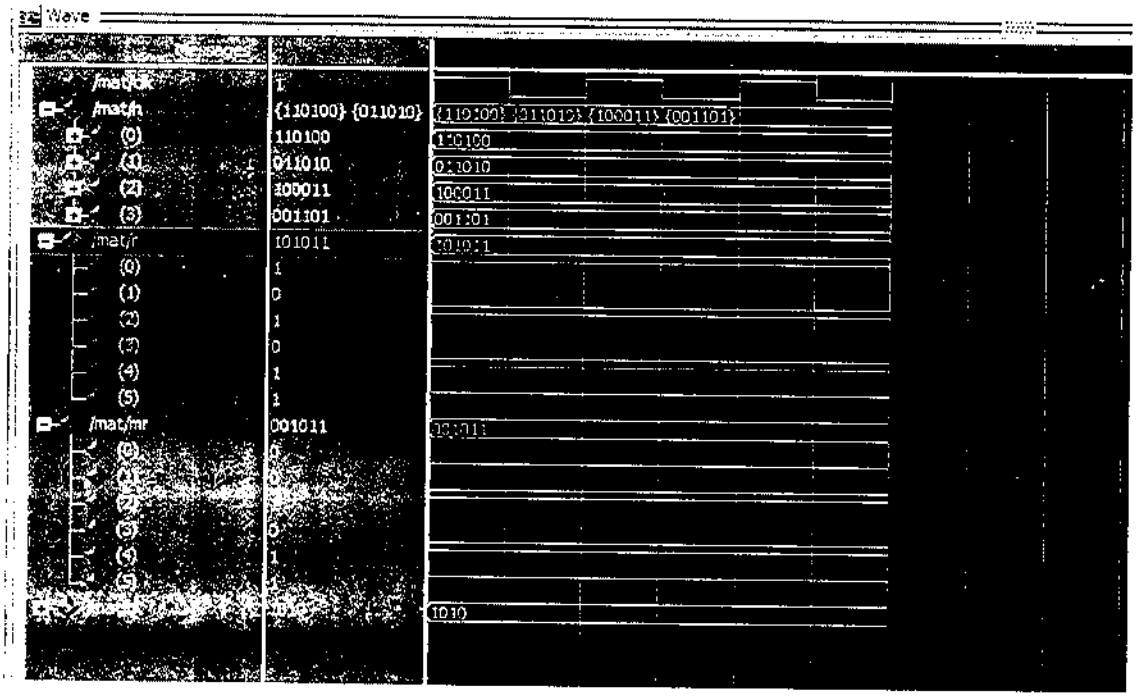


Figure 5.1: Simulated output for the code word [1 0 1 0 1 1]

Simulation result for received code word [0 0 1 0 1 1] is shown in figure 5.2.The result shows that the check node update here is [0 0 0 0] so the received code word is correct and it is the actual code word. Here also we are using the same parity check matrix in the above case

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

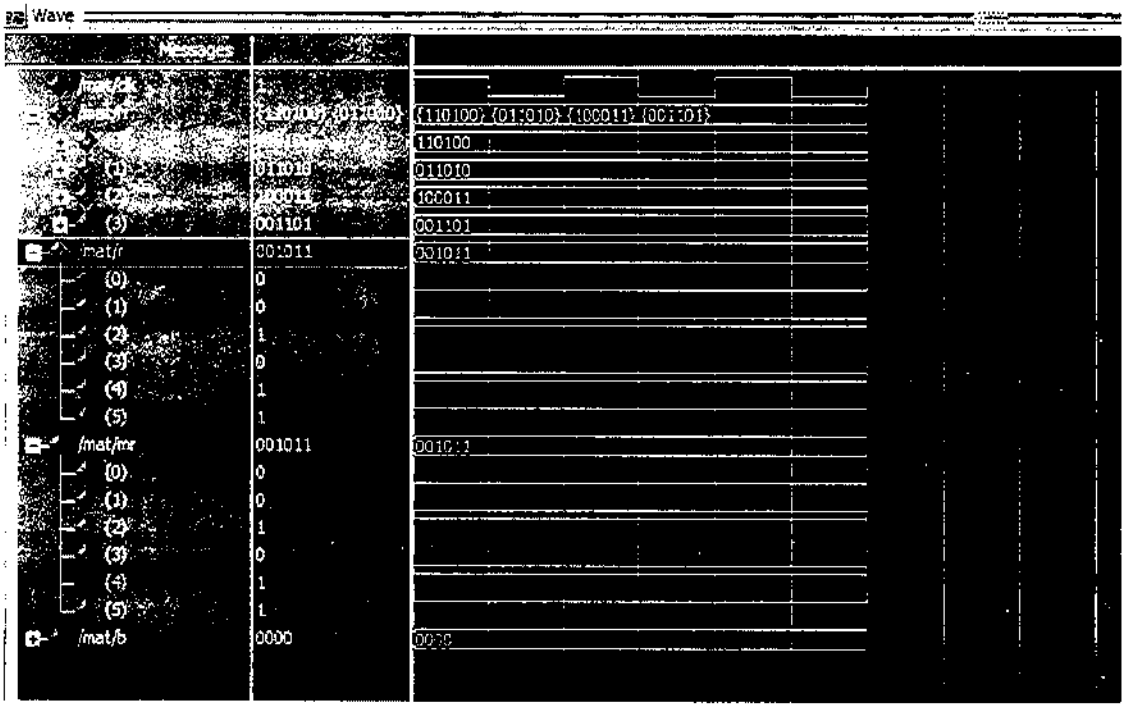


Figure 5.2: Simulated output for code word [0 0 1 0 1 1]

Simulation result of 32 bit code word length hard decision decoding is shown .The parity check matrix used here is of dimension 16 x 32 is shown below

$$
\begin{bmatrix}
1001110101010110100000000000000000 \\
0011100101101110010000000000000000 \\
1101011000100000001000000000000000 \\
0111010001010001000100000000000000 \\
0100101101001011000010000000000000 \\
1011010101100000000000100000000000 \\
1100101001100000000000010000000000 \\
0111100101101110000000001000000000 \\
0110001101011001000000000100000000 \\
0100000110001010000000000010000000 \\
0000000010111011000000000001000000 \\
1011101001000100000000000000010000 \\
0111110010010101000000000000001000 \\
0101000000001000100000000000000100 \\
0001101001001100000000000000000010 \\
0001101001110111000000000000000001
\end{bmatrix}
$$

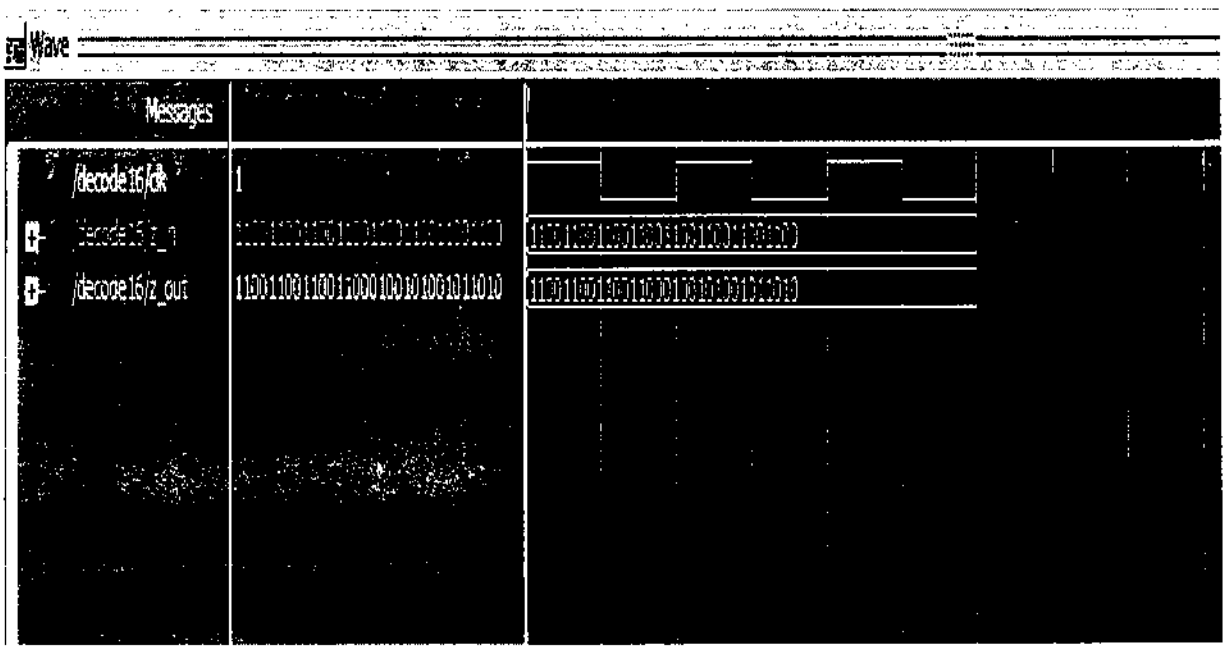


Figure 5.3: Simulated output for 32 bit code word length hard decision decoding

Simulation result of 6 bit soft decision decoding is shown below. The parity check matrix of this is given by

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

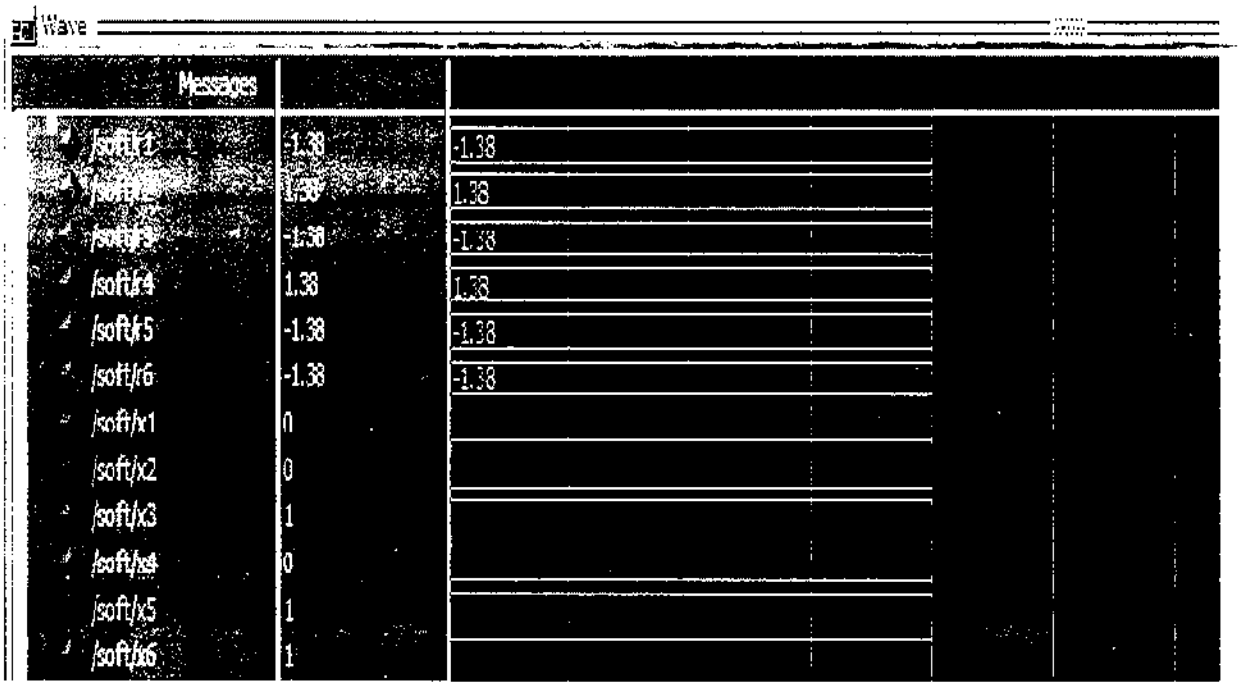


Figure 5.4: Simulated output for soft decision decoding

Synthesis report of hard decision algorithm is shown below. The first table shows the hard decision algorithm of 6 bit code length and the second table represents the hard decision algorithm of 32 bit code length. Various resources and their utilization is shown in table 5.1 and 5.2.

TABLE 5.1: SYNTHESIS REPORT OF 6 BIT CODE LENGTH HARD DECISION ALGORITHM

Resources	Utilization
Number of slices	27/960= 2%
Number of slices flip flops	22/1960=1%
Number of 4 input LUTs	46/1920=2%
Number of bonded IOBs	37/66=56%
Number of GCLKs	1/24=4%

TABLE 5.2: SYNTHESIS REPORT OF 32 BIT CODE LENGTH HARD DECISION ALGORITHM

Resources	Utilization
Number of slices	768/1197= 64%
Number of 4 input LUTs	1536/2100=73%
Number of bonded IOBs	63/64=98%

CHAPTER 6

CONCLUSION

Low density-parity check codes are efficient error correcting codes. These codes can be decoded in iterative way. The iterative decoding approach is already used in turbo codes but the structure of LDPC codes give even better results. In this project, the two different decoding methods of LDPC codes are studied. The two methods are hard decision decoding and the soft decision decoding. These two decoding method are analyzed and simulation results are taken from model sim soft ware. For hard decision decoding both 6 bit code word, of which the parity check dimension 4 x 6 and 32 bit code word .of which parity check matrix dimension 16 x 32 are simulated. Usually in case of communication noises are added in between the transmitter and the receiver. So at the receiver the decoding task is very tuff. So at that time priory probability of the received code word is taken as input to the decoder for getting better performance. So we are using soft decision decoding algorithm. In this project 6 bit code word is given to soft decision decoder and performance is analyzed.

REFERENCES

1. Gallager, "*Low-density parity-check codes,*" IRE Transactions on Information Theory, vol. IT-8, no.1, pp. 21–28, January 1962

2. R. G. Gallager, "*Low-Density Parity-Check Codes*", Cambridge, MA:MIT Press, 1963.

3. Daesun,"*Low Complexity Switch Network for Reconfigurable LDPC Decoder*" IEEE Trans. vol.18, no.1, January 2010

4. Jin sha, "*Multi Gb/s LDPC Code design and implementation*" IEEE Trans vol.17, no 2 February 2009

5. Chen, "*Reduced complexity decoding of LDPC codes*" IEEE Trans.commun, vol 53, aug.2005

6. T. J. Richardson and R. L. Urbanke, "*The capacity of low-density parity check codes under message passing decoding,*" IEEE Trans. Inform. Theory, vol. 47, no. 2, pp. 599–618, February 2001.

7. T. J. Richardson and R. L. Urbanke, "*Efficient encoding of low-density parity-check codes,*" IEEE Trans. Inform. Theory, vol. 47, no. 2, pp. 638– 656, February 2001.

8. D. J. C. MacKay and R. M. Neal, "*Near Shannon limit performance of low density parity check codes,*" Electron. Lett., vol. 32, no. 18, pp. 1645–1646, March 1996, reprinted Electron. Lett, vol. 33(6), pp.457–458, March 1997.

9. M. Yang, Y. Li, and W. E. Ryan, "*Design of efficiently encodable moderate-length high-rate irregular LDPC codes,*" IEEE Trans. Commun., vol. 52, pp. 564-571, Apr. 2004

10. F.R. Kschischang, B.J. Frey and H.-A. Loeliger, "*Factor graphs and the sum-product algorithm*", IEEE Trans. Inform. Theory, vol 47, pp. 498-519, Feb. 2001.

11. M. Chiani and A. Ventura, "*Design and performance evaluation of some high-rate irregular low-density parity-check codes,*" in IEEE GLOBECOM, 2001, vol. 2, pp. 990-994.

12. M.P.C. Fossorier and M. Mihaljevic, "*Reduced complexity iterative decoding of low-density parity-check codes based on belief propagation,*" *IEEE Trans. Commun.*, vol. 47, May 1999.

13. S. Ten Brink, G. Kramer and A. Ashikhmin, "*Design of low-density parity-check codes for modulation and detection*", IEEE Trans. Commun., vol. 52, pp. 670-678, April 2004.

14. R. Narayanaswami, "*Coded modulation with low-density parity-check codes,*" Master's thesis, Dept. Elect. Eng., Texas A&M Univ., College Station, TX, 2001.

15. R. M. Tanner, "*A recursive approach to low complexity codes,*" IEEE Trans. Inform. Theory, vol. IT-27, no. 5, pp. 533–547, September 1981.

16. Todd K. Moon, "*Error Correction Coding, Mathematical Methods and Algorithms*", A John Wiley & Sons, Inc, Publication, New Delhi, India, Ed.2006.

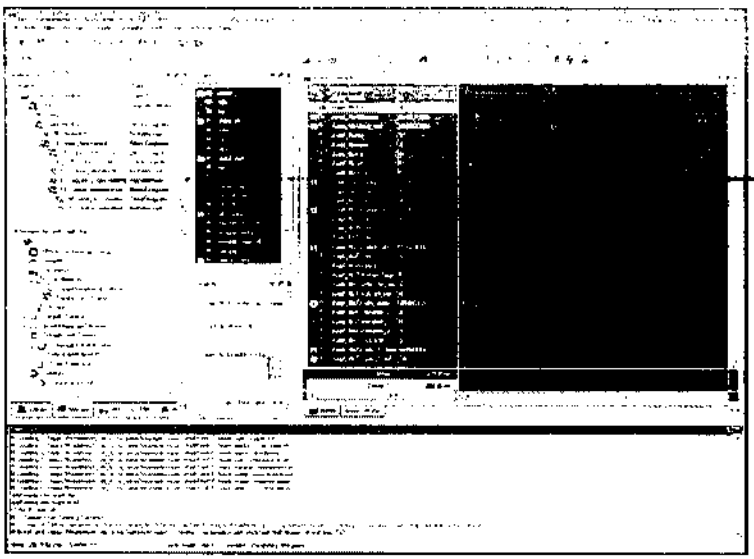
17. M. G. Luby, M. Mitzenmacher, A. Shokrollahi, and D. A. Spielman, "*Improved low-density parity-check codes using irregular graphs and belief propagation*", Technical Report TR-97-044, Berkeley, CA, 1997.

APPENDIX I

Design Creation to Realization

ModelSim Designer

D A T A S H E E T



ModelSim Designer combines easy-to-use, flexible creation with a powerful verification and debug environment

Major product features:

- Project manager, version control, and source code templates and wizards
- [illegible]
- Intuitive graphical waveform editor
- Automated testbench creation
- Text to graphic rendering facilitates analysis
- HTML Documentation option
- VHDL, Verilog, and mixed-language simulation
- Optimized native compiled architecture
- Single Kernel Simulator technology
- Advanced debug including Signal Spy™
- Memory window
- Easy-to-use GUI with Tcl interface
- Support for all major synthesis tools and the Actel, Altera, Lattice, and Xilinx place-and-route flows
- Built-in FPGA vendor library compiler
- Full support for Verilog 2001
- Windows platform support

***Options:** SWIFT Interface support, graphics-based Dataflow window, Waveform Compare, integrated code coverage, and Profiler (for more details on options, please see the ModelSim SE datasheet)*

The Easy to Use Solution for the FPGA Designer

ModelSim® Designer is a Windows®-based design environment for FPGAs. It provides an easy to use, advanced-feature tool at an entry-level price. The complete process of creation, management, simulation, and implementation are controlled from a single user interface, facilitating the design and verification flow and providing significant productivity gains.

ModelSim Designer combines the industry-leading capabilities of the ModelSim simulator with a built-in design creation engine. To support synthesis and place-and-route, ModelSim Designer is plug-in ready for the synthesis and place-and-route tools of your choice. Connection of these additional tools is easy, giving FPGA designers control of design creation, simulation, synthesis, and place-and-route from a single cockpit.

A flexible methodology conforms to existing design flows. Designers can freely mix text entry of VHDL and Verilog code with graphical entry using block and state diagram editors. A single design unit can be represented in multiple views, and the management of compilation and simulation at all levels of abstraction is a single click away.

www.model.com/modelsimdesigner

Mentor Graphics

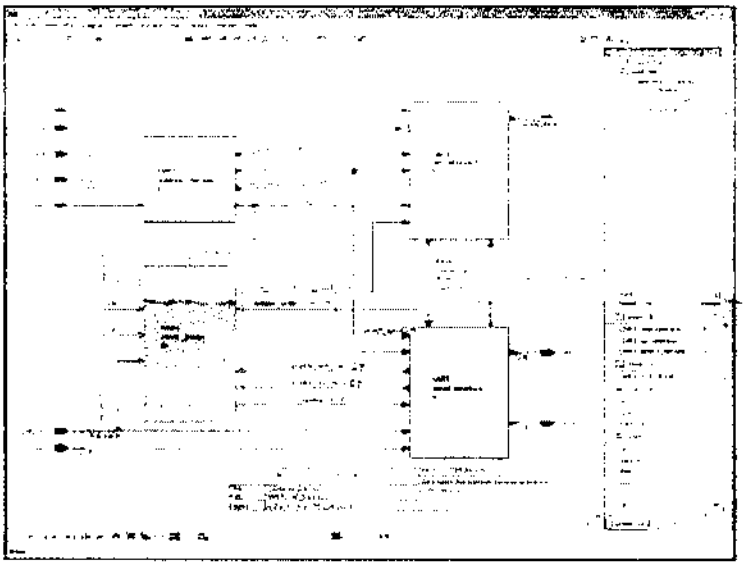
Implementation tasks are achieved through tight integration with the most popular FPGA synthesis and place-and-route tools in the industry. ModelSim Designer can either run these tools directly or automatically via scripts. Either way, the design details are stored and used in a consistent and extensible way.

Project Manager

The flexible and powerful project manager feature allows easy navigation through a design in order to understand design content and execute all necessary tasks. The project manager provides easy access to all design data for each individual design unit throughout the design process. Synthesis results, place-and-route results, back-annotated netlists, and SDF files are associated with the relevant design unit. Project archiving is a few simple clicks away, allowing complete version-control management of designs. The project manager also stores user-generated design documents, such as Word, PowerPoint, and PDFs.

Templates and Wizards

Creation wizards walk users through the creation of VHDL and Verilog design units using either text or graphics. In the case of the graphical editor, HDL code is generated from the graphical diagrams created. For text-based design, VHDL and Verilog templates and wizards help engineers quickly develop HDL code without having to remember the exact language syntax. The wizards show how to create parameterizable logic blocks, testbench stimuli, and design objects. Novice and advanced HDL developers both benefit from time-saving shortcuts.



The Block Diagram editor is a convenient way to visually partition your design and have the HDL code automatically generated.

Intuitive Graphical Editors

ModelSim Designer includes block diagram and state machine editors that ensure a consistent coding style, facilitating design reuse and maintenance. These editors use an intuitive, graphical methodology that flattens the learning curve. This shortens the time to productivity for designers who are migrating to HDL methodologies or changing their primary design language.

Designers can automatically view or render diagrams from HDL code in block diagrams or state machines. When the code changes, the diagram can be updated instantly with its overall intent ensured. This helps designers understand legacy designs and aids in the debugging of current designs.

Automated Testbench Creation

ModelSim Designer offers an automated methodology for testbench generation. The testbench wizard generates VHDL or Verilog code through a graphical waveform editor, with output in either HDL or a TCL script. Users can manually define signals in the waveform editor or use the built-in wizard to define the waveforms, either way, it is intuitive and easy to use and saves considerable time.

Active Design Visualization Enhances Simulation Debugging

During live simulation, design analysis capabilities are enhanced through graphical design views. From any diagram window, simulations can be fully executed and controlled. Enhanced debugging features include graphical breakpoints, signal probing, graphics-to-text-source cross-highlighting, animation, and cause analysis. The ability to overlay live simulation results in a graphical context speeds up the debug process by allowing faster problem discovery and better design iterations.

HTML Documentation

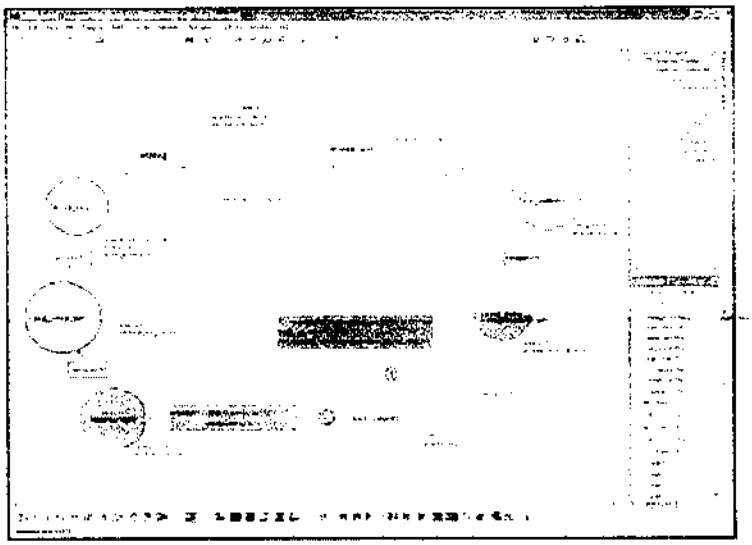
The HTML (HyperText Markup Language) documentation [illegible] allows the user to compile a complete design in HTML format. This allows the graphical [illegible] to be viewed with standards that use an HTML browser. Design Hierarchy and details can be viewed and navigated in an HTML browser. [illegible] and documentation.

Memory Window

The memory window enables easy file viewing and switching of memory locations. VHDL and Verilog variables are automatically extracted in the GUI, delivering powerful search, fill, load, and save functionality. The memory window allows pre-loading of memories, thus saving the time-consuming step of initializing sections of the simulation to load memories. All functions are available via the command line, making them available for scripting.

Intelligent GUI

An intelligently engineered GUI makes efficient use of desktop real estate. ModelSim Designer's intuitive layout of graphical elements (window, toolbars, menus, etc.) makes it easy to step through the design flow. There are also wizards in place which help set up the design environment and make going through the design process seamless and efficient.



The ModelSim Designer user interface provides a rich environment for simulation through the post-synthesis and post-place & route stages of the design process.

Synthesis and Place-and-Route Integration

The industry's popular FPGA synthesis tools, such as Mentor Graphics Precision-RTL and most FPGA vendor synthesis tools, can be integrated into ModelSim Designer with push button convenience. Actel Designer and Libero, Altera Quartus, Lattice ispLEVER, and Xilinx ISE place-and-route software are similarly integrated. Place-and-route results, together with SDF information, are automatically managed by ModelSim Designer after the process is complete making them ready for post-place-and-route gate-level simulation.

FPGA Vendor Library Compiler

ModelSim Designer provides an intuitive mechanism to compile the necessary vendor libraries for post-place-and-route simulation. The compiler detects which FPGA vendor tools have been installed and compiles the necessary libraries as soon as the tool is launched, taking care of this step once and for all.

Visit our web site at www.modelsim.com/products/designer for more information.

APPENDIX II

XILINX®

Spartan-3 FPGA Family Data Sheet

DS099 December 4, 2009 | **Product Specification**

This document includes all four modules of the Spartan-3 FPGA data sheet.

Module 1: Spartan-3 FPGA Family: Introduction and Ordering Information

DS099-1 (v2.5) December 4, 2009

- Introduction
- Features
- Architectural Overview
- Array Sizes and Resources
- User I/O Chart
- Ordering Information

Module 2: Spartan-3 FPGA Family: Functional Description

DS099-2 (v2.5) December 4, 2009

- Input/Output Blocks (IOBs)
  - IOB Overview
  - Select OTM Interface I/O Standards
- Configurable Logic Blocks (CLBs)
- Block RAM
- Dedicated Multipliers
- Digital Clock Manager (DCM)
- Clock Network
- Configuration

Module 3: Spartan-3 FPGA Family: DC and Switching Characteristics

DS099-3 (v2.5) December 4, 2009

- DC Electrical Characteristics
  - Absolute Maximum Ratings
  - Supply Voltage Specifications
  - Recommended Operating Conditions
  - DC Characteristics
- Switching Characteristics
  - I/O Timing
  - Internal Logic Timing
  - DCM Timing
  - Configuration and JTAG Timing

Module 4: Spartan-3 FPGA Family: Pinout Descriptions

DS099-4 (v2.5) December 4, 2009

- Pin Descriptions
  - Pin Behavior During Configuration
- Package Overview
- Pinout Tables
  - Footprints

IMPORTANT NOTE: *Each module has its own Revision History at the end. Use the PDF "Bookmarks" for easy navigation in this volume.*

Spartan-3 FPGA Family: Introduction and Ordering Information

DS099-1 (v2.5) December 4, 2009

Product Specification

Introduction

The Spartan™-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates, as shown in Table 1.

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from the Virtex®-II platform technology. These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment.

The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

Features

- Low-cost, high-performance logic solution for high-volume, consumer-oriented applications
  - Densities up to 74,880 logic cells
- SelectIO™ interface signaling
  - Up to 633 I/O pins
  - 622+ Mb/s data transfer rate per I/O
  - 18 single-ended signal standards
  - 8 differential I/O standards including LVDS, RSDS
  - Termination by Digitally Controlled Impedance
  - Signal swing ranging from 1.14V to 3.465V
  - Double Data Rate (DDR) support
  - DDR, DDR2 SDRAM support up to 333 Mbps
- Logic resources
  - Abundant logic cells with shift register capability
  - Wide, fast multiplexers
  - Fast look-ahead carry logic
  - Dedicated 18 x 18 multipliers
  - JTAG logic compatible with IEEE 1149.1/1532
- SelectRAM™ hierarchical memory
  - Up to 1,872 Kbits of total block RAM
  - Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
  - Clock skew elimination
  - Frequency synthesis
  - High resolution phase shifting
- Eight global clock lines and abundant routing
- Fully supported by Xilinx ISE® and WebPACK™ software development systems
- MicroBlaze™ and PicoBlaze™ processor, PCI®, PCI Express® PIPE Endpoint, and other IP cores
- Pb-free packaging options
- Automotive Spartan-3 XA Family variant

Table 1: **Summary of Spartan-3 FPGA Attributes**

Device	System Gates	Equivalent Logic Cells[1]	CLB Array (One CLB = Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50[2]	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200[2]	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400[2]	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000[2]	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	633	300
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	633	300

Notes:

1. Logic Cell = 4-input Look-Up Table (LUT) plus a 'D' flip-flop. "Equivalent Logic Cells" equals "Total CLBs" x 8 Logic Cells/CLB x 1.125 effectiveness.
2. These devices are available in Xilinx Automotive versions as described in DS314, Spartan-3 Automotive XA FPGA Family.

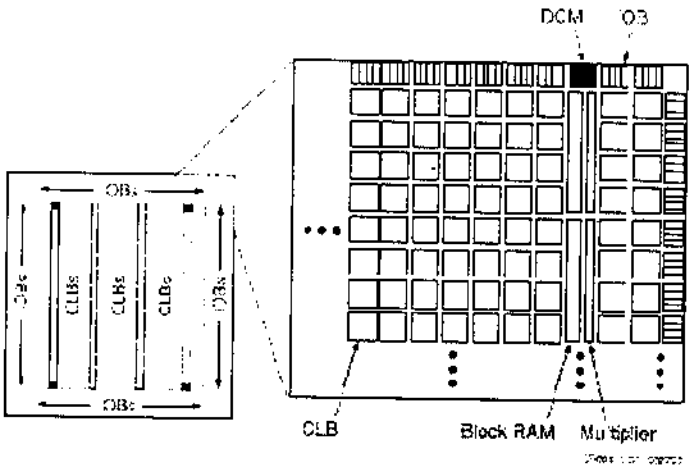
Architectural Overview

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) provide RAM-based Look-Up Tables (LUTs) to implement logic and storage elements (flip-flops or latches). CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
- Input-Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-six different signal standards, including eight high-performance differential standards, are available as shown in [illegible]. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
- Block RAM provides data storage in the form of 18-Kbit dual-port blocks.
- Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

These elements are organized as shown in [illegible]. A ring of IOBs surrounds a regular array of CLBs. The XC3S50 has a single column of block RAM embedded in the array. Those devices ranging from the XC3S200 to the XC3S2000 have two columns of block RAM. The XC3S4000 and XC3S5000 devices have four RAM columns. Each column is made up of several 18-Kbit RAM blocks; each block is associated with a dedicated multiplier. The DCMs are positioned at the ends of the outer block RAM columns.

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.



Notes:

1. The two additional block RAM columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 has only the block RAM column on the left.

Figure 1: **Spartan-3 Family Architecture**

Configuration

Spartan-3 FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. Before powering up, the FPGA configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit wide SelectMAP port.

The recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes the XCF00S PROMs for serial configuration and the higher density XCF00P PROMs for parallel or serial configuration.

I/O Capabilities

The SelectIO feature of Spartan-3 devices supports 16 single-ended standards and 8 differential standards as listed in Table 2. Many standards support the DCI feature, which uses integrated terminations to eliminate unwanted signal reflections.

Table 2: **Signal Standards Supported by the Spartan-3 Family**

Standard Category	Description	V_{CCO} (V)	Class	Symbol (IOSTANDARD)	DCI Option
Single-Ended					
GTL	Gunning Transceiver Logic	N/A	Terminated	GTL	Yes
			Plus	GTLP	Yes
HSTL	High-Speed Transceiver Logic	1.5	I	HSTL_I	Yes
			II	HSTL_II	Yes
		1.8	I	HSTL_I_18	Yes
			II	HSTL_II_18	Yes
			III	HSTL_III_18	Yes
LVCMOS	Low-Voltage CMOS	1.2	N/A	LVCMOS12	No
		1.5	N/A	LVCMOS15	Yes
		1.8	N/A	LVCMOS18	Yes
		2.5	N/A	LVCMOS25	Yes
		3.3	N/A	LVCMOS33	Yes
LVTTL	Low-Voltage Transistor-Transistor Logic	3.3	N/A	LVTTL	No
PCI	Peripheral Component Interconnect	3.0	33 MHz[1]	PCI33_3	No
SSTL	Stub Series Terminated Logic	1.8	N/A (±6.7 mA)	SSTL18_I	Yes
			N/A (±13.4 mA)	SSTL18_II	No
		2.5	I	SSTL2_I	Yes
			II	SSTL2_II	Yes
Differential					
LDT (ULVDS)	Lightning Data Transport (HyperTransport™) Logic	2.5	N/A	LDT_25	No
LVDS	Low-Voltage Differential Signaling		Standard	LVDS_25	Yes
			Bus	BLVDS_25	No
			Extended Mode	LVDSEXT_25	Yes
LVPECL	Low-Voltage Positive Emitter-Coupled Logic	2.5	N/A	LVPECL_25	No
RSDS	Reduced-Swing Differential Signaling	2.5	N/A	RSDS_25	No
HSTL	Differential High-Speed Transceiver Logic	1.8	II	DIFF_HSTL_II_18	Yes
SSTL	Differential Stub Series Terminated Logic	2.5	II	DIFF_SSTL2_II	Yes

Notes:
1. 66 MHz PCI is not supported by the Xilinx IP core although PCI66_3 is an available IO standard.

shows the number of user I/Os as well as the number of differential I/O pairs available for each device-package combination.

Table 3: **Spartan-3 Device I/O Chart**

	Available User I/Os and Differential (Diff) I/O Pairs by Package Type																			
Package	VQG100 VQG100		CP132[1] CPG132		TQ144 TQG144		PQ208 PQG208		FT256 FTG256		FG320 FGG320		FG456 FGG456		FG676 FGG676		FG900 FGG900		FG1156[1] FGG1156	
Footprint (mm)	16 x 16		8 x 8		22 x 22		30.6 x 30.6		17 x 17		19 x 19		23 x 23		27 x 27		31 x 31		35 x 35	
	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff
XC3S50	63	29	83	40	97	43	124	55	-	-	-	-	-	-	-	-	-	-	-	-
XC3S200	63	29	-	-	97	43	140	63	173	76	-	-	-	-	-	-	-	-	-	-
XC3S400	-	-	-	-	97	43	141	62	173	76	221	100	264	116	-	-	-	-	-	-
XC3S1000	-	-	-	-	-	-	-	-	173	76	221	100	333	149	391	175	-	-	-	-
XC3S1500	-	-	-	-	-	-	-	-	-	-	221	100	333	149	487	221	-	-	-	-
XC3S2000	-	-	-	-	-	-	-	-	-	-	-	-	333	149	489	221	565	270	-	-
XC3S4000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	489	221	633	300	712[1]	312[1]
XC3S5000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	489	221	633	300	784[1]	344[1]

Notes:
1. The CP132, CPG132, FGG1156, and PQG208 packages are being discontinued and are not recommended for new designs. See http://www.xilinx.com/support/documentation/spartan-3_customer_notices.htm for the latest updates.
2. All devices listed in a given package column are pin-compatible.
3. User = Single-ended user I/Os. Diff = Differential I/O pairs.

Package Marking

Figure 2 shows the top marking for Spartan-3 FPGAs in the quad-flat packages. Figure 3 shows the top marking for Spartan-3 FPGAs in BGA packages except the 132-ball chip-scale package (CP132 and CPG132). The markings for the BGA packages are nearly identical to those for the quad-flat packages, except that the marking is rotated with respect to the Ball A1 indicator. Figure 4 shows the top marking for Spartan-3 FPGAs in the CP132 and CPG132 packages.

The "-5" and "-4" part combinations may be dual marked as "-50" or "-4" devices. Devices with a single mark are only guaranteed for the marked speed grade and temperature range. Some specifications may exceed the mark rating. Mask revision "E" devices are obsolete. All production devices ship as revision E.

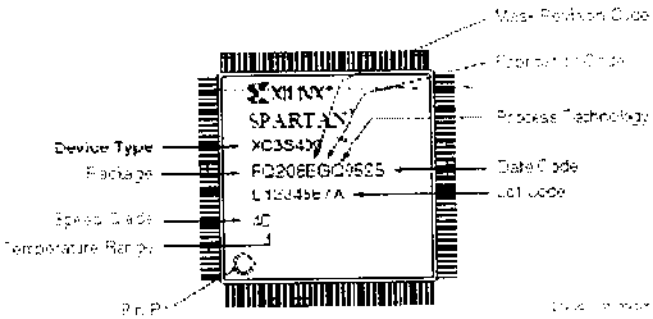


Figure 2: **Spartan-3 QFP Package Marking Example for Part Number XC3S400-4PQ208C**

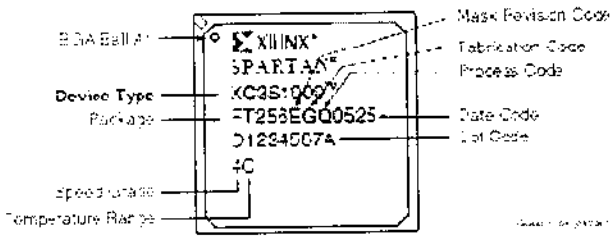


Figure 3: **Spartan-3 BGA Package Marking Example for Part Number XC3S1000-4FT256C**

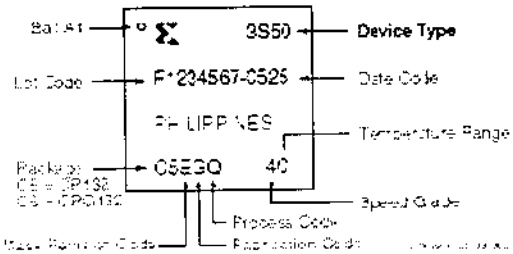
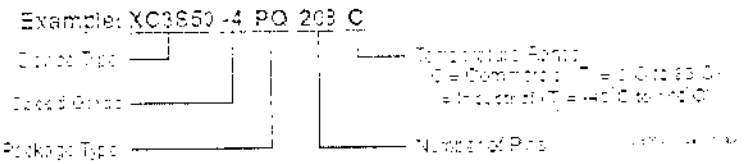


Figure 4: **Spartan-3 CP132 and CPG132 Package Marking Example for XC3S50-4CP132C**

Ordering Information

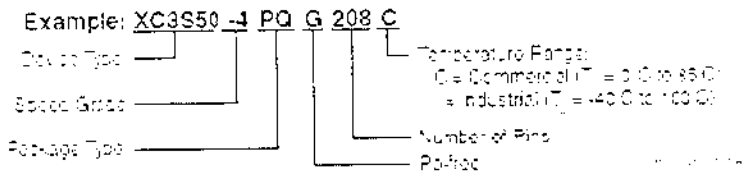
Spartan-3 FPGAs are available in both standard and Pb-free packaging options for all device/package combinations. The Pb-free packages include a special 'G' character in the ordering code.

Standard Packaging



Pb-Free Packaging

For additional information on Pb-free packaging, see XAPP427: "Implementation and Solder Reflow Guidelines for Pb-Free Packages".



Device
XC3S50
XC3S200
XC3S400
XC3S1000
XC3S1500
XC3S2000
XC3S4000
XC3S5000

Speed Grade	
-4	Standard Performance
-5	High Performance [1]

Package Type / Number of Pins	
VQ(G)100	100-pin Very Thin Quad Flat Pack (VQFP)
CP(G)132 [2]	132-pin Chip-Scale Package (CSP)
TQ(G)144	144-pin Thin Quad Flat Pack (TQFP)
PQ(G)208	208-pin Plastic Quad Flat Pack (PQFP)
FT(G)256	256-ball Fine-Pitch Thin Ball Grid Array (FTBGA)
FG(G)320	320-ball Fine-Pitch Ball Grid Array (FBGA)
FG(G)456	456-ball Fine-Pitch Ball Grid Array (FBGA)
FG(G)676	676-ball Fine-Pitch Ball Grid Array (FBGA)
FG(G)900	900-ball Fine-Pitch Ball Grid Array (FBGA)
FG(G)1156 [2]	1156-ball Fine-Pitch Ball Grid Array (FBGA)

Temperature Range (T_J)	
C	Commercial (0°C to 85°C)
I	Industrial (-40°C to 100°C)

Notes:

1. The -5 speed grade is exclusively available in the Commercial temperature range.
2. The CP132, CPG132, FG1156, and FGG1156 packages are being discontinued and are not recommended for new designs. See http://www.xilinx.com/support/documentation/spartan-3_customer_notices.htm for the latest updates.

Revision History

Date	Version No.	Description
04/11/03	1.0	[illegible]
12/04/03	[illegible]	[illegible]
12/04/03	1.0	Added the TQ144 package.
07/13/04	1.3	Added information on Pb-free packaging options.
01/17/05	1.4	Referenced Spartan-3 XA Automotive FPGA family in the [illegible]. Added XC3S50 CP132, XC3S200 FT256, XC3S400 FT256, XC3S400 FG676 options in the [illegible]. Updated Pb-free marking to show mask revision code, manufacturer identity code, and process technology code.
09/10/05	1.5	Added package marking for BGA packages. [illegible] the CP132, CPG132 packages. [illegible]. Added differential complementary single-ended HSTL and SSTL I/O standards.
04/03/06	2.0	Increased number of supported single-ended and differential I/O standards.
04/28/06	2.1	Updated document links.
08/25/07	2.2	Updated [illegible] to allow for dual-marking.
[illegible]/07	2.3	Added XC3S5000 FG676 to [illegible]. Noted that FGG1156 package is being discontinued and updated max I/O count.
08/25/08	2.4	Updated max I/O counts based on FG1156 discontinuation. Clarified dual mark in Package Marking. Updated formatting and links.
12/04/09	2.5	CP132 and CPG132 packages are being discontinued. Added link to Spartan-3 FPGA customer notices. Updated [illegible] with package footprint dimensions.

Design Documentation Available

The functionality of the Spartan-3 FPGA family is described in the following documents. The topics covered in each guide are listed below.

- **UG331: Spartan-3 Generation FPGA User Guide**
  http://www.xilinx.com/support/documentation/user_guides/ug331.pdf
  - Clocking Resources
    - Digital Clock Managers (DCMs)
  - Block RAM
  - Configurable Logic Blocks (CLBs)
  - Distributed RAM
  - SRL16 Shift Registers
  - Carry and Arithmetic Logic
  - I/O Resources
  - Embedded Multiplier Blocks
  - Programmable Interconnect
  - ISE™ Software Design Tools
  - IP Cores
  - Embedded Processing and Control Solutions
  - Pin Types and Package Overview
  - Package Drawings
  - Powering FPGAs
- **UG332: Spartan-3 Generation Configuration User Guide**
  http://www.xilinx.com/support/documentation/user_guides/ug332.pdf
  - Configuration Overview
    - Configuration Pins and Behavior
    - Bitstream Sizes
  - Detailed Descriptions by Mode
    - Master Serial Mode using Xilinx Platform Flash PROM
    - Slave Parallel (SelectMAP) using a Processor
    - Slave Serial using a Processor
    - JTAG Mode
  - ISE IMPACT Programming Examples

For additional Spartan-3 FPGA Starter Kit board hardware, please visit the Spartan-3 FPGA Starter Kit board web page, which includes links to the design examples and the user guide.

- **Spartan-3 FPGA Starter Kit Board Page**
  http://www.xilinx.com/s3sk
- **UG130: Spartan-3 FPGA Starter Kit User Guide**
  http://www.xilinx.com/support/documentation/board_hardware_and_kits/ug130.pdf

Create a Xilinx MySupport user account and sign up to receive automatic e-mail notification whenever the data sheet or the associated user guides are updated.

- **Sign Up for Alerts on Xilinx MySupport**
  http://www.xilinx.com/support/answers/13582.htm

IOBs

For additional information, refer to the 'Using I/O Resources' chapter in UG331.

IOB Overview

The Input/Output Block (IOB) provides a programmable bidirectional interface between an I/O pin and the FPGA's internal logic.

A simplified diagram of the IOB's internal structure appears [illegible]. There are three main signal paths within the IOB: the output path, input path, and 3-state path. Each path has its own pair of storage elements that can act as either registers or latches. For more information, see the 'Storage Element Functions' section. The three main signal paths are as follows:

- The input path carries data from the pad, which is bonded to a package pin, through an optional programmable delay element directly to the I line. There are alternate routes through a pair of storage elements to the IQ1 and IQ2 lines. The IOB outputs I, IQ1, and IQ2 all feed to the FPGA's internal logic. The delay element can be set to ensure a hold time of zero.
- The output path, starting with the O1 and O2 lines, carries data from the FPGA's internal logic through a multiplexer and then a three-state driver to the IOB pad. In addition to this direct path, the multiplexer provides the option to insert a pair of storage elements.
- The 3-state path determines when the output driver is high-impedance. The T1 and T2 lines carry data from the FPGA's internal logic through a multiplexer to the output driver. In addition to this direct path, the multiplexer provides the option to insert a pair of storage elements. When the T1 or T2 lines are asserted High, the output driver is high-impedance (tristate, HiZ). The output driver is active-low enabled.
- A single pulse entering the IOB on any of these paths can be stretched to the storage element to provide an extra clock cycle. Any inverter placed on these paths is automatically absorbed into the IOB.

Storage Element Functions

There are three pairs of storage elements in each IOB, one pair for each of the three paths. It is possible to configure each of these storage elements as an edge-triggered D-type flip-flop (FD) or a level-sensitive latch (LD).

The storage-element-pair on either the Output path or the Three-State path can be used together with a special multiplexer to produce Double-Data-Rate (DDR) transmission. This is accomplished by taking data synchronized to the clock signal's rising edge and converting them to bits synchronized on both the rising and the falling edge. The combination of two registers and a multiplexer is referred to as a Double Data Rate D-type flip-flop (FDDR).

See 'Double-Data-Rate Transmission', page 14 for more information.

The signal paths associated with the storage element are described in Table 4.

Table 4: **Storage Element Signal Description**

Storage Element Signal	Description	Function
D	Data input	Data at this input is stored on the active edge of CK enabled by CE. For latch operation when the input is enabled, data passes directly to the output Q.
Q	Data output	The data on this output reflects the state of the storage element. For operation as a latch in transparent mode, Q will mirror the data at D.
CK	Clock input	A signal's active edge on this input with CE asserted, loads data into the storage element.
CE	Clock Enable input	When asserted, this input enables CK. If not connected, CE defaults to the asserted state.
SR	Set/Reset	Forces storage element into the state specified by the SRHIGH/SRLOW attributes. The SYNC/ASYNC attribute setting determines if the SR input is synchronized to the clock or not.
REV	Reverse	Used together with SR. Forces storage element into the state opposite from what SR does.