University of Alberta

DIGITAL BUILT-IN SELF-TEST OF ANALOG ITERATIVE DECODERS

by

Mimi Wai Mei Yiu

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2006

Your file Votre référence
ISBN: 0-494-13915-3
Our file Notre référence
ISBN: 0-494-13915-3

Dedication

To Dad, Mom, my sisters Amy, Eliza, and Winnie
for their love, encouragement,
understanding, and support.

Abstract

There has been great interest in implementing iterative decoders due to their outstanding performance with bit error rates close to the theoretical limit for error-free transmission known as the Shannon bound. Conventional digital implementations are often complex, demanding significant silicon area and power. Analog circuits for iterative decoding have been proposed and demonstrated by various researchers in recent years. Ensuring high test quality at low cost for these analog-signal designs has become a challenge to test engineers. A design method is presented for testing analog iterative decoders using digital *built-in self-test* (BIST). Mixed signal BIST schemes are often complex and demand larger than acceptable hardware cost. By using a digital BIST scheme, analog iterative decoders can easily be tested in the digital domain. A BIST was designed for an (8, 4) extended Hamming decoder using TSMC 0.18 μm CMOS technology. It is capable of detecting catastrophic faults in the decoder. The decoder has a bit error rate (BER) performance of 0.3 db to 0.8 dB away from maximum likelihood (ML) decoding at speeds of up to 3.7 Mbps. The digital BIST scheme is suitable for any iterative decoder using the sum-product algorithm.

Acknowledgements

I would like to say special thanks to Dr. Vincent Gaudet and Dr. Christian Schlegel for their guidance, support, teaching, funding, and their confidence in me and my work even during hard times.

I would like to thank Dr. Chris Winstead, a mentor and a friend of mine, who had come up with the idea for this thesis. I am deeply indebted to Dr. Winstead for always taking his precious time to help me solving different problems on analog decoding. I would also like to thank Dr. Winstead and Nhan (Dave) Nguyen for designing the BER test interface.

Many thanks to Nhan Nguyen, Tyler Brendon, and John Koob for helping me to solve many problems about analog decoder implementations and design tools applications. Their guidance and enthusiasm helped me to overcome all the periods of frustration and anxiety. Their helpful design tips also saved me a great amount of time on implementing the analog decoder chip.

I would like to extend my thanks to the people in the HCDC lab and VLSI lab for creating a friendly and stimulating working environment, and for many inspiring discussions of communication systems and iterative decoders.

Funding for this project was provided by Canadian Microelectronics Corporation grants 0404CF-ICFAAMY1, NSERC, iCORE, and the Mary Louise Imrie Graduate Student Award.

Table of Contents

List of Tables

List of Figures

List of Symbols

Acronyms

ADC	Analog-to-Digital Converter
ATE	Automatic Test Equipment
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BIST	Built-In Self-Test
BPSK	Binary Phase Shift Keying
CMOS	Complementary Metal Oxide Semiconductor
DAC	Digital-to-Analog Converter
DC	Direct Circuit
DFT	Design for Testability
DRC	Design Rule Checking
DUT	Device Under Test
FEC	Forward Error Correcting
FIFO	First-In First-Out
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
IC	Integrated Circuit
I/O	Input/Output
LDPC	Low Density Parity Check
LFSR	Linear Feedback Shift Register
LLR	Log-Likelihood Ratio
LVS	Logic Versus Schematic
MAP	Maximum a Posteriori
ML	Maximum Likelihood
MUX	Multiplexer
NMOS	N-type Metal Oxide Semiconductor
ORA	Output Response Analyzer
PC	Personal Computer
PCB	Printed Circuit Board
PGA	Pin Grid Array
PMOS	P-type Metal Oxide Semiconductor
RAM	Random-Access Memory
ROM	Read-Only Memory
RTL	Resistor-Transistor Logic
S/H	Sample-and-Hold
SNR	Signal-to-Noise Ratio
TPG	Test Pattern Generator
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VLSI	Very Large-Scale Integrated
XOR	Exclusive OR

Electrical

I_D	drain current
I_U	unit current representing probability 1
Gnd	zero potential, ground
V_{DD}	supply voltage
V_{SS}	zero potential, ground
I_(V_g)	square transistor current when $V_{gs} = 0$
U_T	thermal voltage kT/q, equal to 25.9mV at 300K
V_{ds}	drain-to-source voltage
V_{gd}	gate-to-drain voltage
V_{gs}	gate-to-source voltage
V_{refN}	reference voltage applied to source of NMOS
V_{refP}	reference voltage applied to source of PMOS
V_T	threshold voltage
V_U	unit voltage
W/L	transistor width to length ratio or size ratio

Chapter 1

Introduction

1.1 Motivation

It was shown by Shannon that it is possible to transmit information over a noisy channel with arbitrarily small probability of error, at rates up to the capacity of the channel [43]. Reliable transmission of information over noisy channels requires the use of Forward Error Correcting (FEC) codes where the data is encoded to contain additional redundant information such that errors can be detected and corrected at the receiving site.

Forward Error Correcting codes such as turbo [8] and low density parity check (LDPC) codes [16, 34, 35] can achieve outstanding performance with bit error rates close to the Shannon bound. These types of codes employ the sum-product decoding algorithm [26], a message-passing algorithm that implements probability propagation on a graph. Decoders for these codes exchange probabilistic (soft) information between constituent decoders and iterate several times to improve the decoding reliability.

Since iterative decoding of these high-performance FEC codes requires real number operations and they are computationally demanding, digital implementations can be very complex and challenging to design. Analog implementations of these types of decoders have often been shown to have better performance than digital ones in terms of complexity, silicon area, and power consumption [25, 40, 56]. The sum-product algorithm can be simply implemented in the analog domain using basic transistor circuits, and analog circuits require fewer wire connections between components than fully parallel digital implementations. Analog decoders can save a significant amount of power and silicon area by eliminating the need for analog-to-digital conversion at the receiver front-end. Researchers have found that such analog decoding networks can out perform comparable digital decoders by up to two orders of magnitude in terms of speed when designed for the same power consumption or vice versa [29]. Some research groups have also demonstrated analog Turbo decoders for commercial standards [3, 54]. Moreover, an analog decoder operating at supply voltage of 0.8 V was recently designed to demonstrate the feasibility of low voltage analog decoding [39].

However, increasing the testability of analog iterative decoders has become a major challenge. The testing problem has grown as decoder designs get larger. Modern design and package technologies make external testing even more difficult. For complex analog `designs with a high-volume consumer market, test cost is the bottleneck in reducing overall production cost [13]. Efficient manufacturing of larger decoder chips will depend on their testability. Built-in self-test (BIST), a design-for-testability (DFT) technique in which testing is done using built-in hardware [1], is a feasible way to improve overall testability and facilitate diagnosis and repair of these decoders. Mixed signal BIST schemes usually require significant hardware cost and are often complex. To increase the testability of analog iterative decoders without having complicated test design, a digital BIST scheme is presented. The digital BIST scheme can make testing of analog iterative decoders easier. The BIST scheme is suitable for analog iterative decoders using the sum-product algorithm. An analog decoder using an (8, 4) extended Hamming code is implemented along with the digital BIST scheme. A Hamming code is chosen to be implemented with the digital BIST scheme since it is one of the most widely used linear block codes and it is simple and easy to understand. The digital BIST is capable of detecting catastrophic faults in the Hamming decoder.

1.2 Thesis Outline

This thesis consists of seven chapters. Chapter 2 gives an overview of error correcting codes including background information on communication systems, channel capacity, different types of codes, good examples of codes, and the sum-product algorithm. Chapter 3 presents an overview of analog decoding including previous work, principles, and circuit theory. Chapter 4 gives an overview of built-in self-test, describing its advantages, BIST mechanisms for both digital and analog systems and why digital BIST is beneficial to analog iterative decoders. Chapter 5 presents the digital BIST scheme and how it is implemented in the (8, 4) extended Hamming decoder. Chapter 6 illustrates the test setup and results for the Hamming decoder with BIST. Chapter 7 concludes the thesis and presents recommendations for improvement of the research and future research problems.

Chapter 2

Error Correcting Codes

2.1 Communication Systems

Since the information channel in most communication systems introduces errors during transmission due to noise, error correcting codes are essential to reduce the error rate and improve the reliability of the transmission. Fig. 2.1 shows a typical communication system. The components of a communication system include the source, encoder, modulator, channel, demodulator, decoder, and sink. An information message **u** generated by the source is first encoded into a coded sequence **x** by adding redundant information for error detection or correction. The modulator then performs modulation on the coded information to facilitate transmission over the channel and the demodulator undoes the process at the receiving side. The channel could be a wire, coaxial cable, waveguide, an optical fiber, a radio link, or even a memory. The signal going through the channel is distorted by undesirable signals called noise. Finally, the demodulated noisy signal **y** is decoded by estimating the most likely sent message **û** according to **y** and received by the sink.

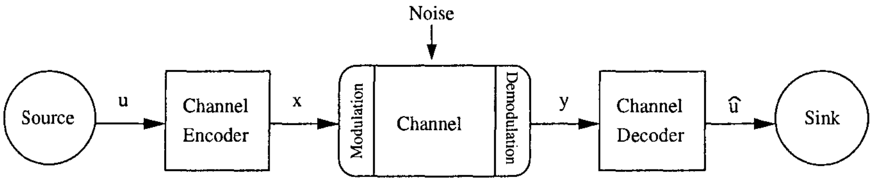


Figure 2.1 Communication system

2.2 Algebraic and Probabilistic Coding

Error Correcting Codes are divided into two categories: *algebraic coding* and *probabilistic coding* [30]. Algebraic coding performs *hard decisions* at the decoding stage, which is useful to provide strong protection against low noise levels. In algebraic decoding, decoder input signals with noise are first digitized into hard bits (1 or 0) by a comparator before doing any error detection or correction.

Probabilistic coding applies *soft decisions* at decoding stage and is suitable for providing moderate protection against high noise levels. In probabilistic decoding, the distorted input signal at the decoder is translated into probability information or an alternate form called the *log-likelihood ratio* (LLR) before decoding takes place. The LLR ratio can be computed by using the equation:

$$LLR = \log\frac{p(y|x=1)}{p(y|x=0)}, \qquad (2.1)$$

where **y** is the received symbol and **x** is the transmitted symbol. The conversion to hard bit decisions is done after the decoding is completed.

2.3 Types of Codes

There are two main types of codes that are commonly used: block codes and convolutional codes.

2.3.1 Block Codes

In block codes, a k-bit block of data is collected and then encoded into a codeword of n-bits ($n > k$). For each k-bit data, there is a distinct codeword of n-bits. So, corresponding to the 2^k different possible messages, there are 2^k different possible codewords. When k data bits are sent by a codeword of n bits, the *code rate* is

$$R = \frac{k}{n}. \tag{2.2}$$

Such a code is called a (*n*, *k*) code. For encoding, codeword **x** is generated by multiplying the information vector **u** by a *generator matrix* **G**:

$$\mathbf{x} = \mathbf{u} \cdot \mathbf{G}. \tag{2.3}$$

Decoding the codeword requires a *parity-check matrix* **H** which is orthogonal to the generator matrix:

$$\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}. \tag{2.4}$$

The received word **r** is multiplied by the transpose of the parity-check matrix and the result **s**, known as *syndrome*, is the binary representation of where the errors occurred:

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T. \tag{2.5}$$

If all elements of **s** are zero, the received word is a valid codeword. If **s** contains non-zero elements and the received word contains only one error, the bit in error can be determined by analyzing which parity checks have failed.

The coding and decoding processes for block codes are *memoryless*. Encoding and decoding of a block of data depends only on that current block and not on any other block.

2.3.2 Convolutional Codes

Convolutional codes introduce memory into the error-correcting process. The encoder takes *k*-bit blocks of information sequence **u** and produces a code sequence **v** of *n* symbol blocks. Each encoded block depends not only on the corresponding *k*-bit information block but also on *m* previous information blocks. Therefore, the encoder has a memory order of *m*. The code rate *R* is also defined as *k/n*. The coding process of convolutional codes is done using a finite state machine consisting of shift registers and sum modulo 2 units (XOR gate). Generator matrices for convolutional codes are described by polynomials. Fig. 2.2

shows an example of a feed-forward convolutional encoder with k = 2, n = 4, and m = 2 and its generator matrix is

$$G = \begin{bmatrix} 1 & 0 & D & 1 \\ 0 & 1 & 1 & D \end{bmatrix}, \tag{2.6}$$

where D represents a unit delay. Decoding of convolutional codes can be done by using the *Viterbi algorithm*, a decoding algorithm proposed by Andrew Viterbi [14, 50]. It is a very general algorithm used to deduce the sequence of states of a finite state machine (FSM) that generated a given output sequence.

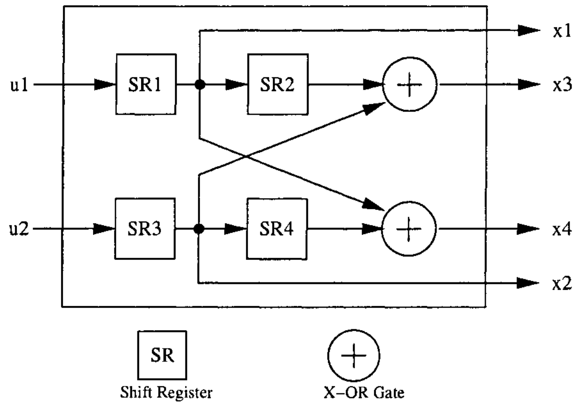


Figure 2.2 Convolutional encoder with k = 2, n = 4, and m = 2

2.4 Channel Capacity

Channel capacity represents the amount of information that can be sent through a channel. The additive white Gaussian noise (AWGN) channel is the most fundamental form of all communication channel models. The Gaussian Channel is shown in Fig. 2.3 where **Y** is the channel output, **X** is the transmitted information, and **N** is a zero-mean Gaussian random variable with unit variance $\sigma_N^2 = N_0/2$ where N_0 is the spectral noise power in the channel. The output of the channel is given by

$$\mathbf{Y} = \mathbf{X} + \mathbf{N}. \tag{2.7}$$

The channel capacity of a band-limited AWGN channel is given by

$$C = W \log\left(1+\frac{S}{N}\right), \tag{2.8}$$

where W is the channel bandwidth, S is the average transmitted power, and N is the average noise power. The term S/N is also called the *signal-to-noise ratio* or SNR. According to Shannon's Channel Coding Theorem, it is possible to design a code that permits error-free transmission across the channel at a rate R, provided that $R < C$.

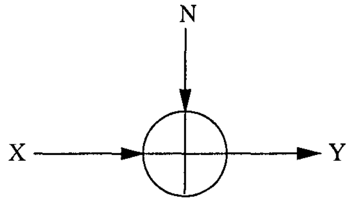


Figure 2.3 Additive white Gaussian noise channel

2.5 Hamming and Minimum Distances of a Code

The Hamming distance is defined as the number of bits which differ between two codewords and the minimum distance is the smallest Hamming distance over all pairs of distinct codewords. The minimum distance is a very important indication of the error robustness of a code. For a linear block code with a minimum distance of d_{min}, the number of errors t that can be corrected is

$$t = (d_{min} - 1)/2, \tag{2.9}$$

or the number of errors t that can be detected is

$$t = d_{min} - 1, \tag{2.10}$$

but not both simultaneously (assuming hard decisions). The greater the minimum distance of a code, the bigger the differences between each pair of codewords, therefore, the greater the chance to correctly decode the transmitted word.

2.6 Linear Codes and Systematic codes

A block code **C** is *linear* if, for codewords **a** and **b** in **C**, **a** + **b** is also in **C**. Therefore, the sum of two codewords is also a codeword for a linear block code. A *systematic* code is a special case of linear block codes. For a codeword composed of n bits $x_1, x_2, \ldots, x_n$, and a data word composed of k bits $u_1, u_2, \ldots, u_k$, if $x_1 = u_1, x_2 = u_2, \ldots, x_k = u_k$ and the remaining bits from x_{k+1} to x_n are linear combinations of u_1 to u_k, the code is known as a systematic code. Thus in a systematic code, the first k bits of a codeword are the original data bits and the last n - k bits are the parity bits.

2.7 Hamming Codes

Hamming Codes were discovered by Richard Hamming in 1949 [23, 43]. They are systematic linear block codes with codewords that are 2^m - 1 bits long ($m \geq 3$), having 2^m - m - 1 information bits, m parity bits and a minimum distance of 3. They can correct any single error or detect any double error.

The codewords of Hamming codes consist of the original information bits and the calculated parity bits. For a (n, k) Hamming code, the size of the generator matrix is k x n and it can be partitioned into a k x k identity matrix $\mathbf{I}_k$ and a k x (n - k) parity generation matrix **P**. The columns of **P** are selected so that each column is unique:

$$\mathbf{G} = [\ \mathbf{I}_k \mid \mathbf{P}\ ]. \tag{2.11}$$

The parity check matrix is an (n - k) x n matrix in the form:

$$\mathbf{H} = [\ \mathbf{P}^T \mid \mathbf{I}_{n-k}\ ]. \tag{2.12}$$

Hamming codes can be extended by adding an extra column of parity bits into the **G** matrix such that the code can detect up to 3 errors but can still only correct a single error. The minimum distance is then changed from 3 to 4:

$$\mathbf{G}_{ext} = [\ \mathbf{G} \mid \mathbf{p}\ ]. \tag{2.13}$$

Each extra bit in **p** is added to maintain an even number of 1s in each row. If there is an even number of 1s in the row, the extra bit becomes 0, otherwise it becomes 1. The parity check matrix for the extended Hamming code is constructed by adding a row of all 0s to the **H** matrix and then a column of 1s to the **H** matrix.

2.8 Low Density Parity Check (LDPC) Codes

Low density parity check (LDPC) codes were first discovered by Gallager in 1962 [16]. However, the coding community largely ignored them since early iterative algorithms used hard decisions and LDPC codes require high complexity computation. These codes had little value until the discovery of soft-information iterative algorithms. In 1996, Mackay *et al.* rediscovered LDPC codes and showed that they have near-Shannon limit performance [34, 35]. The rediscovery was prompted by the invention of Turbo Codes in 1993 by Berrou, Glavieux, and Thitimajshima [7].

LDPC codes are specified by very sparse random parity check matrices and these codes possess promising distance properties. LDPC codes are linear block codes that can be defined in terms of a $(n - k)$ x n parity check matrix **H**. For regular LDPC codes, the **H** matrix has a small fixed number j of 1s per column ($j > 2$) and a small fixed number k of 1s per row ($k > j$). Irregular LDPC codes, which have non-constant number of 1s in each column and row in the H matrix, can perform better than regular LDPC codes in the waterfall region[1]. To obtain good bit error rate (BER) performance, large block lengths are required.

2.9 Trellis Codes

Codes that can be defined by *trellis* diagrams are called Trellis codes. Trellis diagrams were first introduced in Forney's paper on the Viterbi algorithm in 1973 [14]. A trellis diagram consists of states and labeled edges such that every path in

[1] The waterfall region is the segment of the BER curve with the steepest descent.

the diagram represents a codeword or a code sequence. The encoding of any trellis code is done by a finite-state machine (FSM). The trellis diagram can plot the succession of states of the FSM over time.

A trellis representation can be used for both block codes and convolutional codes. Unlike block codes, convolutional codes can have infinitely long codewords and the encoding process can continue indefinitely. However, we can add a sequence of zeros at the encoder input after a certain number of information bits to drive the FSM back to the initial state to terminate the code. Such termination turns a convolution code into a block code. The zeros that are added for termination are called *termination* bits *L*.

For example, Fig. 2.4 shows a simple convolutional encoder and the encoder can be described by a FSM with four states. The state transition diagram of the FSM is shown in Fig. 2.5. The trellis diagram of such a convolutional encoder is presented in Fig. 2.6. A trellis diagram is a time unrolled version of the state transition diagram. In this example, two termination bits are added to terminate the code. In the trellis diagram, time is represented horizontally from left to right and the diagonal line segments represent the transitions.

2.10 Turbo Codes

Berrou, Glavieux, and Thitimajshima introduced Turbo codes, also known as Parallel Concatenated Convolutional Codes (PCCC), in 1993 [7]. The discovery of these codes was a critical event in the error control coding discipline since these can achieve Shannon's bound to within 0.5 dB of SNR [43], outperforming all other coding schemes.

An example Turbo encoder is presented in Fig. 2.7. Turbo codes use two parallel-concatenated convolutional encoders connected by an *interleaver* to generate codes. The interleaver is used to permute the input bits so that the two encoders operate on different input sequences.

An example Turbo decoder is presented in Fig. 2.8. Decoding of Turbo Codes can be achieved by using two soft-input/soft-output (SISO) decoders using the *Maximum a posteriori* (MAP) algorithm [6], an interleaver, and a deinterleaver. During decoding, the result calculated from decoder 1 is sent to decoder 2, and result coming from decoder 2 is feed back to decoder 1. This process is *iterated* many times until reliable bit decisions are reached.

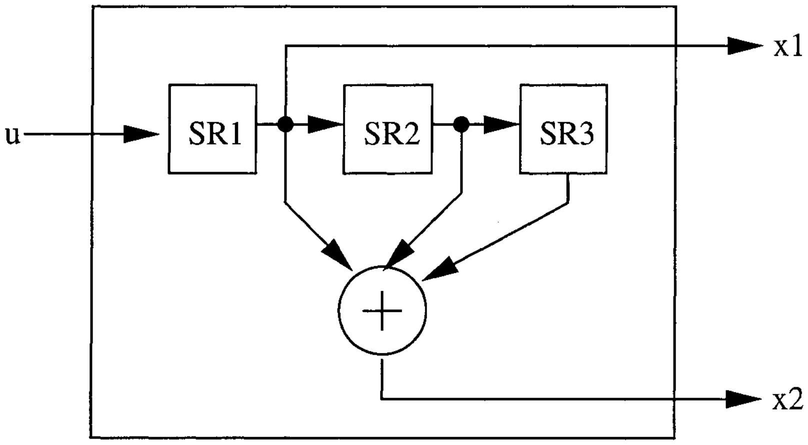


Figure 2.4 A simple convolutional encoder

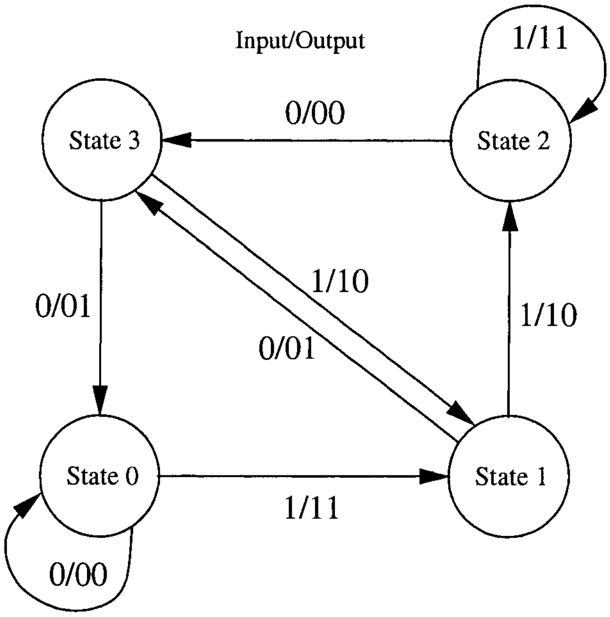


Figure 2.5 State transition diagram of the FSM describing the encoder in Fig. 2.4

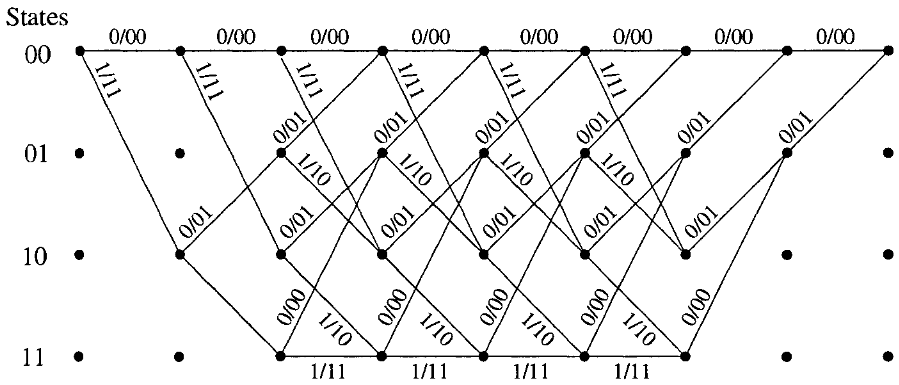


Figure 2.6 Trellis diagram of the convolutional encoder in Fig. 2.4

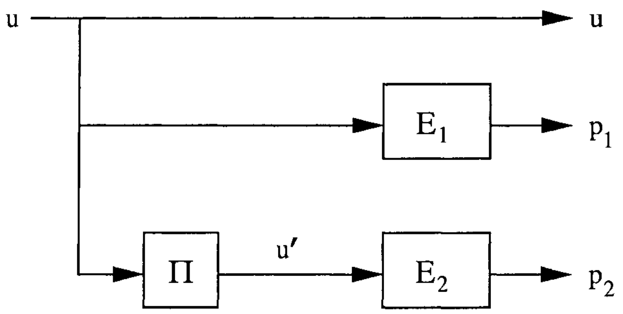


Figure 2.7 Turbo Encoder containing two convolutional encoders E1 and E2, and one interleaver Π

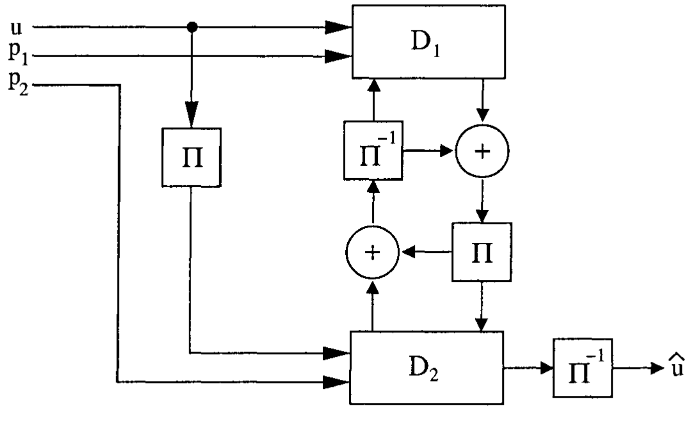


Figure 2.8 Turbo Decoder containing two MAP decoders, an interleaver Π, and a deinterleaver Π^{-1}

2.11 The Sum-Product Algorithm

A factor graph represents how complex functions, dependent on many variables, can factor into a product of simpler functions, each of which depends on a subset of the variables [15, 26]. Factor graphs consist of nodes and bi-directional edges which describe the network of connections among variables. For the majority of codes, factor graphs are bipartite since these graphs are composed of only two types of nodes, and there are only connections between nodes of different types. This type of graph can be used to provide a graphical representation of the parity check matrix.

The sum-product algorithm is an important decoding algorithm in error-control coding that operates by *message passing* on a factor graph. Local probability functions associated with the global function are computed at each node and the results, known as messages, are passed on to the connecting nodes. This process is also referred to as *probability propagation* since the algorithm deals with messages that are probabilities. The algorithm is named 'sum-product algorithm' because its computation rules consist of additions and multiplications only. The decoding procedure of the sum-product algorithm describes decoders for many codes such as trellis codes, LDPC codes, block product codes, and turbo codes. Decoding on factor graphs using the sum-product algorithm is optimal if the graph is cycle-free. If the graph is not cycle-free, we can increase the block length of the code to a sufficiently large number to obtain a performance close to the optimal decoder.

Fig. 2.9 shows a factor graph representing a regular LDPC code with the following H matrix:

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

The factor graph contains equality nodes at the top and check nodes at the bottom and edges connecting the two types of nodes. Edges between nodes are *bidirectional.* Each equality node and check node represents a column and a row in **H,** respectively. When h_{ij} in **H** is 1, an edge exists between the i^{th} check node and j^{th} equality node. In the factor graph, the number of edges connected to a node is called the *degree* of the node. Since there are two 1s in each column of **H** and four 1s in each row, each equality node has a degree of 2 and each check node has a degree of 4.

For most factor graphs, we only need to describe the operations in nodes with three edges. Any nodes with a degree of more than three can be broken down into a cascade of three-edge blocks as shown in Fig. 2.10. This notion will be used later on in our implementation.

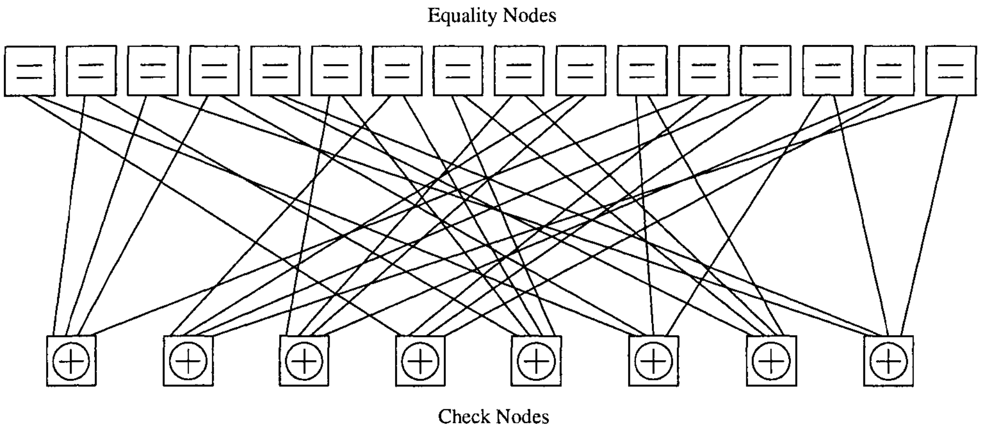


Figure 2.9 Factor graph of a regular LDPC code

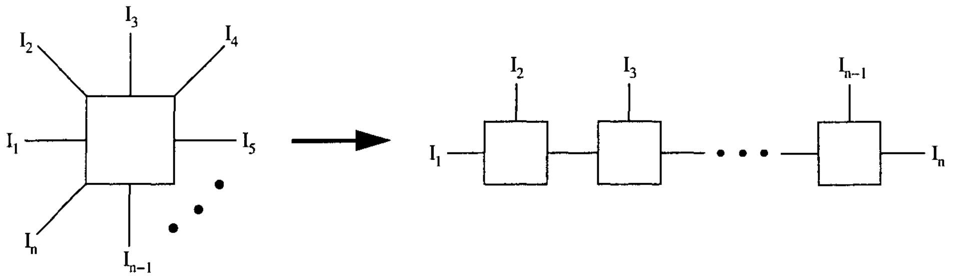


Figure 2.10 n-edge factor graph node constructed by a cascade of three-edge nodes

For a three-edge node with inputs X and Y and output Z, given that a probability distribution for X is $p_X = \{p_X(0),\ p_X(1)\}$, where $p_X(0)$ is the probability that X equals 0 and $p_X(1)$ is the probability that X equals 1, and for Y is $p_Y = \{p_Y(0), p_Y(1)\}$, the probability function for an equality node is

$$\begin{bmatrix} p_Z(0) \\ p_Z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_X(0)\, p_Y(0) \\ p_X(1)\, p_Y(1) \end{bmatrix}. \tag{2.14}$$

Here γ is a normalization factor to ensure that $p_Z(0) + p_Z(1) = 1$. The probability function for a check node is

$$\begin{bmatrix} p_Z(0) \\ p_Z(1) \end{bmatrix} = \begin{bmatrix} p_X(0)\, p_Y(0) \ + \ p_X(1)\, p_Y(1) \\ p_X(0)\, p_Y(1) \ + \ p_X(1)\, p_Y(0) \end{bmatrix}. \tag{2.15}$$

When decoding on factor graphs using the sum product algorithm, we have to follow the extrinsic information rule, which states that the message sent from a node *n* on an edge *e* is a function of all messages received at node *n*, except the message received on edge *e* [26]. Let us assume messages are expressed as LLRs and r_k denotes the received symbol at equality node *k* from an AWGN channel. The decoding sequence for the sum-product algorithm is then as follows [41]:

Step 1. Initialize messages as $\frac{2}{\sigma_N{}^2} r_k$ for each equality node, where σ_N is the noise variance and $\sigma_N{}^2 = N_0/2$. N_0 is the spectral noise power of the AWGN noise in the channel.

Step 2. Pass initial messages from equality nodes to check nodes.

Step 3. Calculate check node probability information and send the messages from check nodes to equality nodes with which they are connected.

Step 4. Calculate equality node probability information and send the messages from equality nodes to the check nodes with which they are connected.

Step5. Repeat step 3 and 4 until a fixed number of iterations have been completed or the estimated codeword $\hat{\mathbf{u}}$ satisfies the constraint $\hat{\mathbf{u}} \cdot \mathbf{H}^T = 0$.

2.12 Chapter Summary

In this chapter, a brief review of error correcting codes was presented. We have explained how a communication system works, how codes are divided into different categories, some terms and definitions of coding theory, and the sum-product decoding algorithm. The analog decoder implemented in this thesis is based on the concepts and decoding algorithm discussed in this chapter.

Chapter 3

Analog Iterative Decoding Circuits

3.1 Analog Implementation of Iterative Decoders

In Chapter 2, we covered the sum-product algorithm used for iterative decoding. The algorithm works with probability information, and messages are passed between nodes on factor graphs representing a specific code. We called this process 'probability propagation'. Since the algorithm deals with probability distributions, its implementation requires real-number arithmetic and digital implementations of such arithmetic are often complex. In 1998, Hagenauer [21, 22] and Loeliger *et al.* [29, 30] independently proposed to decode error-correcting codes by analog circuits. The work by both Hagenauer and by Loeliger *et al.* was inspired by "turbo"-style decoding of codes described by graphs [15, 52, 53]. The analog circuits are used to implement the sum and product functions of iterative decoding algorithms in a simple way. With these circuits, decoders using the sum-product algorithm can be directly implemented in analog VLSI. The main advantage of analog decoders is that the decoder runs in continuous time until it reaches an equilibrium state that corresponds to the closest transmitted codeword. Lustenberger *et al.* built the first working analog decoder using discrete transistors [31]. Lustenberger *et al.* then went on to produce larger designs, most notably a decoder chip [33] announced in 1999. Hagenauer *et al.* implemented an actual decoder one year later [37]. Winstead *et al.* [55] built the first complementary metal-oxide semiconductor (CMOS) decoder. Moreover, Nguyen *et al.* [40] implemented a low voltage analog decoder, and Gaudet and Gulak [17] designed a programmable interleaver for analog Turbo decoders. Other working decoder ICs have also recently been reported [3, 24, 51].

3.2 Sum-Product Module

The sum-product module shown in Fig. 3.1(a) is the basic building block for factor graph-based probability propagation networks. It takes two probability mass functions $p_X(x)$ and $p_Y(y)$ then computes the third probability mass function $p_Z(z)$ according to the following equation [29]:

$$p_Z(z) = \gamma \sum_{x\in X}\sum_{y\in Y} p_X(x)\, p_Y(y)\, f(x,y,z)\,, \tag{3.1}$$

where f is a function with codomain {0, 1} expressing the relationship between random variables x, y, and z. The scaling factor γ is added to the equation to ensure probabilities add up to 1. We can simply illustrate the {0, 1}-valued functions f by trellis diagrams as shown in Fig. 3.2(a) and Fig. 3.2(b). Nodes on the left correspond to the elements in **X** ($x\in$ **X**), nodes on the right correspond to the elements in Z ($z\in$ **Z**), and $f(x, y, z) = 1$ if and only if an edge between $x\in$ **X** and $z\in$ **Z** labeled $y \in$ **Y** exists.

The sum-product module can be decomposed into three sub-modules depicted in Fig. 3.1(b). The three sub-modules are a normalization block, a sum block $\sum$, and a product block $\prod$. In a circuit realization, probabilities are represented by currents. Given a probability distribution $p_X = \{p_X(0),\ p_X(1)\}$, the probability currents are defined as follow:

$$I_{X0} = I_U\ p_X(0)\,, \tag{3.2}$$

$$I_{X1} = I_U\ p_X(1)\,. \tag{3.3}$$

where I_U is a unit current representing a probability of 1. The probability $p_X(0)$ is equal to I_{X0}/I_U and $p_X(1)$ is equal to I_{X1}/I_U. Summation of currents is done by connecting wires together, and normalization and multiplication of currents are realized by using Gilbert vector normalizers and Gilbert vector multipliers, respectively, as will be explained in the next sections.

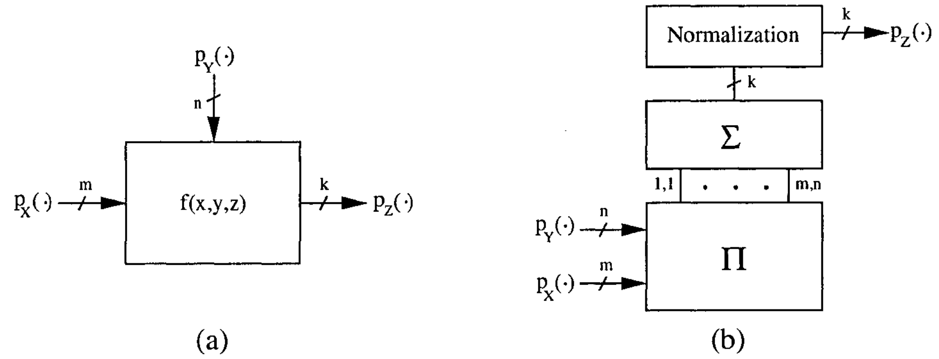


Figure 3.1 Sum-product module: (a) top-level block diagram [29] and (b) module divided into smaller blocks

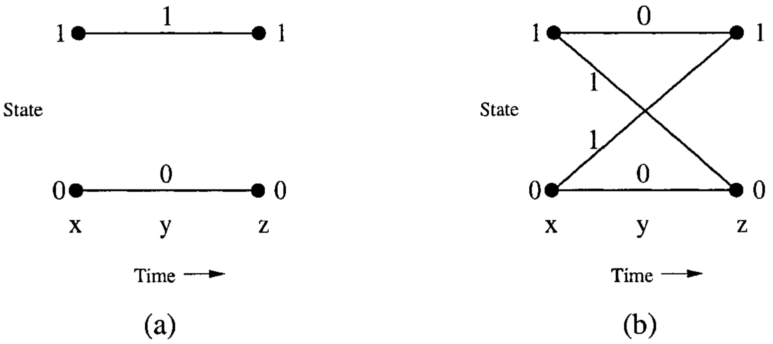


Figure 3.2 Trellis diagram for f(x, y, z) = 1 iff: (a) x ⊕ y = z and (b) x = y = z [29]

3.2.1 Current Multiplication

Current multiplications can be performed conveniently by Gilbert vector multipliers operating in the subthreshold region [18]. The multiplier is a standard circuit for real number multiplications. Fig. 3.3 presents the Gilbert vector multiplier for the product block ∏. The output currents of the multiplier are easily derived using the translinear principle in [19]:

$$I_{zij} = \frac{I_{xi}\, I_{yj}}{\sum_{j=1}^{n} I_{yj}}. \tag{3.2}$$

In order to perform multiplication, all the transistors in the Gilbert multiplier should operate in the subthreshold region. In the subthreshold region, each transistor has to satisfy the condition $v_{gs} < V_T$, where v_{gs} refers to the voltage between the gate and source of a MOS transistor and V_T refers to the threshold voltage. The drain current is [57]:

$$I_D = I_0 \frac{W}{L} \exp\left(\frac{\kappa \cdot v_{gs}}{U_T}\right)\left[1 - \exp\left(\frac{-\kappa \cdot v_{ds}}{U_T}\right)\right], \tag{3.3}$$

where I_0 is a technology constant with units of amperes, W and L are the transistor's width and length, κ is a technology constant $\approx$ 0.7, and U_T is the temperature dependent thermal voltage $\approx$ 25 mV at room temperature. Moreover, we want to keep v_{ds} sufficiently large so that it has little effect on I_D in equation (3.3). The MOS transistor is said to be in saturation when v_{ds} is large enough to be neglected and we can ensure this condition by setting the reference voltage V_{refN} in Fig. 3.3 to be about 0.3 V or greater. In equation (3.3), the drain current is then an exponential function of the gate to source voltage.

3.2.2 Current Summation

In order to complete the computation of (3.1), we need to sum the currents $I_{i,j}$ for each $z \in \mathbf{Z}$ for which $f(x_i, y_j, z) = 1$. As mentioned in the beginning of Section 3.2, summation of currents can easily be accomplished by tying wires together and relying on Kirchhoff's current law.

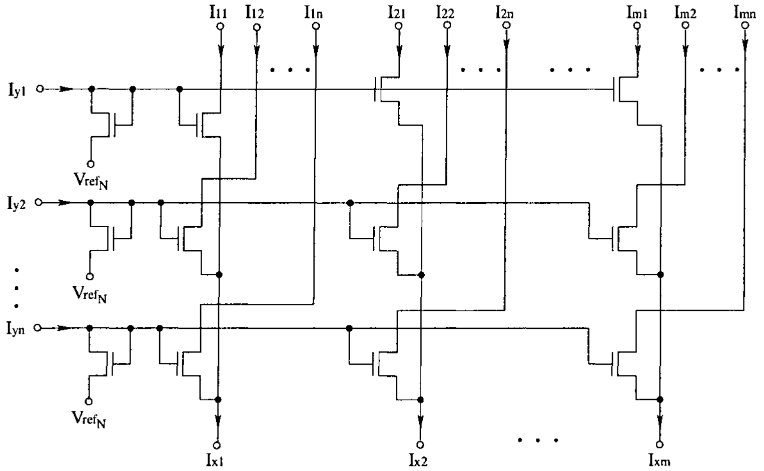


Figure 3.3 Gilbert vector multiplier for two input vectors of length m and n

3.2.3 Normalization

In the sum-product circuits, attenuation of outputs happens due to the current loss of unused product terms. This is because in the sum-product algorithm, the product of two small real numbers below unity tends towards zero. If there are many stages of sum-product computations, the attenuation may become severe. This issue can be resolved by scaling up the current outputs and making the sum of all outputs from the Gilbert multiplier equal to I_U, the unit current representing a probability of 1. In [20], Gilbert has presented a vector scaling circuit that performs the exact normalization function we needed to resolve the attenuation problem. The circuit implements the following function:

$$I_{out,j} = I_u \frac{I_{in,j}}{\sum_{l=1}^{L} I_{in,l}}. \tag{3.4}$$

The normalization of the outputs does not affect the correctness of the algorithm. Fig. 3.4 presents the Gilbert normalizer circuit for a two-element current vector along with the sum-product circuit.

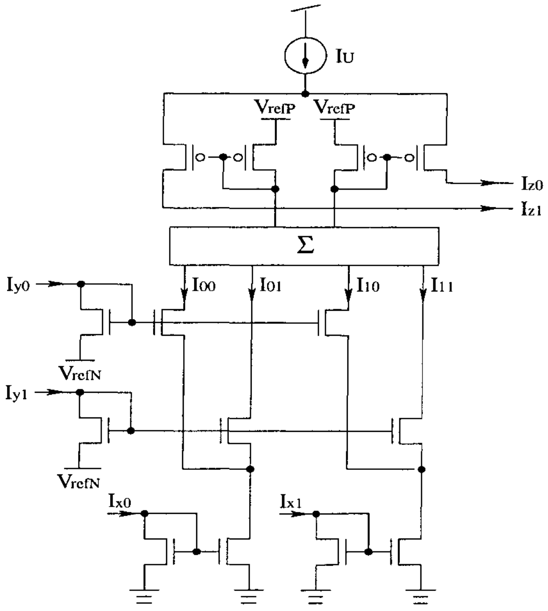


Figure 3.4 General sum-product circuit with Gilbert normalizer

3.3 Factor Graph Nodes

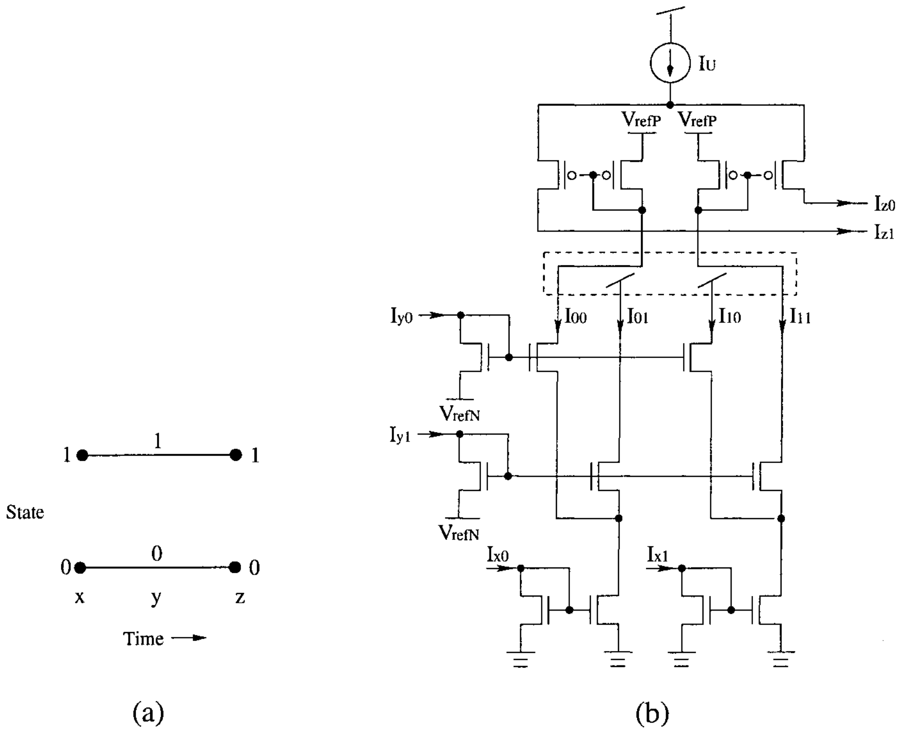
Factor graphs consist of equality and check nodes. They are implemented using the sum-product module and with its circuit shown in Fig. 3.4. We now describe the specific implementations of equality and check nodes.

3.3.1 Equality Nodes

For an equality node, the function $f(x, y, z)$ in equation (3.1) is equal to 1 if $x = y = z$, or else, f is equal to 0. The corresponding trellis diagram is shown in Fig. 3.5(a). The output currents of an equality node are calculated by:

$$\begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} = \gamma \begin{bmatrix} I_{x0} I_{y0} \\ I_{x1} I_{Y1} \end{bmatrix}, \qquad (3.5)$$

where γ is a constant that depends on the bias current in the normalizer. Fig. 3.5(b) illustrates the sum-product circuit for a *unidirectional* equality node.



(a) (b)

Figure 3.5 Equality node: (a) Trellis representation (b) One-direction circuit in transistor level

3.3.2 Check Nodes

For a check node, the function *f*(*x*, *y*, *z*) in equation (3.1) is equal to 1 if $x \oplus y = z$, where ⊕ denotes mod-2 addition, or else, *f* is equal to 0. The corresponding trellis diagram is shown in Fig. 3.6(a). The sum-product module for a check node becomes an XOR gate. The output currents of a check node are calculated by:

$$\begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} = \gamma \begin{bmatrix} I_{x0}I_{y0} + I_{x1}I_{y1} \\ I_{x0}I_{Y1} + I_{x1}I_{y0} \end{bmatrix}. \tag{3.6}$$

Fig. 3.6(b) illustrates the sum-product circuit for a unidirectional check node.

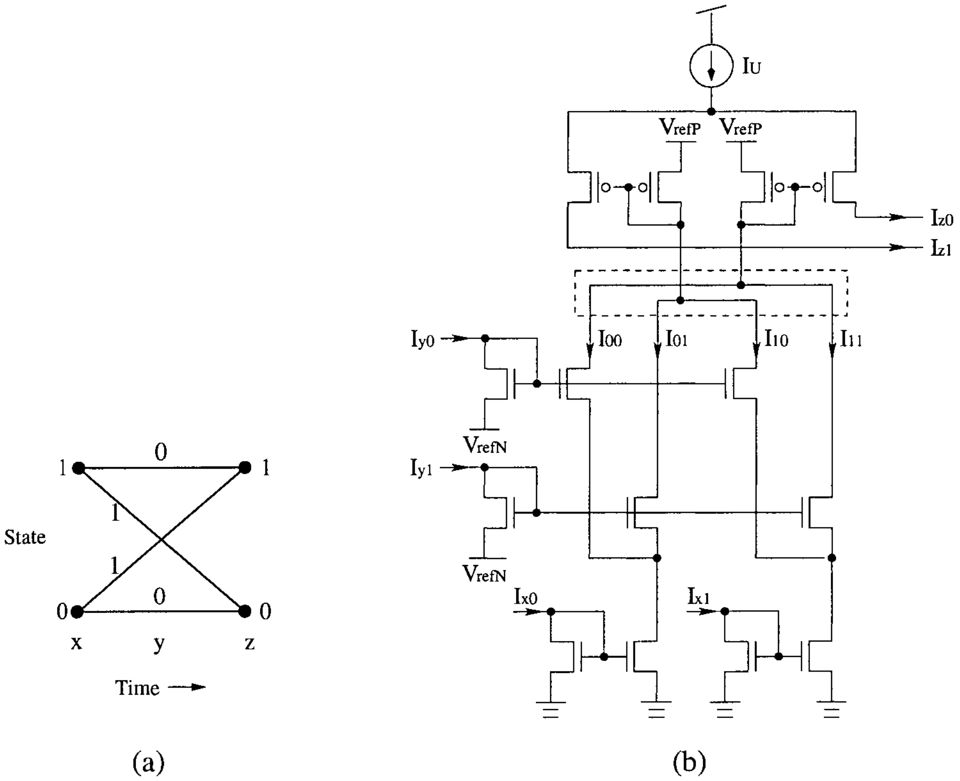


Figure 3.6 Check node: (a) Trellis representation (b) One-direction circuit in transistor level

3.3.3 Bi-directional Nodes

In a factor graph, all edges are bi-directional since messages are passed back and forth between the equality nodes and check nodes. Construction of three-edge bi-directional nodes requires three instances of unidirectional sum-product modules connected to each other as illustrated in Fig. 3.7.

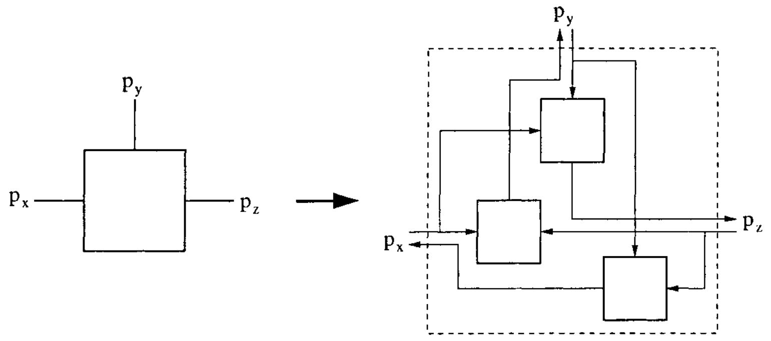


Figure 3.7 Three-edge bi-directional node constructed by three unidirectional sum-product modules

3.4 LLR Ratio to Probability Distribution

Before performing decoding on a factor graph using the sum-product algorithm, the decoder has analog voltage inputs representing LLRs. We can convert the LLR voltages into probability currents by using a differential pair circuit shown in Fig. 3.8. The probability currents of this circuit are calculated by [62]:

$$I_{X0} = \alpha e^{(Vin-Vs)}, \tag{3.7}$$

$$I_{X1} = \alpha e^{(Vref-Vs)}, \tag{3.8}$$

where α is a current constant. The relationship between the input voltages and the output currents can be expressed as:

$$K\,\mathrm{LLR}(X) = V_{in} - V_{ref} = K\ln\left(\frac{I_{X1}}{I_{X0}}\right), \tag{3.9}$$

where K is a voltage constant.

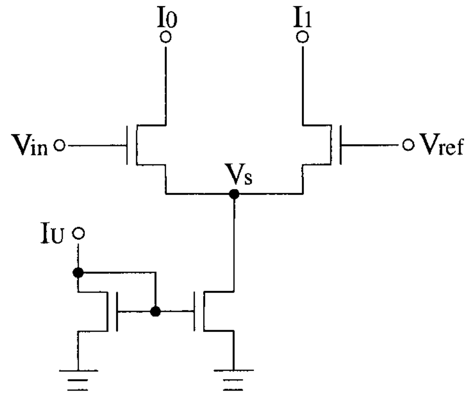


Figure 3.8 Differential Pair for LLR voltages to probability currents conversion

3.5 Chapter Summary

In this chapter, we have discussed previous work on analog decoding and the implementation of analog iterative decoders using the sum-product algorithm based on Gilbert multipliers. The circuits for different parts of a decoder were illustrated including the normalization circuit, sum-product circuit, and the LLR to probability conversion circuit. These circuits are the fundamental circuits used to build the Hamming decoder in this thesis. In the next chapter, we will introduce the topic of built-in self-test. Background information and a BIST architecture are provided.

Chapter 4

Built-In Self-Test

4.1 VLSI Testing

Testing of integrated circuits (ICs) is an inevitable process to ensure high quality of functionality in commercial products. *Verification testing* is performed during the early design phase to ensure the correctness of the design. When the verification is successful, it indicates the beginning of production, which implies large-scale manufacturing. During mass production of the design, fabricated chips are tested in the factory and this is called *manufacturing testing*.

Automatic test equipment (ATE) is used to perform manufacturing testing to guarantee the performance of each device after it is fabricated [9]. Test programs are written to control the test procedures of such testers. The test program directs the ATE to generate test patterns (such as digital signals and sine waves) to apply to the *device under test* (DUT). The ATE tester then analyzes the response coming out of the DUT and determines whether the device is good or bad. The work of a general ATE tester is illustrated in Fig. 4.1.

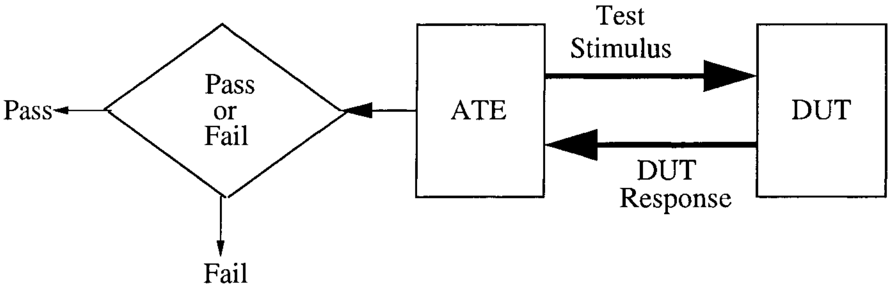


Figure 4.1 Duty of a general ATE tester

However, as today's ICs become faster and more complex, the testing difficulty increases rapidly. Since the number of input/output (I/O) pins fails to grow as quickly as the logic content, physical access to internal circuitry becomes very difficult. Testing program generation for ATEs becomes much more complex and time consuming. It is challenging to achieve a high fault coverage, making it hard to detect all faults in the device. Testers for high-speed and complex ICs can cost millions of dollars. In the near future, it may cost as much to test the silicon as it costs to manufacture it [5]. Therefore, in order to achieve efficient and economical manufacturing, one must consider the need for production test capabilities during the early design phase. Such methodology is called *design for testability* (DFT). Adding these capabilities to ensure the testability of a device at the design phase can make the most savings since testing cost increases in every stage. If defects are detected at a later stage in the manufacturing process, the cost will be higher to repair/remove the faulty items.

4.2 Design for Testability

Design for testability (DFT) is a set of rules and strategies applied during the design phase of a device, in which test structures are inserted, resulting in a more easily or thoroughly testable product [8]. The added test structures do not provide any additional functionality to the original device but will reduce the test time during manufacturing to lower the production cost. There are many types of DFT. Some DFT methodologies are well structured using industry-defined standards. Other methodologies are designed to solve some test problems for a specific device or category of devices. DFT is of crucial importance to the semiconductor industry. The main advantage of DFT is to guarantee high quality and reliability while reducing the test cost and decreasing the testing difficulty for designers.

4.3 Built-In Self-Test

Built-in self-test (BIST) is a DFT technique where additional hardware features are included into integrated circuits to carry out testing [1]. With BIST, a chip can perform test functions by itself. Test pattern generation and response analysis circuitry are built into the chip, which is akin to ATE going onto the chip. The concept of BIST has been used in the military-aerospace industry for decades and began to appear in the semiconductor industry about 15 years ago [5]. An excellent tutorial on BIST, including its functional and economic advantages, is given in [1].

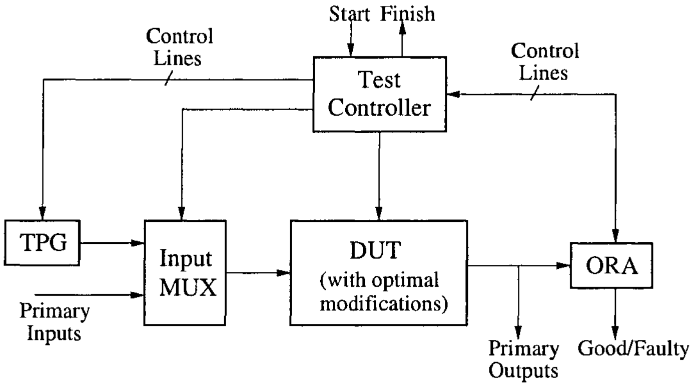


Figure 4.2 A typical BIST architecture for digital systems

4.4 BIST Structure

4.4.1 BIST for Digital Systems

BIST designs can differ in many ways. Fig. 4.2 shows a typical BIST architecture for digital systems. The self-test must be performed in a test mode which is initiated by a start signal, and whose completion is indicated by a complete signal. The test controller controls the operations of all the functional blocks. The test pattern generator (TPG) creates test vectors, which can be implemented using read-only memory (ROM) with stored patterns, a counter, or a linear feedback shift register (LFSR). The input multiplexer (MUX) selects between test pattern and primary inputs depending on signal coming from the test controller. The output response analyzer (ORA) performs compaction on the output response(s) of the device-under-test (DUT) when necessary and compares the results with the fault-free response(s). It can be implemented using a ROM with stored response, or a LFSR as a signature analyzer.

4.4.2. BIST for Analog Circuits

Analog testing is made more difficult by the wide variety of functionalities and design specifications of analog circuits. Several functional BIST schemes have been designed for special classes of analog and mixed-signal circuits with the use of analog-to-digital converters (ADC) and digital-to-analog converters (DAC) [4, 49]. Teraoka *et al.* used an on-chip digital signal processor (DSP) core to test an ADC and DAC embedded in a single-chip speech codec [47]. Tabatabaei and Ivanov presented a built-in current monitor suitable for supply current testing of analog circuit blocks [46]. Nagi, Chatterjee, and Abraham proposed a signature-analysis scheme to solve the tolerance problem posed by the imprecise nature of analog signals. Chatterjee has proposed a concurrent testing method for linear analog circuits using continuous check sums [10]. The state of the art in mixed-signal DFT and BIST has been described in a tutorial in [12].

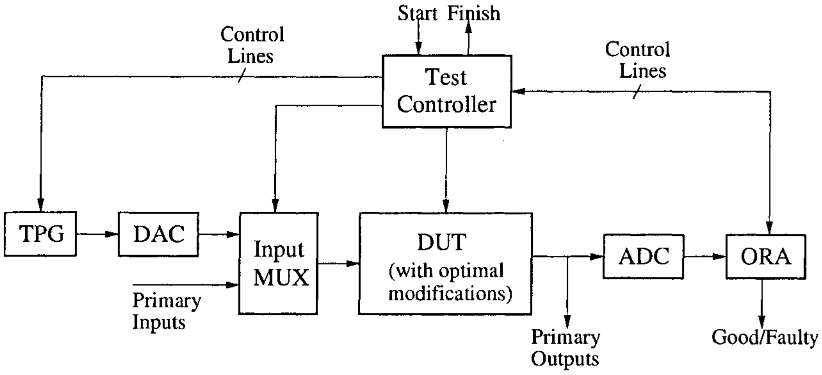


Figure 4.3 BIST architecture for analog systems

Fig. 4.3 shows a general BIST for analog circuits. This is similar to Fig. 4.2 except that a digital-to-analog converter (DAC) and an analog-to-digital converter (ADC) are added to convert digital test signals to analog waveforms and vice versa. However, the output response analyzer (ORA) works in a radically different way. In a mixed signal system, the sampling noise in the DAC and ADC combined with the processing and environmental variations in analog circuits will prevent an exact output response. One effective approach to solve this problem is to use a digital accumulator to obtain the sum of the magnitudes of the analog circuit test response. The accumulator-based ORA determines the pass/fail status of the BIST by evaluating whether the final sum is within a predetermined range of values [44].

4.5 Digital BIST for Analog Iterative Decoders

Advanced error correcting codes and/or coded modulation schemes are an essential part of most modern data transmission systems for both wireless and wireline applications. Analog iterative decoders have often been shown to have better performance than digital ones in terms of implementation complexity, silicon area, and power consumption [25, 40, 56]. Testing strategies are very important for all IC designs and this is certainly true for analog decoder designs. BIST is becoming a necessity.

This leads us to the following question. How can we test the analog iterative decoders in a simple way?

A common approach for testing analog decoders is to perform a sophisticated statistical test to confirm that the chip operates within design margins. However, it requires the test program to perform signal modulation, Gaussian noise insertion, and bit error counting, which makes testing very difficult. Moreover, it is often time-consuming to measure a suitably low bit error rate (BER) when performing a statistical analysis.

A possible solution to provide easy testing for analog decoders would be to build a digital BIST. A digital BIST scheme is favorable for analog iterative decoders because of its simplicity. First of all, it avoids the use of DACs to change the test pattern from the TPG into an analog signal. DACs demand significant silicon area and BISTs associated with DACs are often complex due to their sampling noise [36]. Second, when using digital BIST schemes, decoders can be divided into sub-circuits and can be tested separately with simpler test methods as analog BIST schemes cannot do so. Third, if analog iterative decoders are tested in the analog domain, it will be complicated and time-consuming. In [56], Winstead *et al.* illustrate that it is the nature of analog iterative decoders to hide errors until they present themselves at a low BER after fabrication. Fig. 4.4 presents the BER curves of a MAP decoder for an (8, 4) Hamming code with and without defects. Defects cause an error floor[1] at a low BER rate and note that this is for a very small/weak (8, 4) Hamming code. In a larger-scale decoder, a fault can be much less obvious because the code is so powerful. Finally, in [59], large-scale analog iterative decoders have been shown to be tolerant to device mismatch under two conditions. First, the mismatch standard deviation must not exceed a critical value, beyond which the decoder's performance degrades beyond acceptable limits. Fig. 4.5 shows the threshold[2] loss for several regular LDPC ensembles due to mismatch. According to Fig. 4.5, large analog LDPC decoders should be robust to intrinsic errors even if the mismatch is as high as 10 - 25%. Second, all transistors must respond to their gate inputs, which means there are no stuck-at faults. The first condition is statistical and can be met by design. In CMOS processes, we can meet mismatch targets by increasing the transistor size. The second condition applies to all transistors, and this can be guaranteed by digital BIST where the digital behavior of Gilbert multipliers is observed. Digital BIST can detect a high percentage of stuck-at fault errors for analog iterative decoders. In order to test the analog decoders in the digital domain, modifications of the decoder circuits are required. This will be explained in Chapter 5.

[1] An error floor is the segment of the BER curve towards high SNR where the curve has a shallower slope.

[2] The threshold for a regular LDPC code is defined as the minimum signal-to-noise ratio, which allows error-free decoding, for an infinite length code.

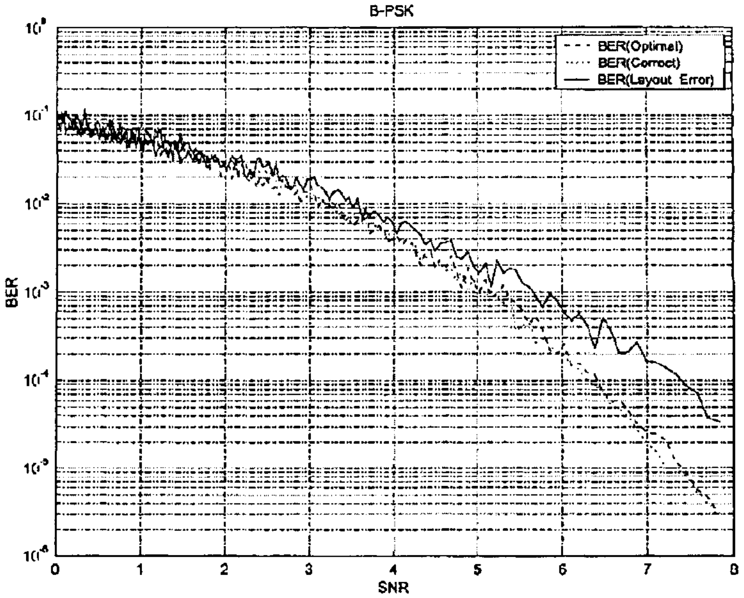


Figure 4.4 BER curves of an (8, 4) Hamming MAP decoder with/without defects [56]

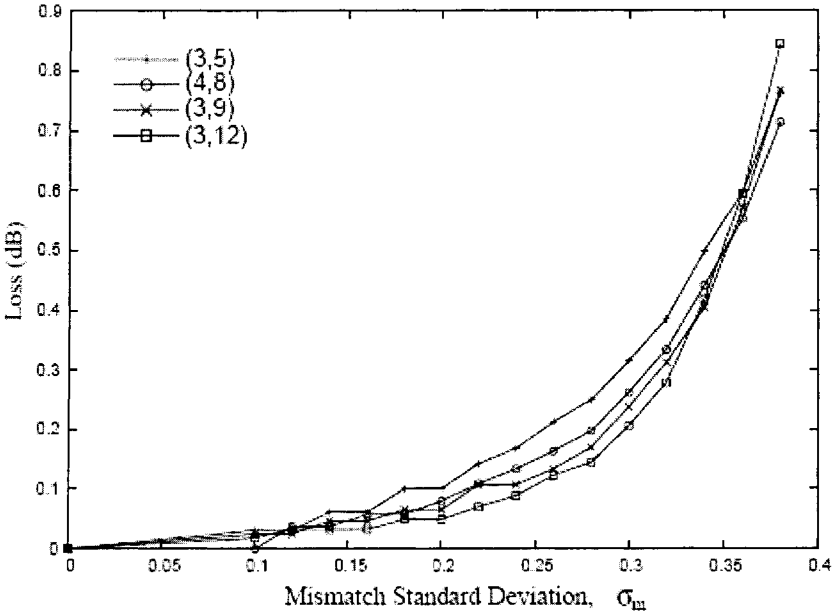


Figure 4.5 Threshold loss due to mismatch for several regular LDPC ensembles [59]

4.6 Chapter Summary

In this chapter, background information on VLSI testing and DFT are introduced. We also presented the concept of BIST along with its advantages and general architectures. In addition, we have discussed the reasons for designing a digital BIST scheme for analog iterative decoders. In the next chapter, we will present a BIST scheme suitable for analog iterative decoders and how it is implemented for an (8, 4) extended Hamming decoder.

Chapter 5

Digital BIST and Implementation

5.1 Digital BIST Scheme

5.1.1 Proposed Methodology

Fig. 5.1 shows a system block diagram for the (8, 4) Hamming code decoder chip. Chip designs for other codes are similar to this one but with different numbers of sample-and-hold circuits (S/H) and comparators depending on the block length and rate of the code. The serial-to-parallel interface was originally developed by S. Yu *et al.* [62, 63]. Operation of the interface is synchronized by a clock signal. It converts the serial data stream received from the channel to a parallel set of inputs passed to the computation core. The computation core represents the analog network, which performs probability propagation corresponding to the factor graph. The bank of comparators makes digital decisions according to the core outputs and latches all values at once.

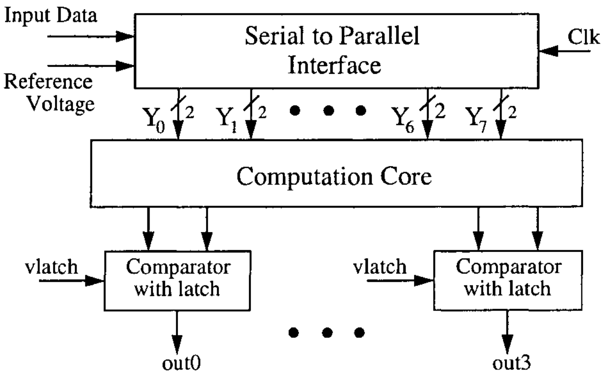


Figure 5.1 Structure of a general analog iterative decoder chip

The proposed strategy for testing the decoder chip is to divide the entire chip into two sections and to design a BIST for each of these sections. The first section of the chip is the computation core itself, and the second section is the input/output (I/O) interface that contains the S/H cells and the comparators. Fig. 5.2 shows the structure of the decoder chip with BISTs.

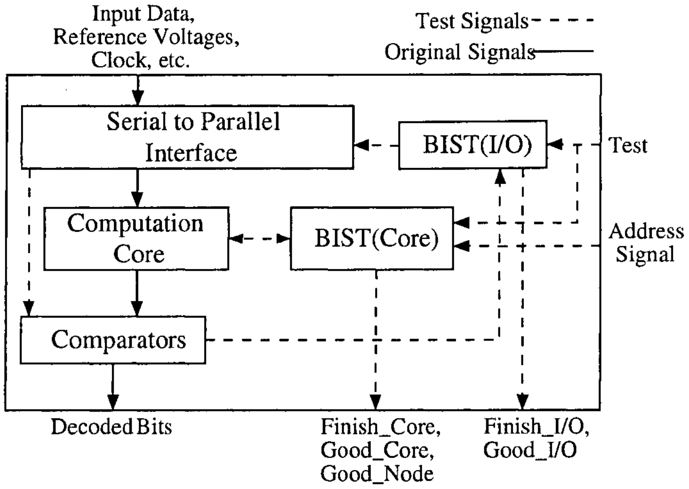


Figure 5.2 Decoder chip with BIST

5.1.2 BIST for the Computation Core

The computation core in Fig. 5.1 represents the analog network, which corresponds to a factor graph. All the nodes with more than three edges are constructed by connecting 3-edge bi-directional nodes in a chain, and all 3-edge bi-directional nodes are constructed by connecting three unidirectional circuits. Therefore, one needs only to consider the implementation of unidirectional nodes from which larger designs can be built. Thus the computation core contains only unidirectional equality and check nodes based on Gilbert multipliers. As a result, the BIST scheme is consequently developed to test each unidirectional node separately. Since the computation core consists of only two distinct elements, equality nodes and check nodes, we only have to deal with two types of circuits, shown in Fig. 3.5 and Fig. 3.6, during testing.

The purpose of the BIST is to check if all transistors in the unidirectional circuits are working properly. A digital BIST can simply carry out such a test for the circuits. In order to do so, the analog circuit needs to be converted into a digital logic gate during test mode.

Fig. 5.3 demonstrates how the check node circuit is modified to allow digital testing. Note that the original circuit is the same as in Fig. 3.5 except the current source is represented at the transistor level and the input diode-connected transistors are moved to the outputs. The circuit becomes a voltage-in, voltage-out sum-product module, which offers an alternate way to make connections between nodes. Modifications of the check node circuit are done by adding switches to reroute the wires and shifting the bias voltage and reference voltages to ground (*Gnd*) or the supply voltage (V_{DD}). When feeding in ones and zeros to the modified circuit, it behaves like a static CMOS logic gate. During test mode, the 'Dig' signal goes to V_{DD} and the 'Ang' signal goes to *Gnd*. This reconfigures M1-M4 to form two inverters. In effect, the entire circuit acts like a differential XOR gate. Table 5.1 describes the behavior of the modified circuit during test mode.

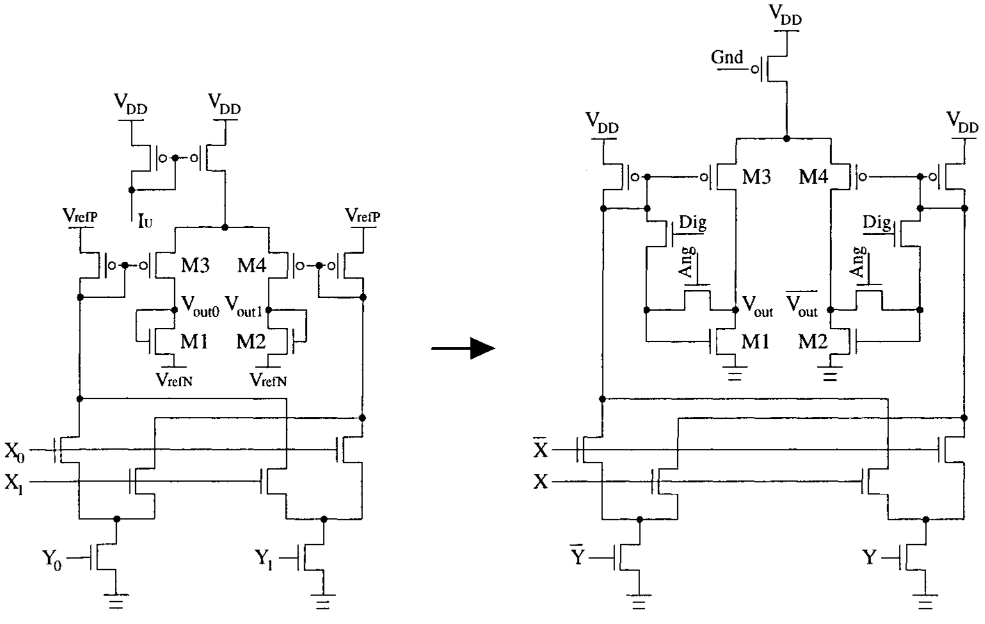


Figure 5.3 Modifications of a check node sum-product circuit

Table 5.1 Logic behavior of the modified sum-product circuit

X	Y	Vout	$\overline{\text{Vout}}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1
Vout = $\overline{X}Y + X\overline{Y}$			

The same topology is used to test the unidirectional equality nodes. Fig. 5.4 demonstrates how the equality node circuit is modified. The only difference is that four more transistors are added to transform the equality node circuit into a check node circuit during test mode. Therefore, the equality node circuit also acts like an XOR gate during testing and Table 5.1 is also suitable for the modified equality node circuit.

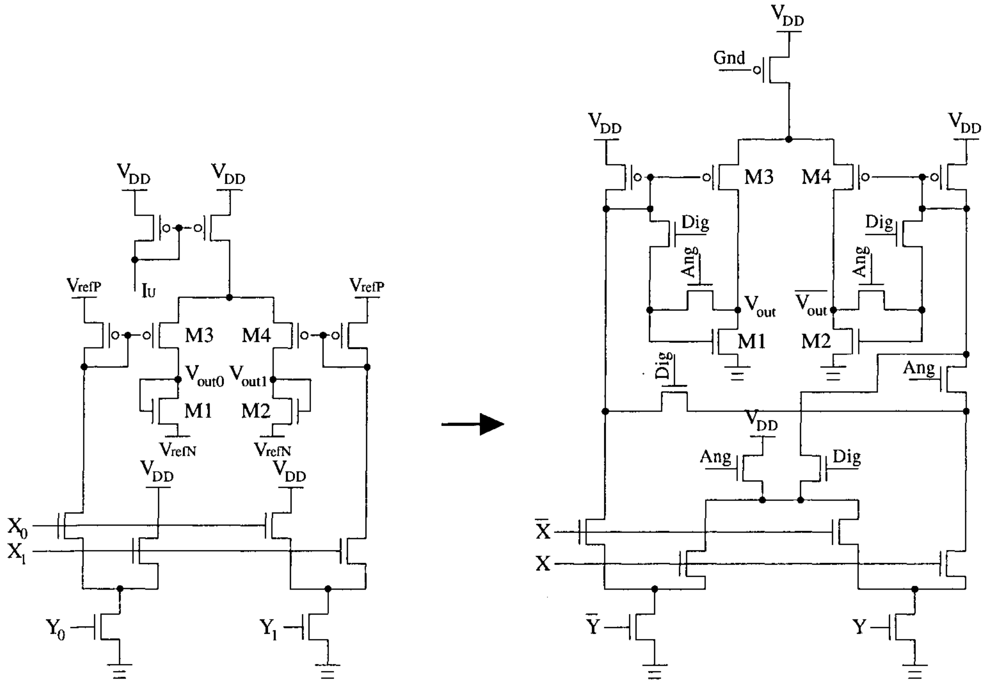


Figure 5.4 Modifications of an equality node sum-product circuit

When testing the computation core, all of the unidirectional node circuits have to be disconnected and this can be done by adding switches as illustrated in Fig. 5.5. Switches can be built by using *transmission gates*. To start self-test, the BIST sends a signal to the computation core to disconnect all circuits and to transform them into logic gates. Afterward, it sends test vectors XY = 00, 01, 11, 10 into each node and analyzes the response from each node according to Table 5.1. It then indicates if the core is good or faulty. When testing is completed, the BIST sends out a finish signal and waits for an input signal to convert the core back to normal operation. Fig. 5.6 shows a flow chart describing the self-test operations.

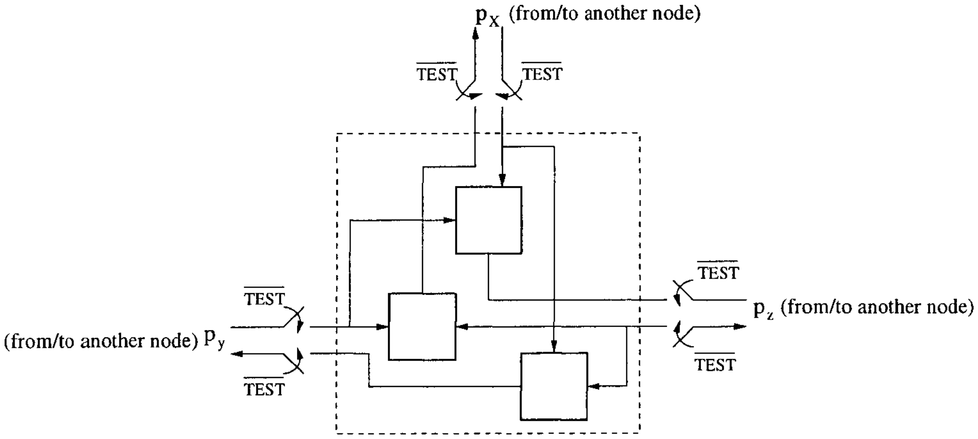


Figure 5.5 Unidirectional circuits disconnected by additional switches

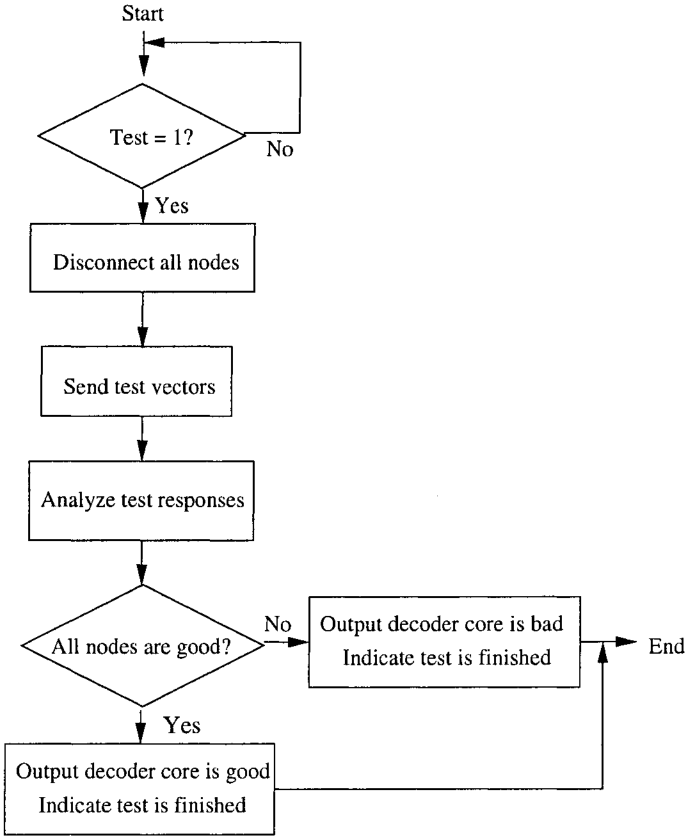


Figure 5.6 Flow chart describing the decoder self-test operations

For test debugging purposes, an optional feature is added to the BIST. The test results of three unidirectional circuits are gathered into one group, called a 'check group' or 'equality group' for check nodes or equality nodes, respectively. Through the address signal, one can access the test result of each group to see if any part of the core is not working.

5.1.3 BIST for I/O Interface

The serial-to-parallel interface in Fig. 5.1 consists of some sample-and-hold (S/H) cells and these cells are composed of one switch for selection, one capacitor for storing the data, and one buffer for isolating the data if there is another stage following [63]. Fig. 5.7 shows the schematic of a single S/H input stage.

The purpose of the I/O BIST is to ensure the switches are working, the capacitors are storing correct values, and the comparators are making proper decisions. A simple BIST can be built to test the I/O interface. During test mode, test patterns are sent and stored into the S/H cells. The stored values are then converted into probabilities and they bypass the computation core and directly reach the comparators. Finally, outputs from the comparators are checked. Two signals are created to indicate test completion and the test result. Although the I/O interface uses continuous analog voltages as inputs to the S/H cells, a 1-bit test is sufficient to detect severe faults.

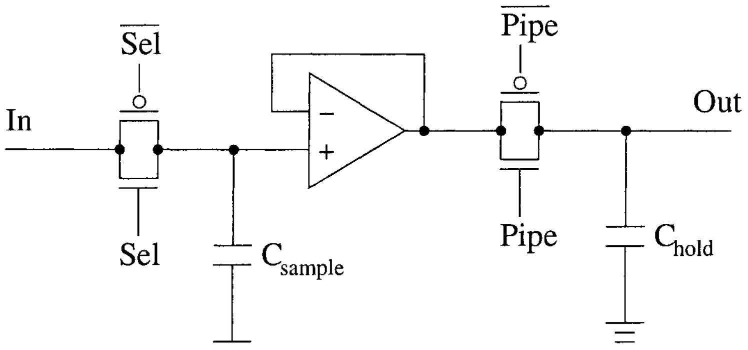


Figure 5.7 Single S/H input stage

5.2 Implementation

A proof-of-concept implementation of the BIST scheme proposed in the previous sections was built for a CMOS analog decoder for an (8, 4) extended Hamming code. The implementation details are explained in this section. The decoder was designed using 0.18 μm 6M1P CMOS technology.

5.2.1 Analog (8, 4) Hamming Decoder Core

The (8, 4) Hamming decoder has the following generator matrix and parity check matrix:

$$\mathbf{G}=\begin{bmatrix}1&0&0&0&1&0&1&1\\0&1&0&0&1&1&1&0\\0&0&1&0&1&1&0&1\\0&0&0&1&0&1&1&1\end{bmatrix},\qquad \mathbf{H}=\begin{bmatrix}1&1&1&0&1&0&0&0\\0&1&1&1&0&1&0&0\\1&0&0&0&1&0&1&1\\0&0&0&1&0&1&1&1\\0&1&0&1&1&0&0&1\\1&1&0&0&0&1&0&1\\1&0&1&0&0&1&1&0\\0&0&1&1&1&0&1&0\end{bmatrix}.$$

The **H** matrix has a size of 8x8 because redundant bits are added to improve the performance of the decoder at high SNRs [31]. The factor graph corresponding to the **H** matrix is shown in Fig. 5.8. When computing the final values Y_n, we need to include the intrinsic information U_n from the channel to correctly produce a reliable decision. The factor graph is then transformed to that shown in Fig. 5.9. The larger equality nodes compute extrinsic information from the connected check nodes and the smaller equality nodes calculate the outputs Y_n by using both extrinsic information and intrinsic information.

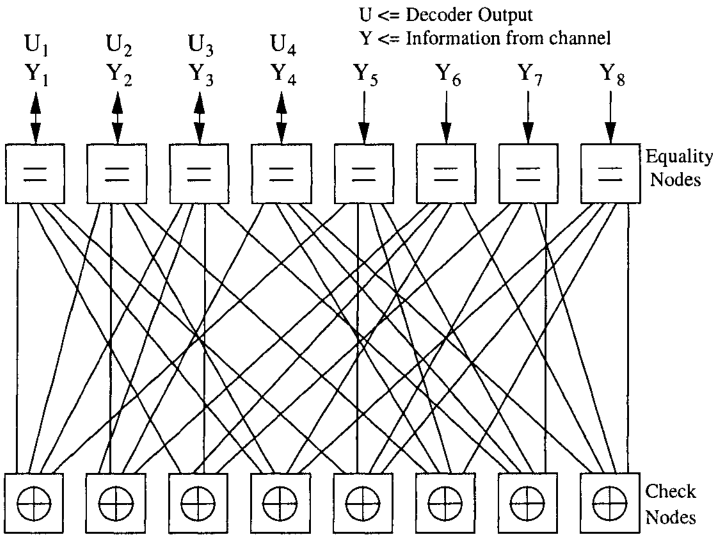


Figure 5.8 Factor graph of (8, 4) Hamming decoder

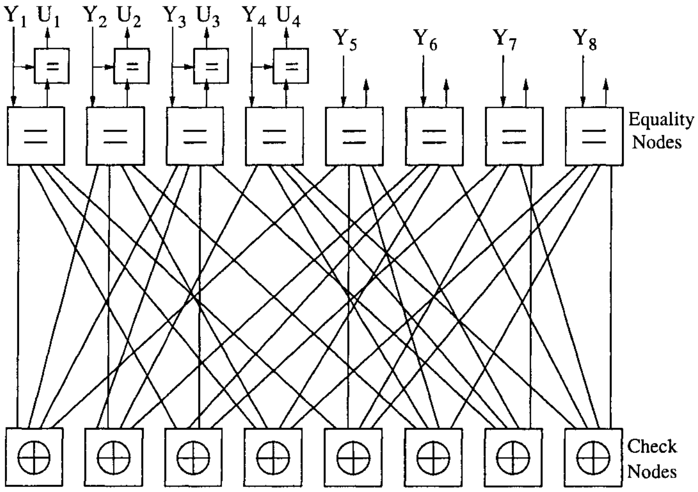


Figure 5.9 Transformed factor graph of (8, 4) Hamming decoder

Implementation of check nodes:

Let us first look at the most basic unidirectional check node circuits shown in Fig. 5.10 and Fig. 5.11, respectively. The circuit in Fig. 5.10 is named CHECK1 and the circuit in Fig. 5.11 is named CHECK1_NG. Both schematics have the same functionality except the latter has an extra pair of diode-connected transistors at the output along with some switches for digital testing. This is because the output of a unidirectional node circuit may feed into the *X* input or *Y* input of other nodes, or to the *X* input of one node and *Y* input of another node. Therefore, the output diode-connected transistors may have to have a reference voltage of V_{refN} or *Gnd* (V_{SS}), or both. In both circuits, the transistors which function as switches have the minimum transistor size W/L = 0.42 μm/0.18 μm and the rest of the transistors which perform decoding were sized as W/L = 1 μm/0.3 μm to reduce the mismatch variance.

As mentioned in Section 3.3.3, connecting three unidirectional circuits forms a bi-directional node. Fig. 5.12 and 5.13 show the bi-directional three-edge check node, named CHECK3_1NG and CHECK3_2NG. The first contains one extra switch and this switch is used for error injection. This feature is added for testing purposes to see if the BIST can detect and determine the locations of errors. Therefore, when performing self-test, one can inject an error for every CHECK3_1NG circuit in the decoder core and the BIST should be able to detect the fault. The SWITCH_PPT block is used to switch between V_{refP} and V_{DD} depending whether the device is operating in decoding mode or test mode. The SWITCH_PPT is built using two PMOS pass transistors and is illustrated in Fig. 5.14(a). The SWITCH blocks are used to separate each unidirectional circuit during test mode and are built by transmission gates illustrated in Fig. 5.14(c). These switches also work as input multiplexers (MUXs) to select between analog decoding inputs and digital test inputs.

Each check node in the factor graph of Fig. 5.1 has four edges. This is implemented by combining two three-edge bi-directional check nodes (CHECK3_XNG) into a CHECK4 node as demonstrated in Fig. 5.15. The SWITCH_NPT block is used to switch between V_{refN} and V_{SS} depending on which mode the decoder is operating in, and its schematic is shown in Fig. 5.14(b). The SWITCH blocks function as de-multiplexers used to direct the outputs of the CHECK3_XNG nodes to either the BIST or the connected equality nodes, also depending on which mode the decoder is using.

We mentioned in Section 5.1.2 that the test results of three CHECK1/CHECK1_NG nodes are gathered to one group and can be accessed through an address signal. Three CHECK1/CHECK1_NG nodes actually become a CHECK3_XNG node, so we can access the test results of each CHECK3_XNG in the decoder. The factor graph in Fig. 5.1 has eight CHECK4 nodes; therefore, test results of sixteen CHECK3_XNG nodes can be accessed in the decoder.

Implementation of equality nodes:

The idea used for building four-edge check nodes (CHECK4) can be applied to build a five-edge equality node (EQUALITY5). Fig. 5.16 and Fig. 5.17 present the unidirectional equality node circuit with and without an extra pair of diode-connected transistors and they are named EQUALITY1 and EQUALITY1_NG, respectively. Fig. 5.18 and Fig. 5.19 shows the EQUALITY3 and EQUALITY3_NG circuits. EQUALITY3 has an extra switch to inject errors. The five-edge equality node named EQUALITY5 can be built by using three EQUALITY_3 or EQUALITY3_NG blocks as illustrated in Fig. 5.20. There are eight EQUALITY5 nodes in the factor graph; therefore, test results of twenty-four EQUALITY3/EQUALITY3_NG nodes can be accessed in the decoder through an address signal.

The smaller equality node in the factor graph is actually a unidirectional equality node circuit (EQUALITY1) without the diode-connected transistors at the outputs. It receives the intrinsic information form the channel and the extrinsic information from the larger equality node and calculates the final decoded outputs. Since it is a voltage-in current-out circuit, we called it EQUALITY1_IOUT. The circuit is shown in Fig. 5.21. Two invertors are added to the circuit and they perform inversion only when the circuit is in test mode. The diode-connected transistors are removed for the EQUALITY1_IOUT circuit; therefore, invertors are necessary to produce logic outputs during test mode. The test results of the four EQUALITY1_IOUT circuits are gathered into one group. Fig. 5.22 shows the EQUALITY_OUT that contains the EQUALITY1_IOUT and the input MUXs.

Implementation of factor graph decoder core:

The (8, 4) Hamming decoder described by the factor graph in Fig. 5.1 can be directly mapped to the analog circuit network shown in Fig. 5.23. The factor graph is built by connecting the CHECK4 nodes, EQUALITY5 nodes and the EQUALITY_OUT nodes according to the edges in the factor graph. During the decoding process, probability computations are performed within each node and results are passed back and forth between the CHECK4 and EQUALTIY5 nodes. Then the EQUALITY_OUT computes the final current outputs and passes the results to the comparators of the output interface to make final bit decisions. Some reset circuits are also built to initialize the interconnections between the EQUALITY5 nodes and CHECK4 nodes. Probabilities are equalized after each codeword is decoded. The circuit is shown in Fig. 5.24 and named as RESET.

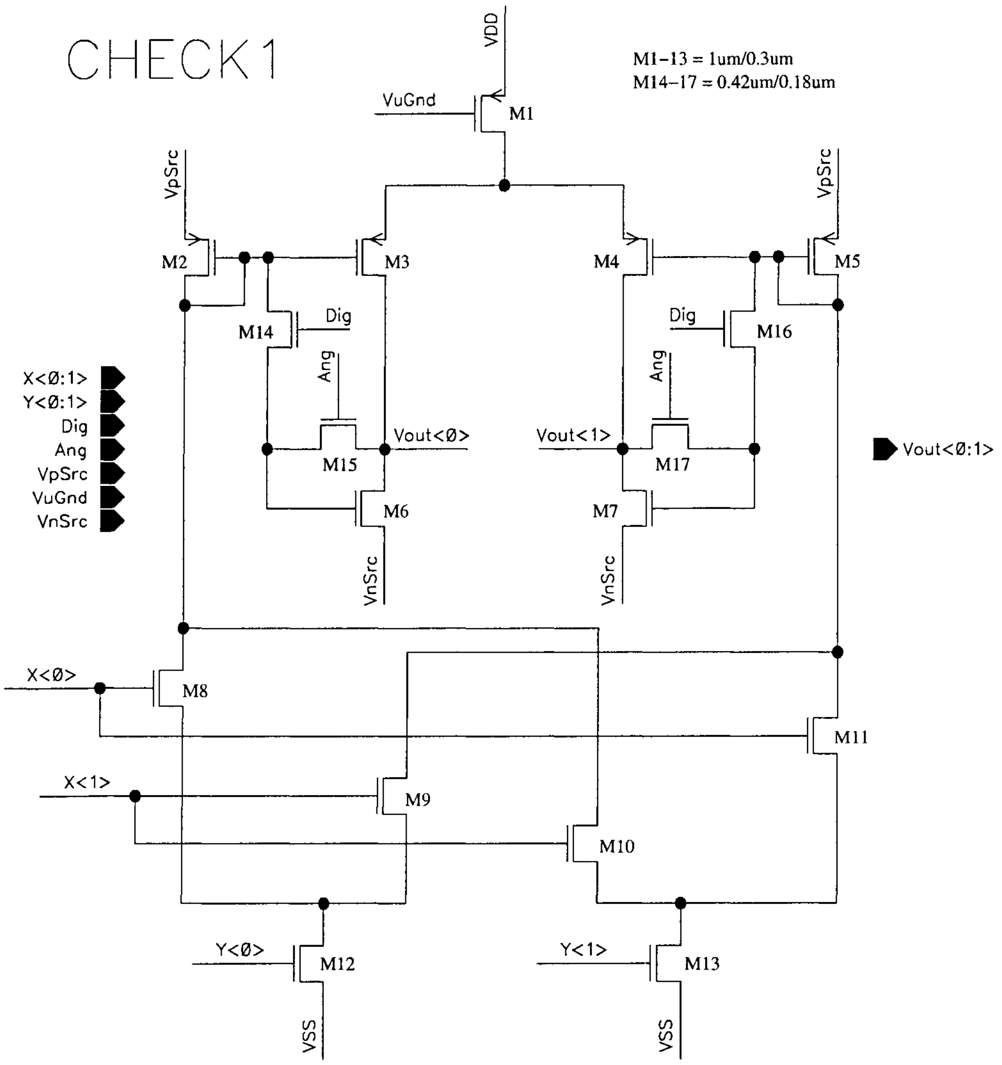


Figure 5.10 Unidirectional check node circuit (CHECK1)

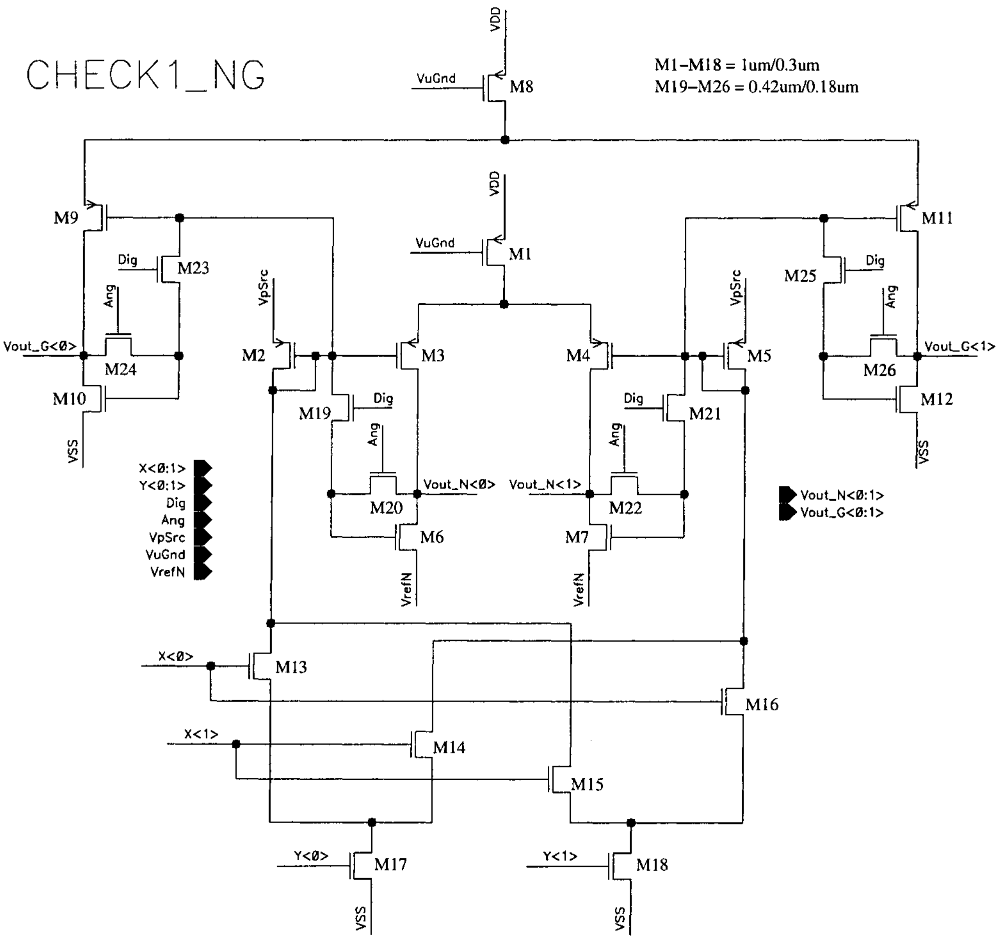


Figure 5.11 Unidirectional check node circuit with extra pair of diode connected output transistors. (CHECK1_NG)

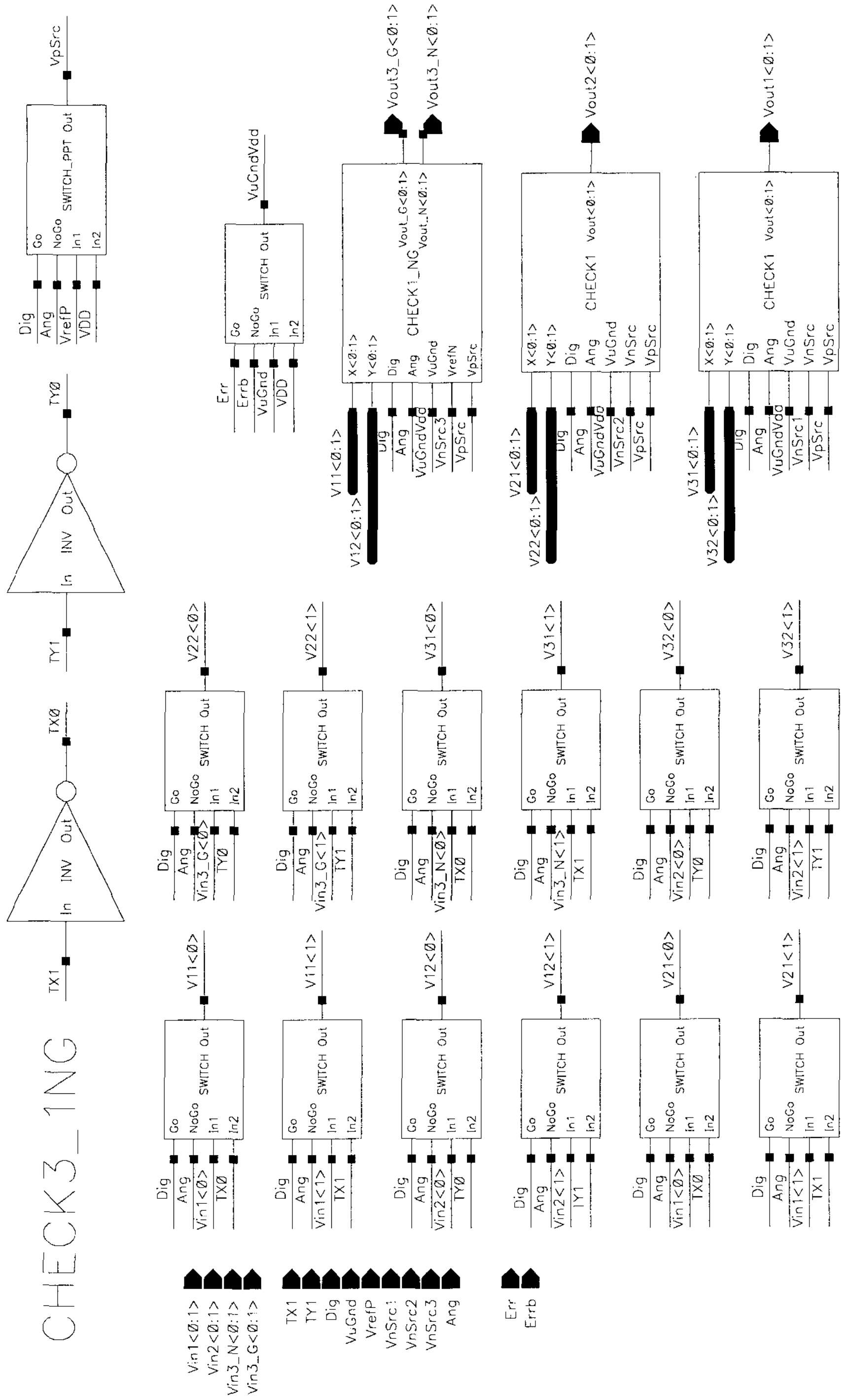


Figure 5.12 Bi-directional three-edge check node (CHECK3_1NG)

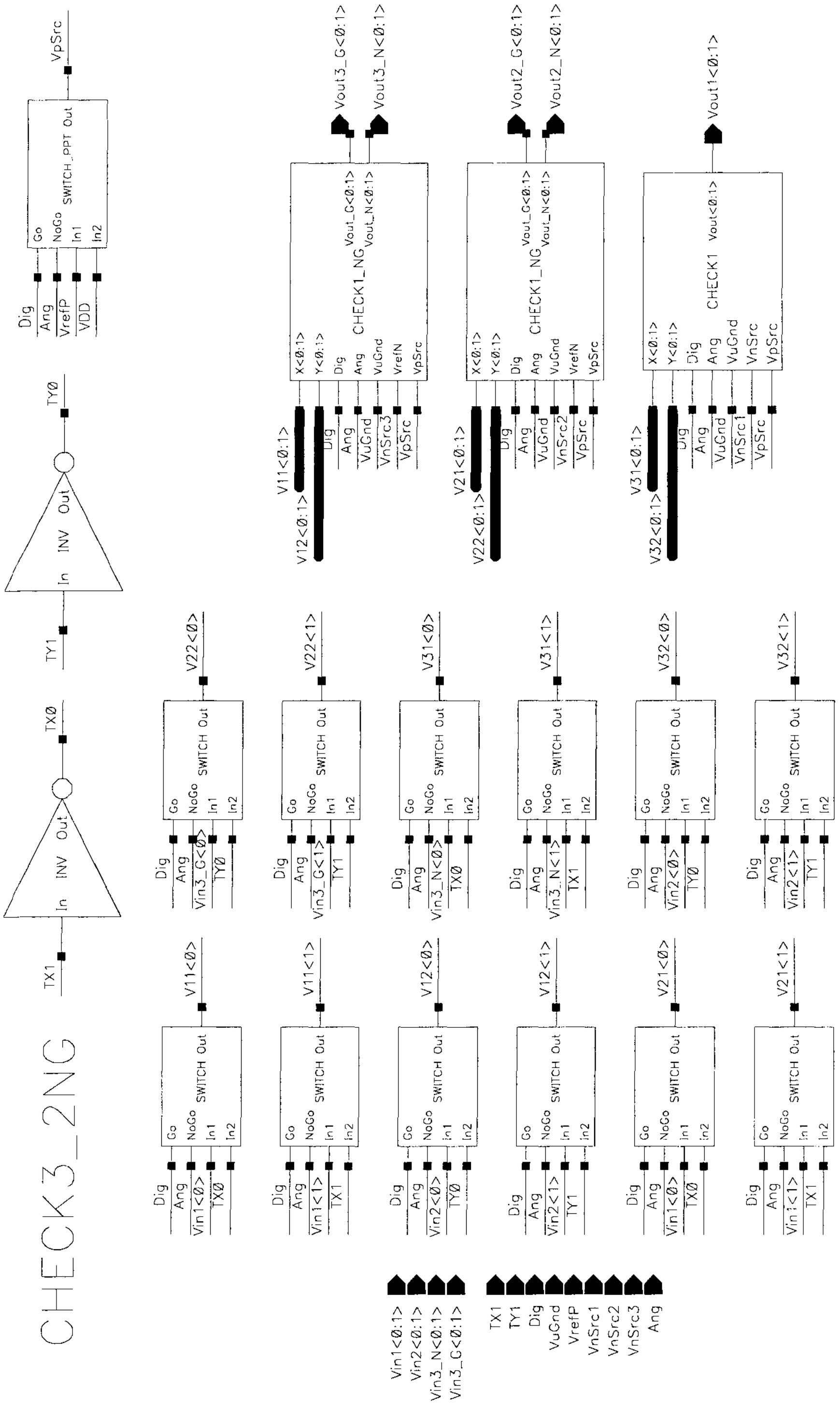


Figure 5.13 Bi-directional three-edge check node (CHECK3_2NG)

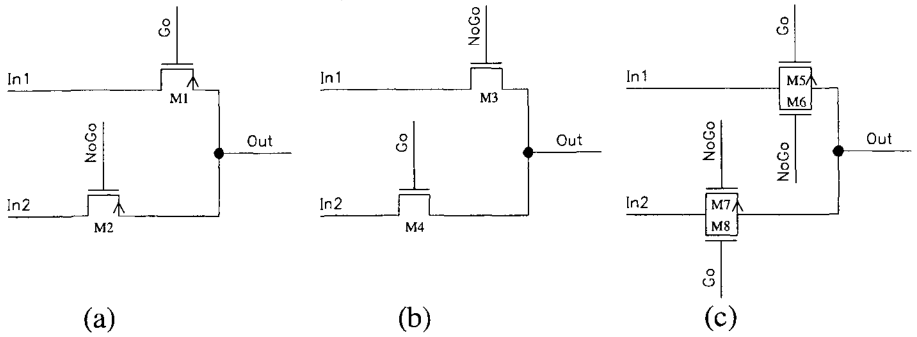


Figure 5.14 Switches implemented by: (a) PMOS pass transistor (b) NMOS pass transistor (c) Transmission gate

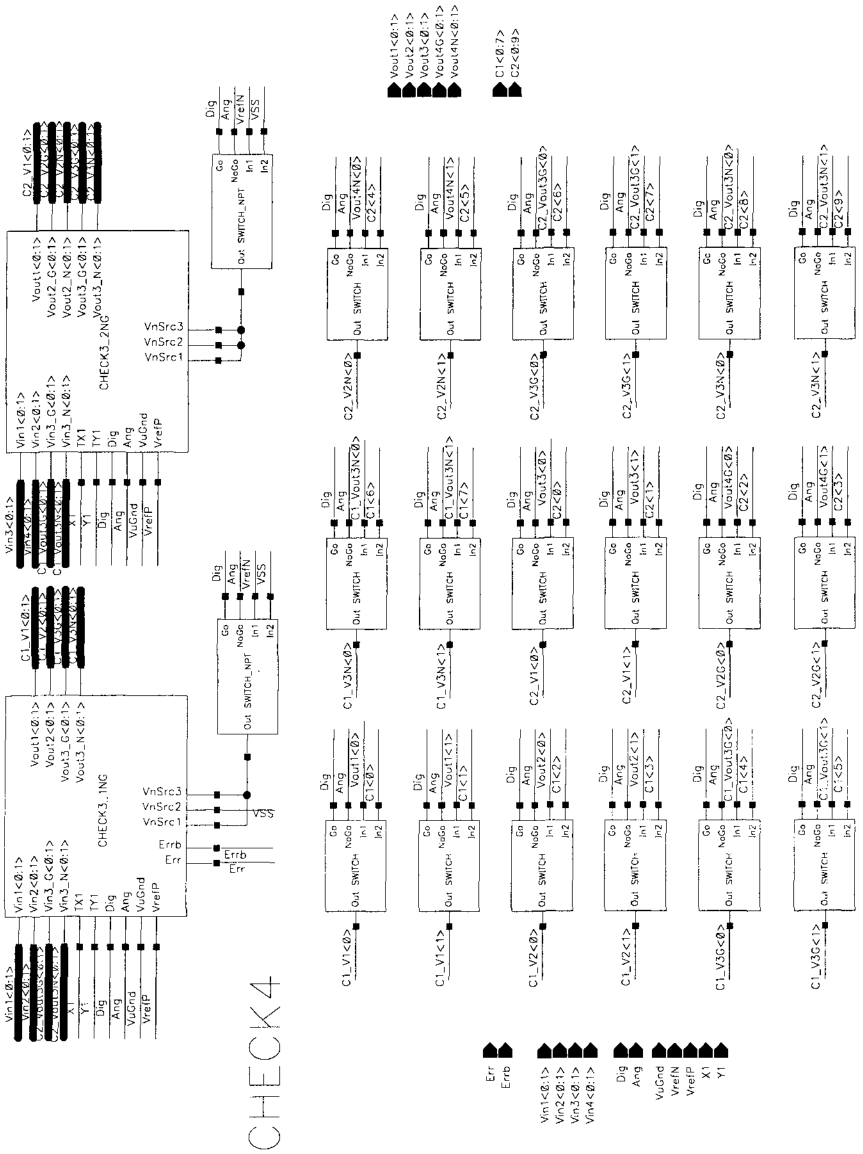


Figure 5.15 Four-edge check node (CHECK4)

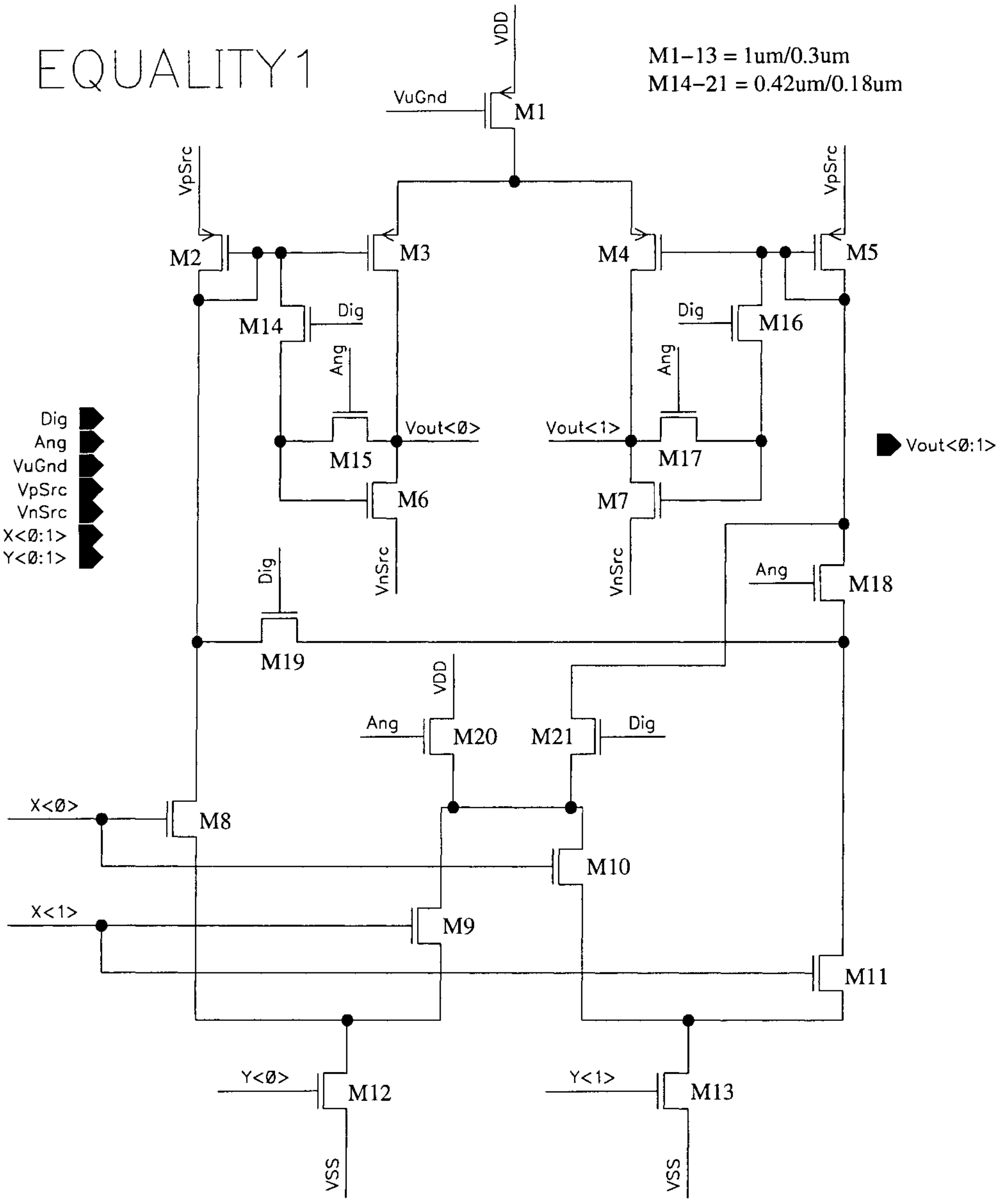


Figure 5.16 Unidirectional equality node circuit (EQUALITY1)

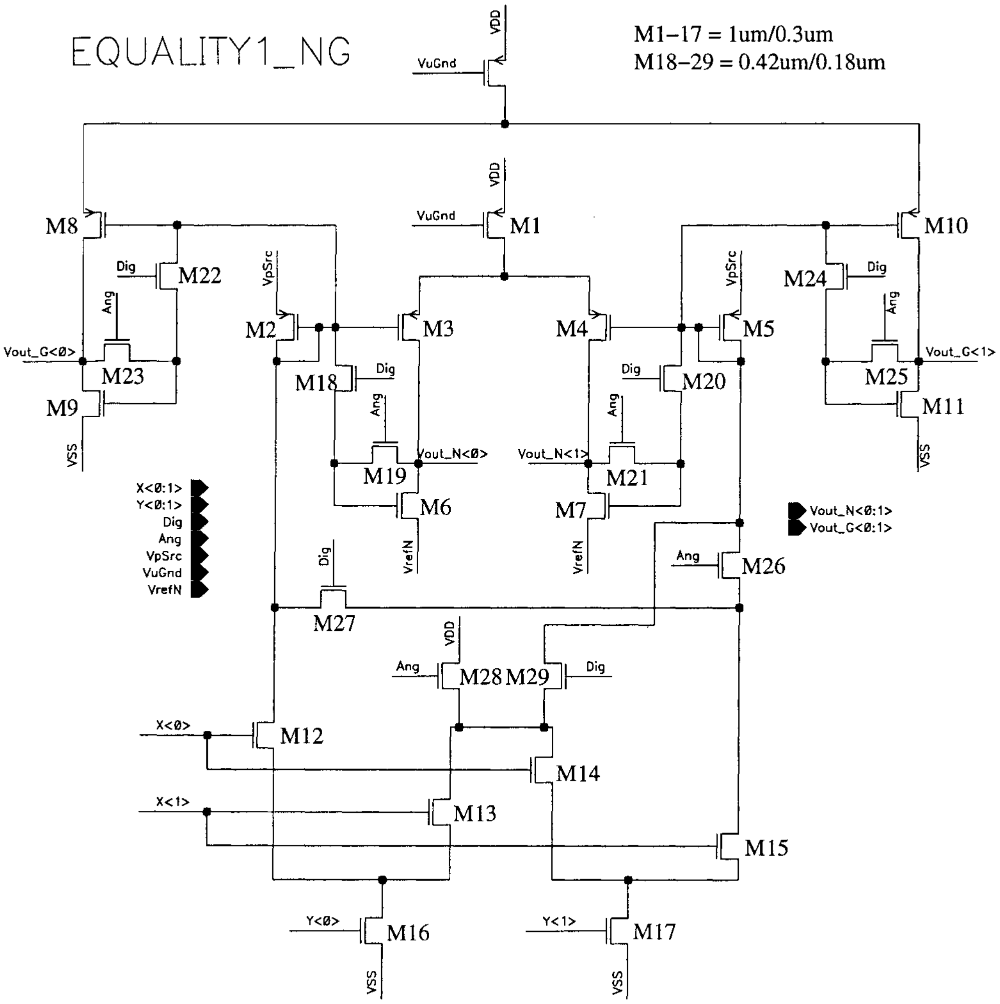


Figure 5.17 Unidirectional equality node circuit with extra pair of diode-connected transistors (EQUALIY1_NG)

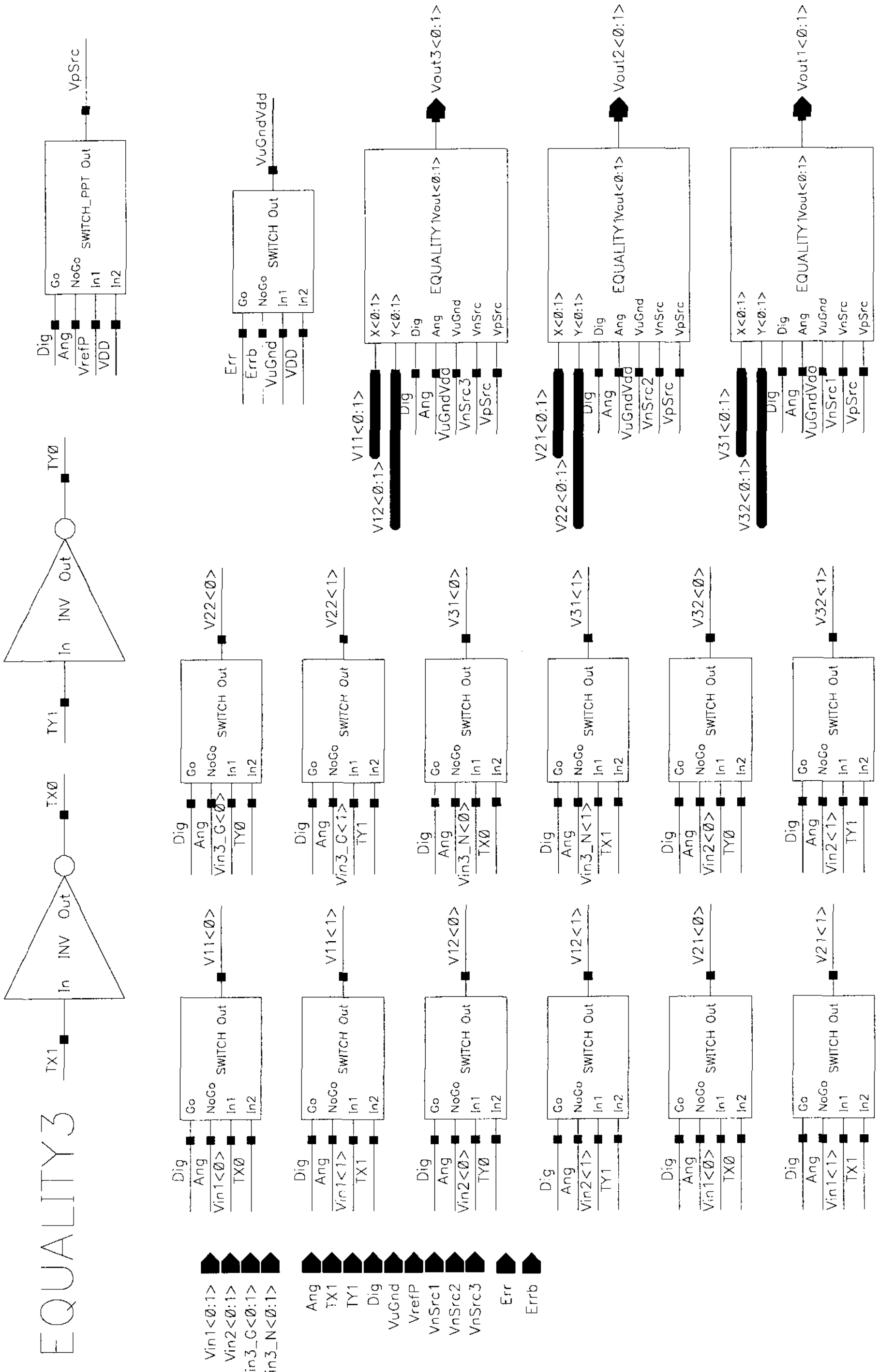


Figure 5.18 Bi-directional three-edge equality node (EQUALITY3)

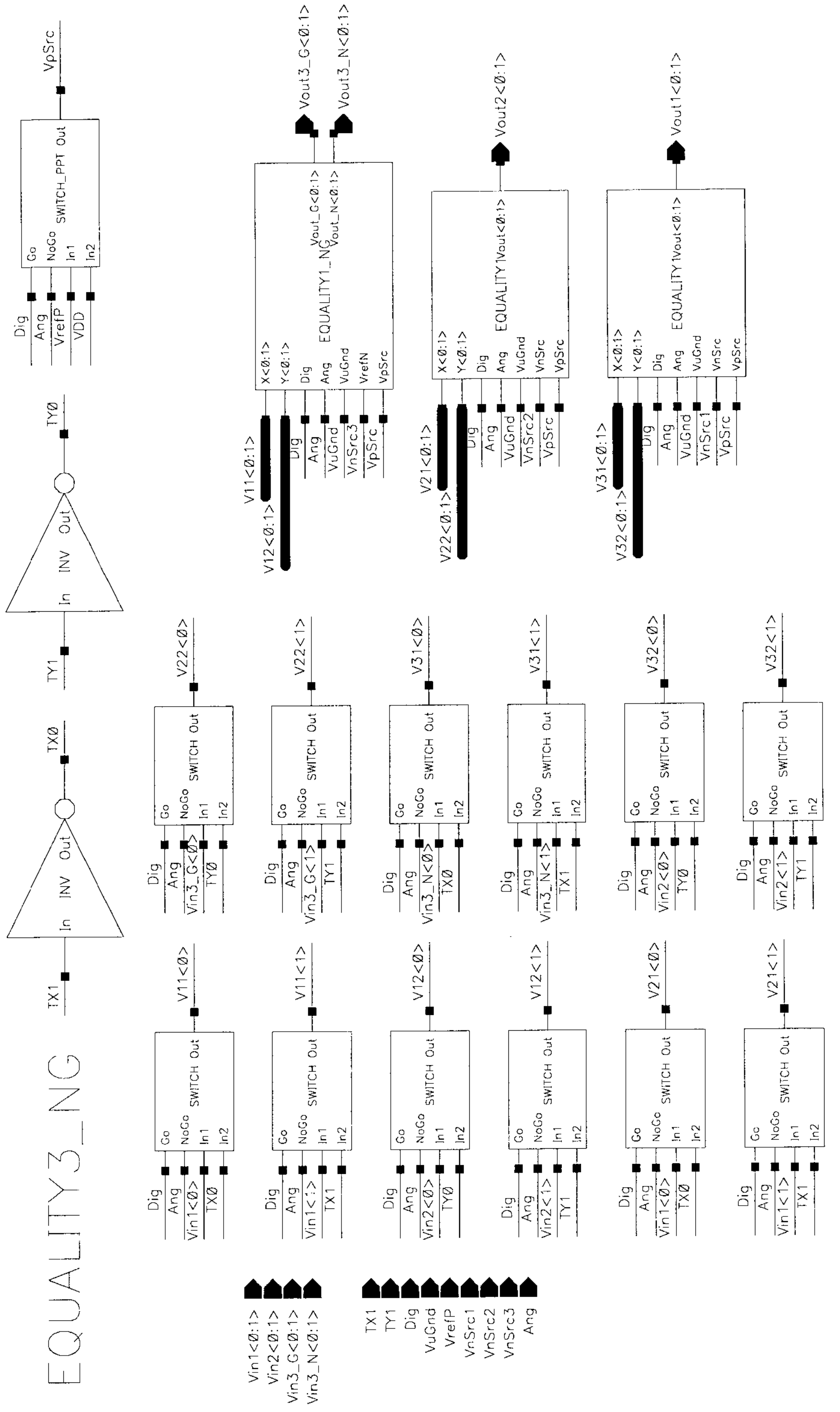


Figure 5.19 Bi-directional three-edge equality node (EQUALIY3_NG)

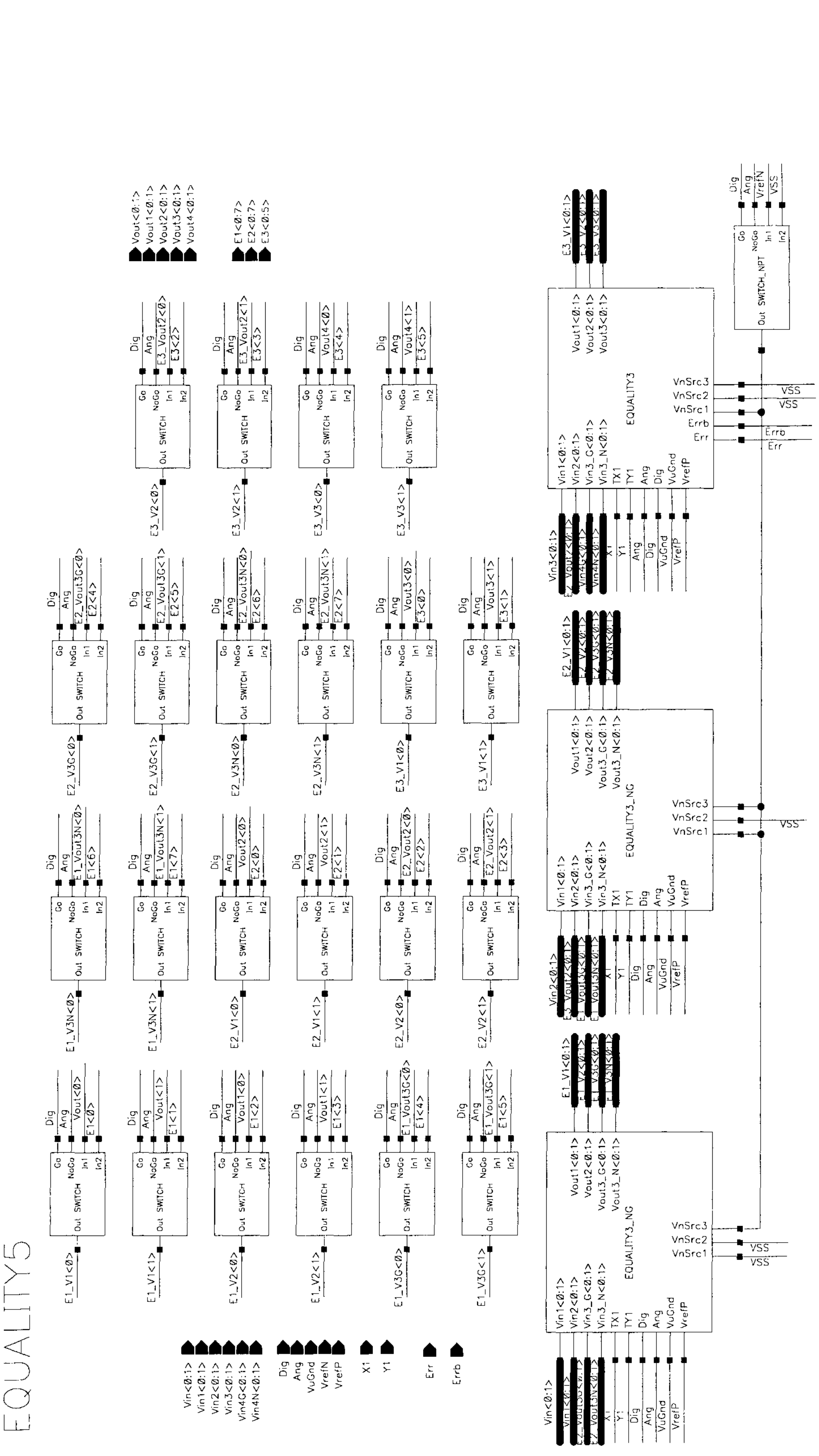


Figure 5.20 Five-edge equality node (EQUALITY5)

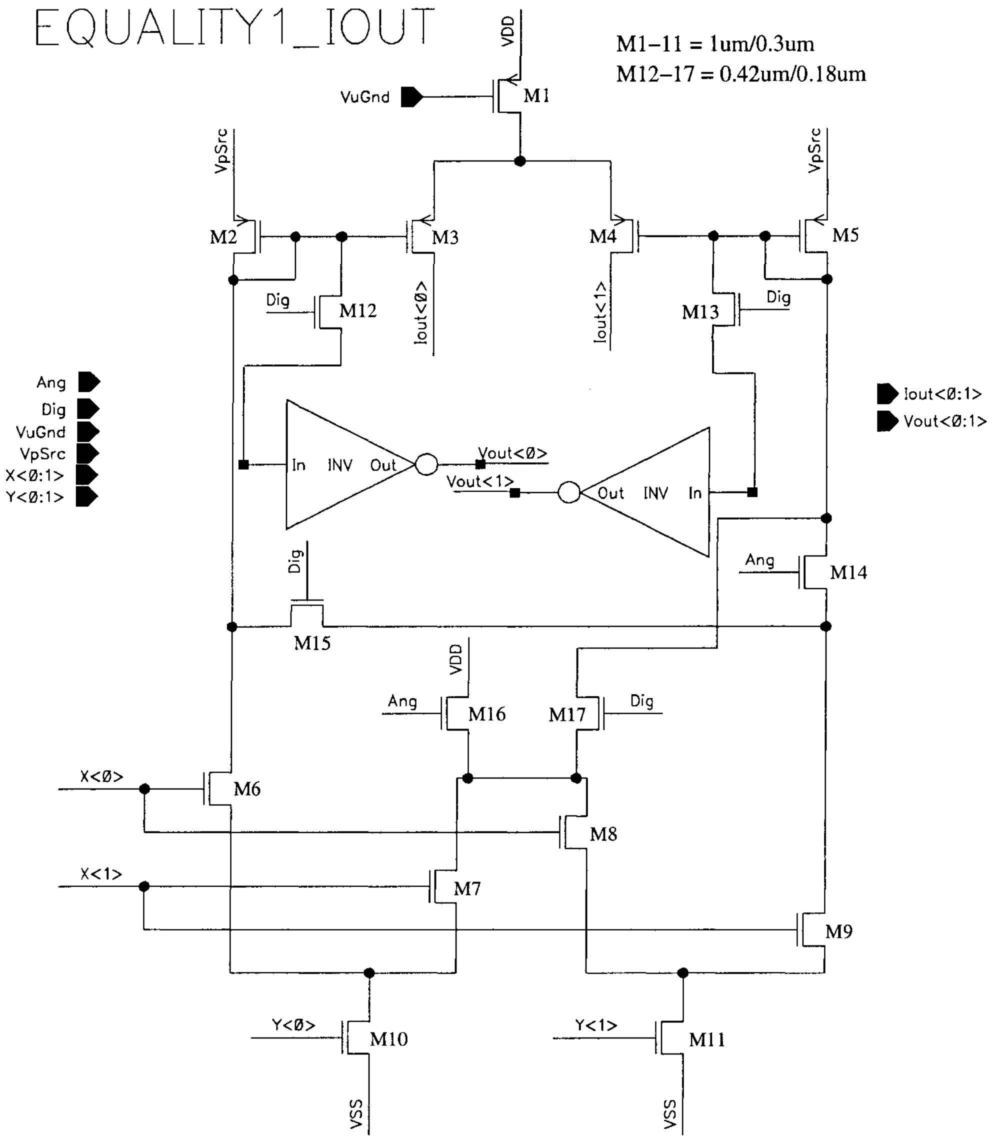


Figure 5.21 Unidirectional equality node circuit with current outputs (EQUALITY_IOUT)

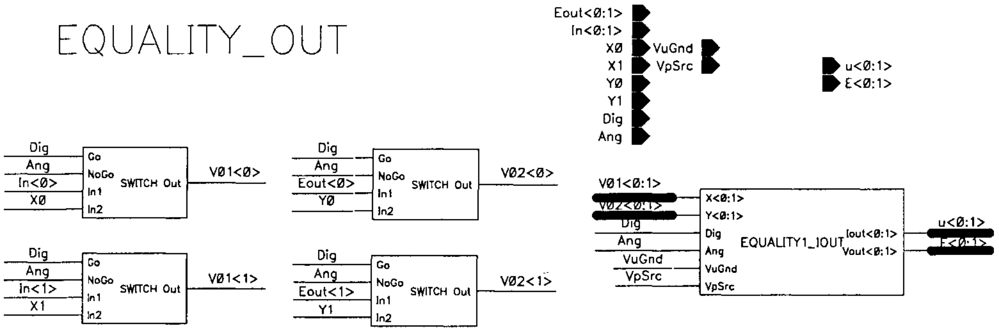


Figure 5.22 EQUALITY1_IOUT with switches as input MUXs (EQUALITY_OUT)

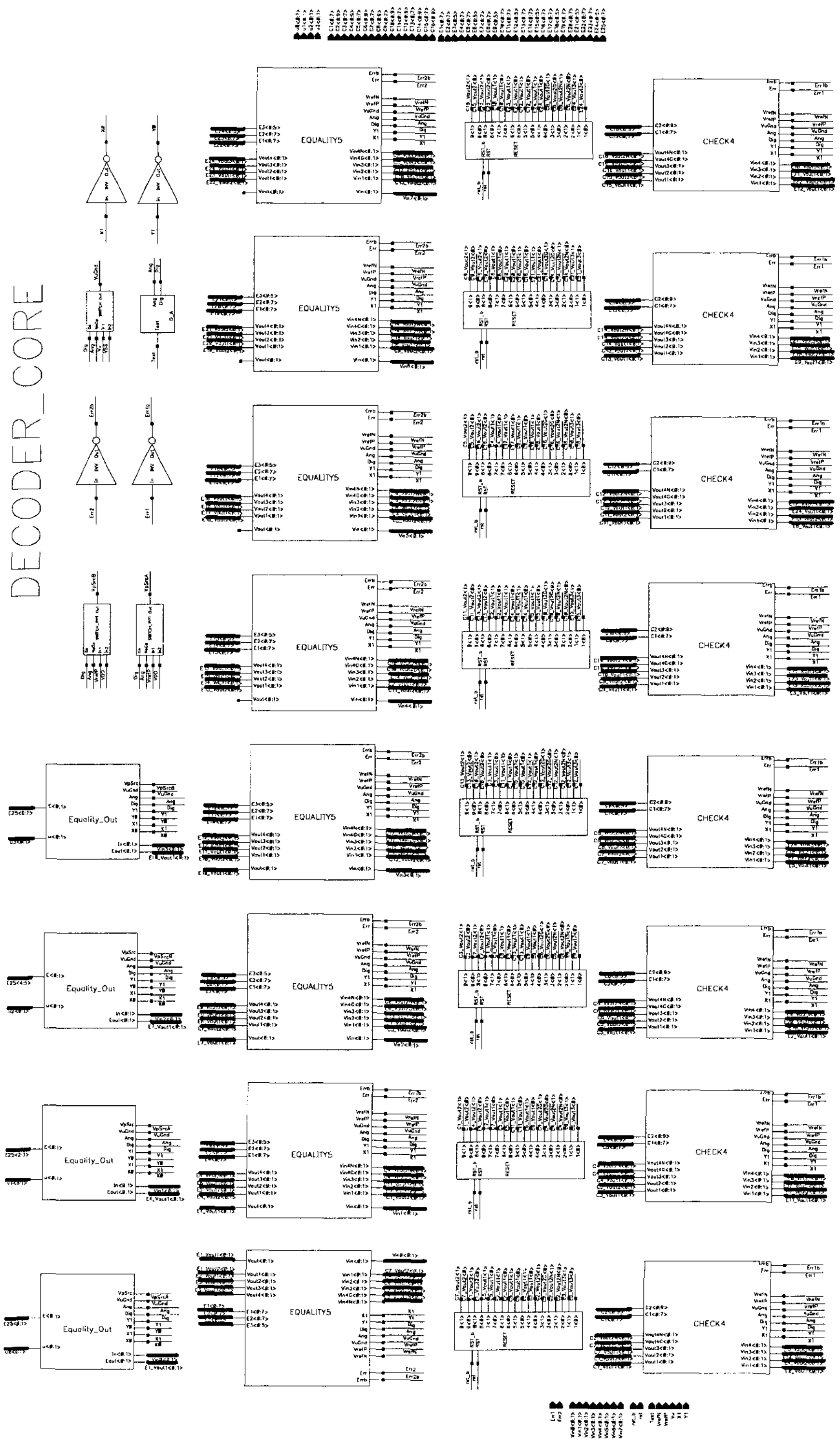


Figure 5.23 The (8, 4) Hamming decoder core

All transistor size = 0.42um/0.18um

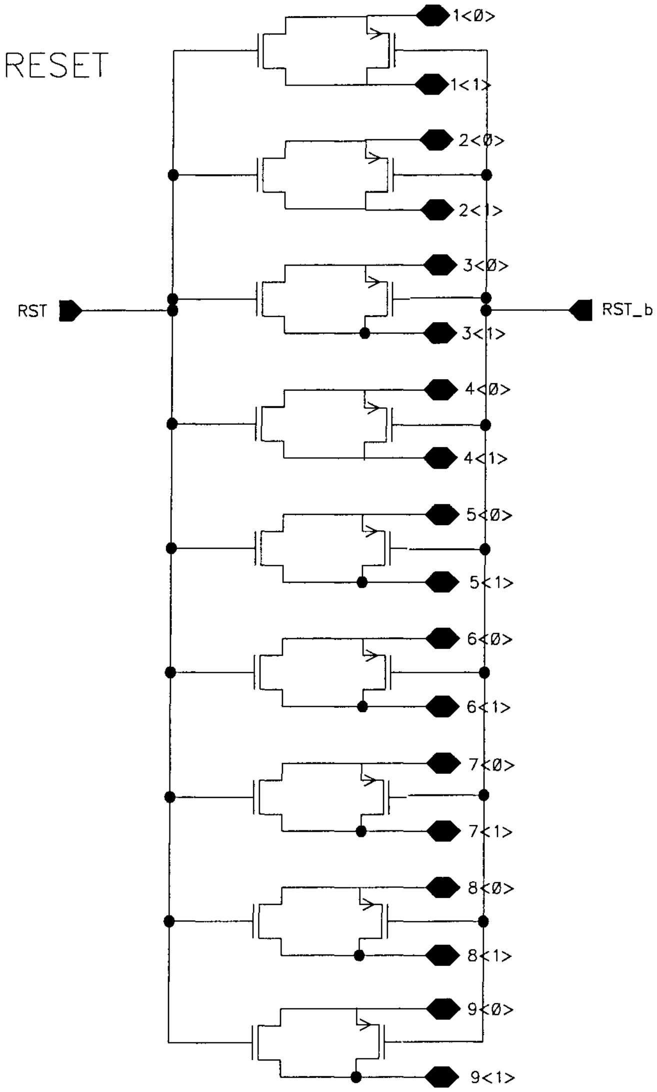


Figure 5.24 Circuit for equalizing probabilities (RESET)

5.2.2 BIST for Decoder Core

The decoder BIST controller was designed in VHDL and the top-level diagram of the BIST is shown in Fig. 5.25. The signals C1, ..., C16 and E1, ..., E25 are the test result groups. They are n-bit vectors where $n = 6$, 8, or 10 depending on the number of diode-connected output transistors in each node. The description of each signal is provided in Table 5.2. In Fig. 5.25, the Show_Node signal is a 6-bit vector used to access test result groups. The test result groups include the CHECK3_1NG, CHECK3_2NG, EQUALITY3, and EQUALITY3_NG nodes and they can be accessed through this address signal. Fig. 5.26 demonstrates how each 3-edge bi-directional node was assigned with a number and Table 5.3 shows how the number was mapped to the 6-bit address signal. The operations of the BIST can best be described by the flow diagram presented in Fig. 5.27. When self-test starts, all unidirectional node circuits in the decoder are separated and the BIST sends the test vectors to these circuits. It then analyzes the circuit response and outputs the test result. A multiplexer was added to the design to access the internal test results.

Table 5.2 Description of the decoder BIST signals

Signal	Description
Clk	Clock signal controlling the timing of the circuit.
Test	Controls the start/stop of the self-test. Test = 1 --> Start, Test = 0 --> Stop.
X	Test pattern sending to the X-input of each unidirectional node circuit.
Y	Test pattern sending to the Y-input of each unidirectional node circuit.
C1-16, E1-25	Circuit response from the decoder core
Finish	Indicates the end of the self-test. Finish = 1 --> Test Ended.
Good_Core	Indicates if the decoder is good. Good_Core = 1 --> Decoder is good, else decoder is faulty.
Show_Node	Address signal used to select which three-edge node is accessed for checking the test result.
Good_Node	The test result of the selected three-edge node. Good_Node = 1 --> Node is good, else node is faulty.

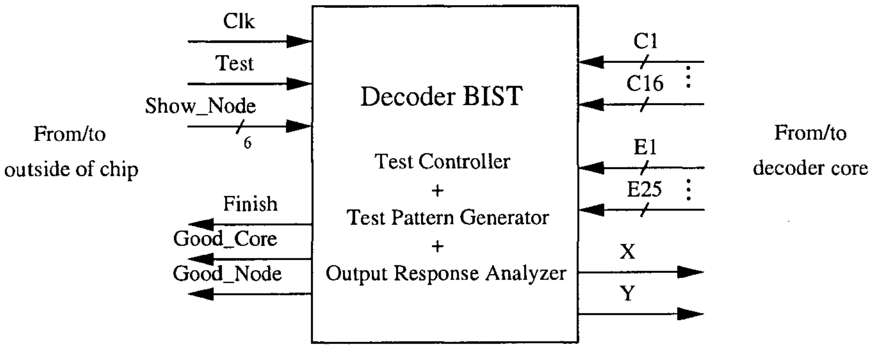


Figure 5.25 Top-level diagram of the decoder BIST

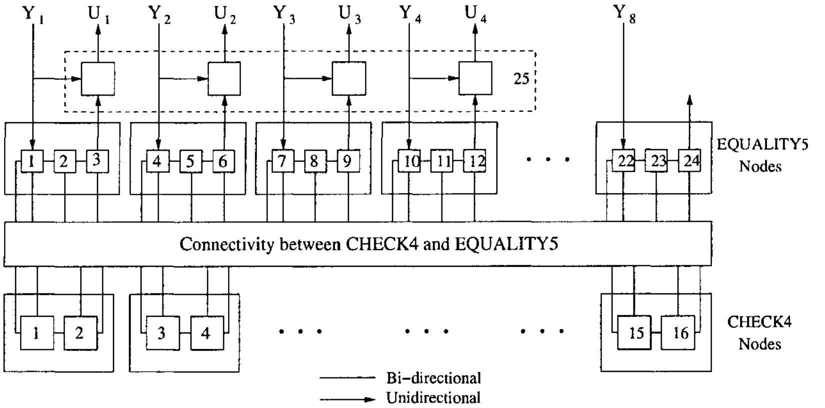


Figure 5.26 Three-edge nodes assigned with numbers

Table 5.3 Mapping from node number to address signal

Show_Node	Check Node No.	Show_Node	Equality Node No.	Show_Node	Equality Node No.
00 0000	C1	10 0000	E1	11 0000	E17
00 0001	C2	10 0001	E2	11 0001	E18
00 0010	C3	10 0010	E3	11 0010	E19
00 0011	C4	10 0011	E4	11 0011	E20
00 0100	C5	10 0100	E5	11 0100	E21
00 0101	C6	10 0101	E6	11 0101	E22
00 0110	C7	10 0110	E7	11 0110	E23
00 0111	C8	10 0111	E8	11 0111	E24
00 1000	C9	10 1000	E9	11 1000	E25
00 1001	C10	10 1001	E10		
00 1010	C11	10 1010	E11		
00 1011	C12	10 1011	E12		
00 1100	C13	10 1100	E13		
00 1101	C14	10 1101	E14		
00 1110	C15	10 1110	E15		
00 1111	C16	10 1111	E16		

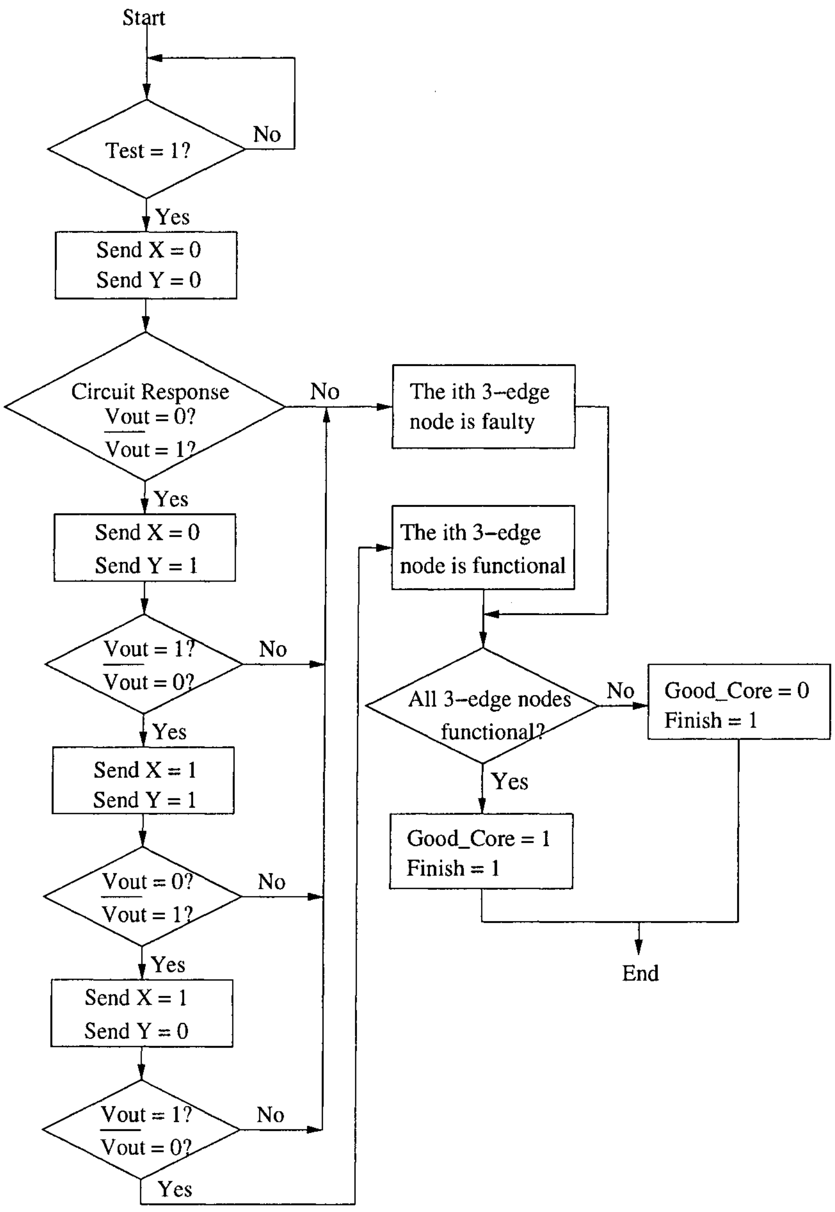


Figure 5.27 Flow diagram of decoder BIST

5.2.3 Universal I/O Interface

The I/O interface used for our (8, 4) Hamming decoder was previously designed by Yu *et al.* in [62, 63] though in a different technology. Fig. 5.28 shows the structure of the I/O interface. The input interface contains two chains of eight sample-and-hold (S/H) cells, one for V_{in} and one for V_{ref}. The cells are used to convert the serial input into a set of parallel inputs. Initially, the LLR voltages are stored into the first capacitor of the S/H cells. After an entire codeword is received, the stored data are passed in parallel to the second capacitor for holding. Meanwhile, the first capacitor starts to sample the next incoming codeword. The held LLR voltages are then converted into probability currents/voltages and are passed to the decoder core for analog decoding. The core then generates pairs of soft output currents representing the probabilities that the received bits were 1s or 0s. Finally, the set of comparators compares the two probabilities, makes final hard decisions and latches all outputs at once. Besides having parallel outputs, shift registers are added to make serial outputs available at DOUT1.

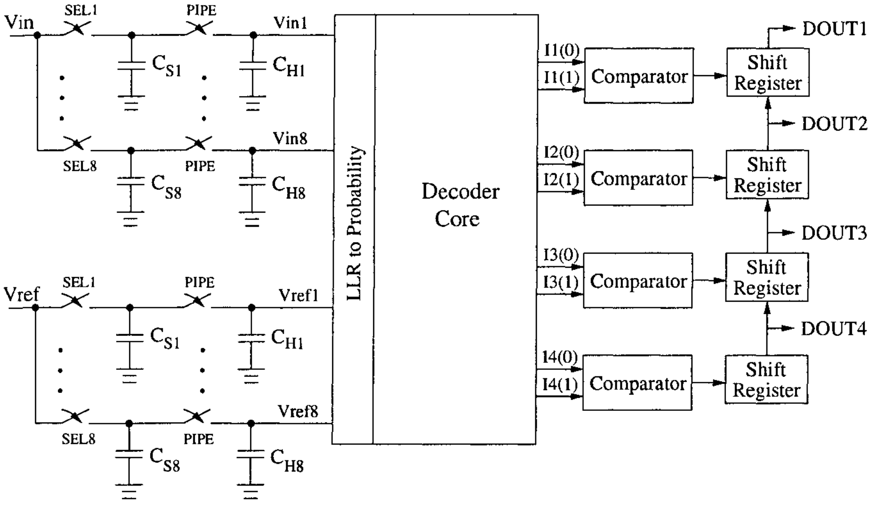


Figure 5.28 Structure of the I/O interface

Fig. 5.29 shows a chain of S/H cells at the transistor level. The operation of the S/H chain can be demonstrated through a timing diagram shown in Fig. 5.30. Initially, we need a FRAME signal at the beginning to reset the interface and to indicate the start of the incoming codeword stream. The reset takes place at the falling-edge of the clock and therefore, the FRAME signal has to stay high until the falling-edge of clock events. After resetting the interface, the first LLR voltage is stored at C_{S1} by enabling the SEL1 signal, and the second LLR voltage is stored at C_{S2} by enabling SEL2 at the next clock cycle, and so on. While the eighth LLR voltage is stored at C_{S8} and SEL8 is high, the SEL8 causes a discharge in the capacitor C_{Hi} for i = 1, ..., 8 in order to clear the old values. At the next clock cycle, the PIPE signal goes high and fetching the stored values from C_{Si} to C_{Hi}. The PIPE signal is also used to reset the decoder core. This process repeats itself for the next incoming codewords.

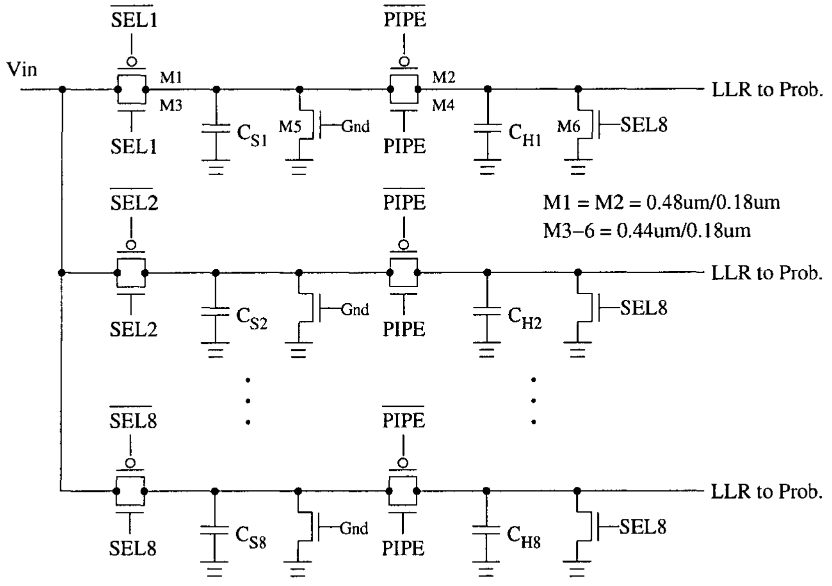


Figure 5.29 Chain of S/H cells at transistor level

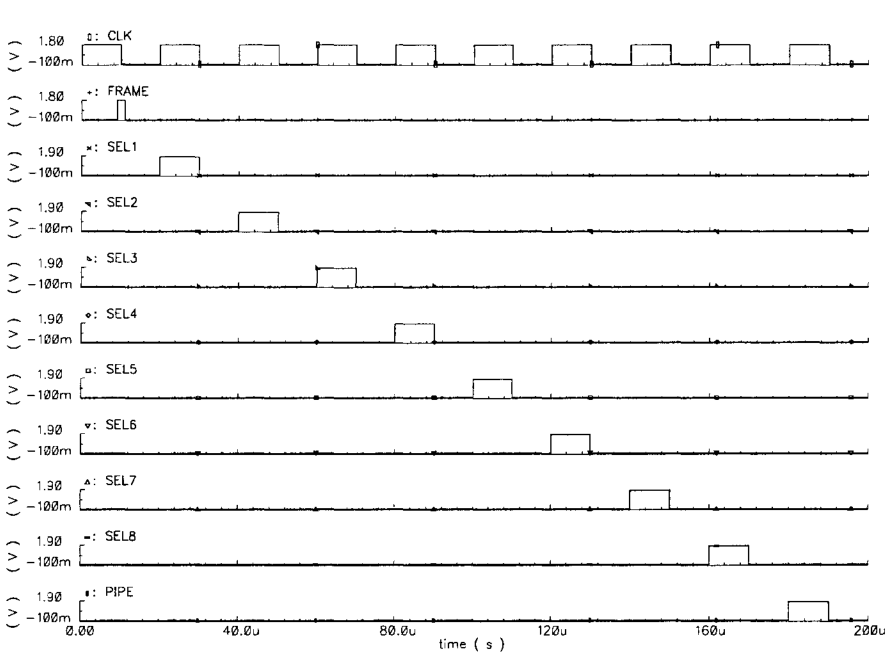


Figure 5.30 Timing diagram of the S/H interface

Fig. 5.31 shows the output interface consisting of four comparators and four shift registers, and Fig. 5.32 shows its timing diagram. The transistor-level schematic of the comparator is shown in Fig. 5.33. The comparator first receives the current inputs, then makes a decision and stores the result onto the set-reset (SR) latch. The decision is then passed into the shift register when SAMPLE is high and the shift register outputs the decoded bit at falling-edge of the same clock cycle. The serial output is provided by shifting the register outputs to DOUT1 during the next three consecutive cycles.

When decoding starts, the first decoded outputs appear seventeen cycles after the FRAME signal and the subsequent decoded outputs arrive every nine cycles. The one extra cycle is due to the PIPE signal for fetching values from C_{Si} to C_{Hi} and to reset the decoder.

In order to perform self-test on the I/O interface, switches were added to function as input MUXs, to pass the signals directly from the input interface to the output interface, and to route the outputs of the comparators back to the I/O BIST.

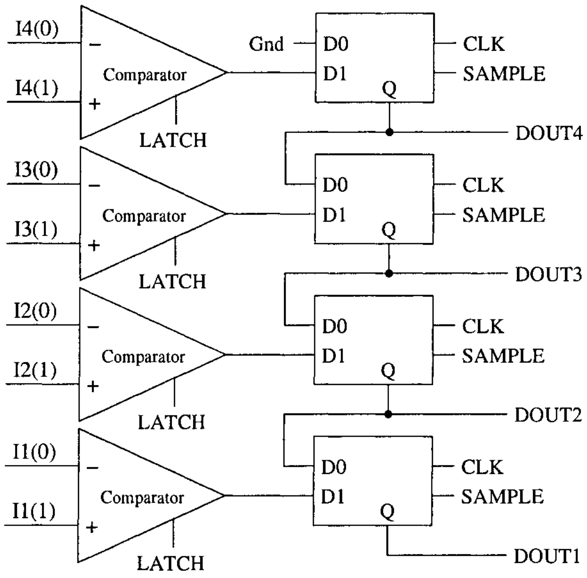


Figure 5.31 Output interface

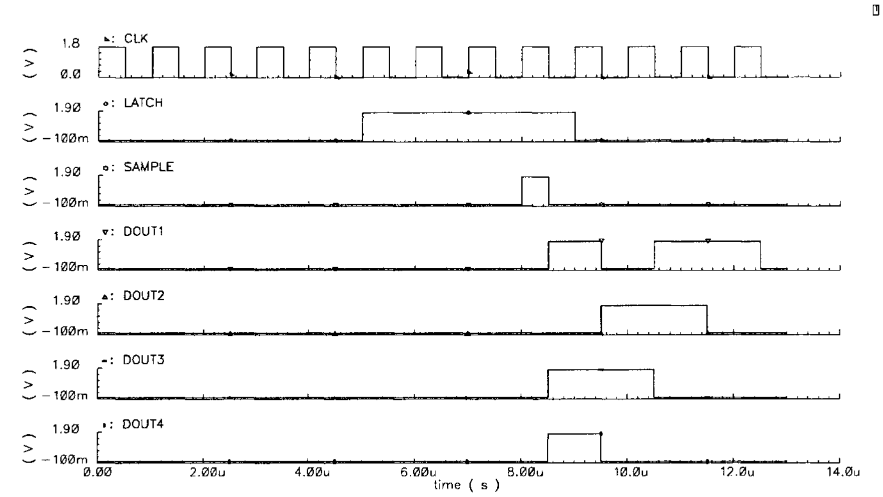


Figure 5.32 Timing diagram of output interface

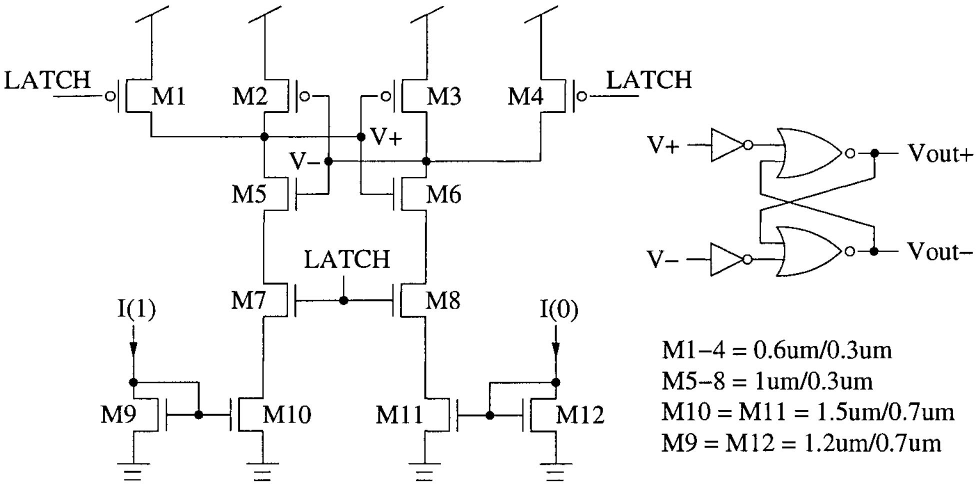


Figure 5.33 Comparator at transistor level

5.2.4 BIST for the I/O Interface

The I/O BIST controller was designed in VHDL and the top-level diagram of the BIST is shown in Fig. 5.34. The description of each signal is provided in Table 5.4. By observing the I/O interface in Fig. 5.28, we can tell that the input interface has eight pairs of outputs but the output interface has only four pairs of inputs. To solve this issue, we can add some MUXs to route either the first or the last four pairs of voltage outputs to the comparators. The BIST can initially test the first four S/H cells of each S/H chain and then test the other four. When self-test starts, the I/O BIST sends the test vectors to the S/H cells serially and the first four pairs of stored bits are then converted to probabilities and passed directly to the comparators. The BIST evaluates the output response from the comparators and then it starts sending test vectors again but this time the last four pairs of stored bits are converted to probabilities and sent to the comparators instead. Comparator outputs are then checked again with the BIST. Finally, the BIST determines whether the I/O interface is functional or faulty. Fig. 5.35 shows the flow diagram used to describe the operations of the I/O BIST.

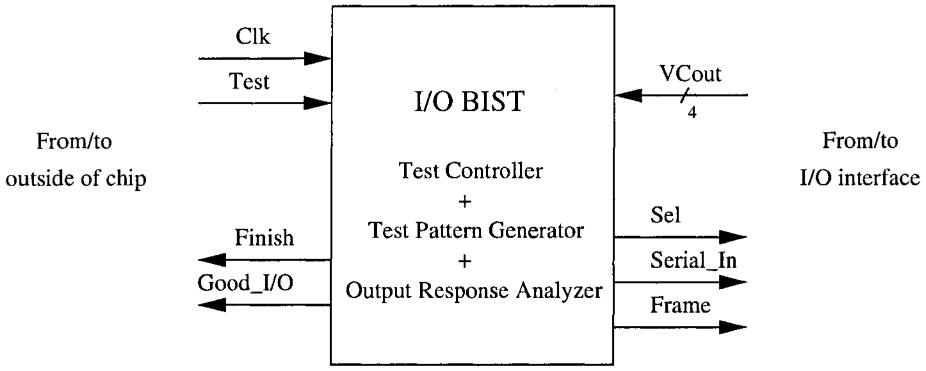


Figure 5.34 Top-level diagram of I/O BIST

Table 5.4 Description of the I/O BIST signals

Signal	Description
Clk	Clock signal controlling the timing of BIST operations.
Test	Controls the start/stop of the self-test. Test = 1 --> Start, Test = 0 --> Stop.
Frame	Frame signal to reset the I/O interface
Serial_In	Test pattern sending to the input interface
VCout	Outputs from the comparators
Finish	Indicates the end of the self-test. Finish = 1 --> Test Ended.
Good_I/O	Indicates if the decoder is good. Good_IO = 1 --> I/O interface is good, or else the interface is faulty.

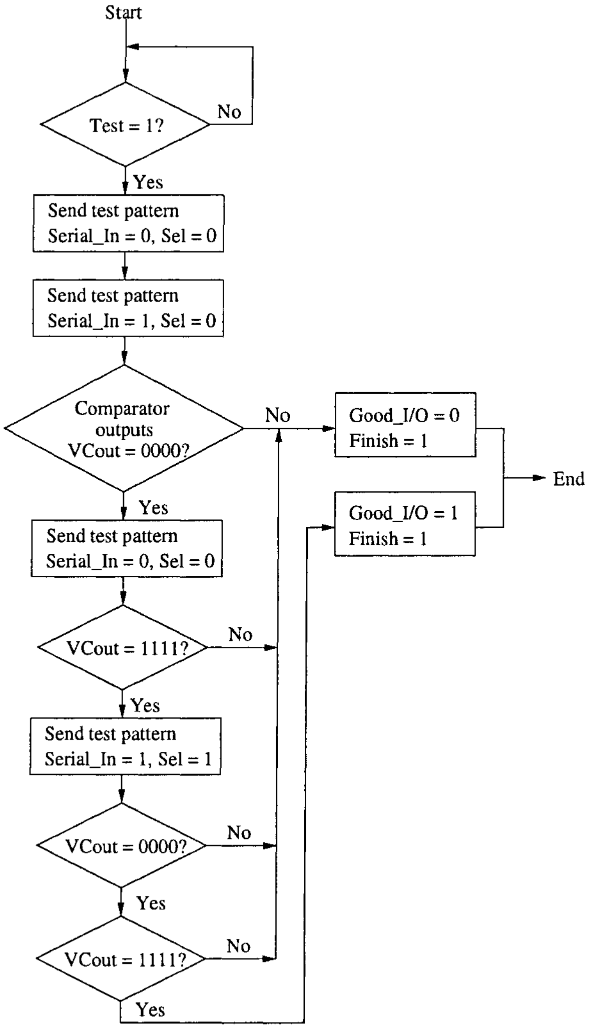


Figure 5.35 Flow diagram of I/O BIST

5.3 Implementation Procedures

In order to build the entire decoder chip, we have divided the chip into four sections: a decoder core, a decoder BIST, an I/O interface, and an I/O BIST. Since the decoder core and I/O interface are analog circuits and the two BIST designs are digital circuits, they were designed using different tools and are finally combined together in Cadence.

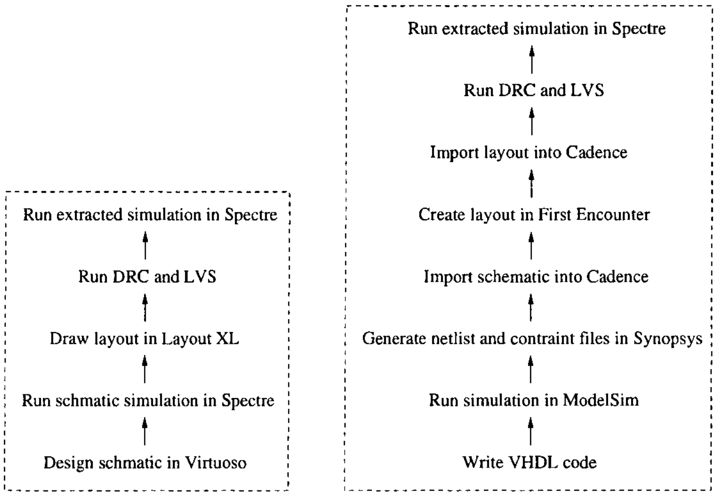
The decoder core was designed in Cadence using a full-custom flow. Since extra rows are added to the generator matrix of the code, it makes the parity check matrix contain an equal number of 1s in the rows and columns. The code becomes more regular and therefore, all nodes have the same architecture and we can reuse the circuitry for one node to build another node. This reduces the complexity of the schematic and layout design process. A bottom-up strategy was used to build the decoder core. We started from the most basic unidirectional node circuit and its schematic was designed in Cadence Virtuoso and simulated using Cadence Spectre. As the function of the circuit was verified, we started drawing the circuit layout in Cadence Layout XL. The shape of the layout was kept as regular as possible. The number of metal layers used was minimized at the lower level such that higher metal layers can be saved for later use at the top level. Power rails were added at the top and bottom of the layout so that overlapping can be done when combining layout into larger blocks.

The layout was then extracted with parasitic capacitance, and LVS and DRC were performed to ensure that the layout matched with the schematic and that no design rule was violated. The layout was then simulated and compared with schematic results. After the basic blocks were built, they were used to integrate larger blocks. The process was repeated until the entire decoder core was built. Fig. 5.36(a) shows the procedures for implementing our analog circuit.

As the I/O interface was used in other decoders designed by our group [39, 54], the schematics and layouts of the interface were provided. Simulations were performed to verify the functionality of the interface. To facilitate self-test, switches were added to the schematics and layouts of the interface and LVS, DRC, and simulations were run to ensure correctness of the design.

The decoder BIST and I/O BIST were both designed in VHDL using the Xilinx ISE tools. First of all, VHDL code was simulated with a VHDL testbench in ModelSim to verify the functionality of the code. We then translated the VHDL file into a netlist, which contains standard cell information and their connectivity. The netlist represents the circuit, which behaves as described in the VHDL code, and it was generated by Synopsys Design Analyzer. The program also generated timing constraint files used for creating the layout. We then created the schematic in Cadence by importing the netlist. Layout was drawn with an automated floor planning and routing tool called First Encounter. First Encounter creates a layout according to the circuit and timing information from the netlist and constraint files. The layout was then imported into Cadence. LVS, DRC, and simulations were run to verify the design. Fig. 5.37 shows the automated I/O BIST layout as an example. The same procedures were used for both the decoder BIST and I/O BIST. Fig. 5.36(b) reveals the procedures of implementing a digital circuit.

Finally, the top-level design was built by combining the decoder core, decoder BIST, I/O interface, and I/O BIST. The top-level layout was done by connecting wires manually in Cadence Layout XL. LVS, DRC, and Spectre simulations were necessary. Then I/O drivers and bond pads were added to the top-level design in order to finish implementing the entire chip. Our chip consists of two decoders: a full decoder and a back-up decoder. The back-up decoder does not contain the output interface and the I/O BIST. The back-up decoder was added into the chip because of the comparator offset problem encountered by other researchers in our group who also used the same I/O interface in their designs [39, 54]. Therefore, without comparators and shift registers, the back-up decoder sends decoded probability currents directly to the chip outputs. External operational amplifiers and comparators are needed if we want to perform verification test on the back-up decoder.


Run extracted simulation in Spectre
Run DRC and LVS
Draw layout in Layout XL
Run schmatic simulation in Spectre
Design schmatic in Virtuoso
Run extracted simulation in Spectre
Run DRC and LVS
Import layout into Cadence
Create layout in First Encounter
Import schematic into Cadence
Generate netlist and contraint files in Synopsys
Run simulation in ModelSim
Write VHDL code

Analog Circuit Design
(a)

Digital Circuit Design
(b)

Figure 5.36 Design procedures of (a) analog circuits (b) digital circuits

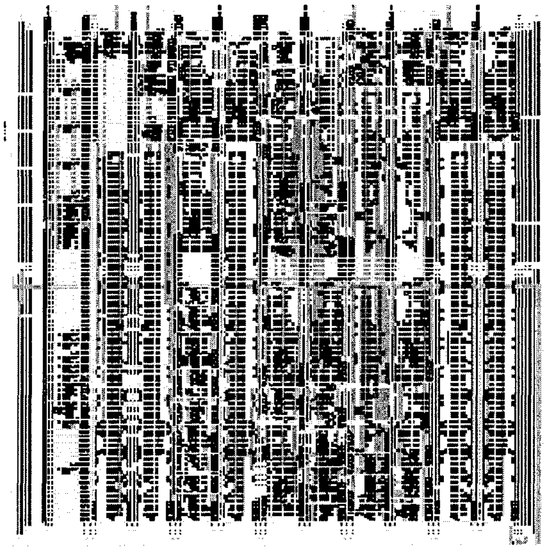


Figure 5.37 Automated layout of the I/O BIST

5.4 Simulations

5.4.1 Simulations on Self Test

The digital BISTs were designed in VHDL and then verified in ModelSim. The VHDL code was simulated by feeding appropriate input vectors from a testbench file and monitoring the outputs. Two conditions were checked: the BIST testing a DUT with faults and the BIST testing a DUT with no fault.

A clock cycle of 40 ns was used for the ModelSim simulations. Fig. 5.38 shows the waveform of the decoder BIST testing a DUT with no fault. In the waveform, self-test started at 60 ns and the decoder BIST started sending test vectors at 100 ns. Data was sent at the rising-edge of the clock and responses were checked at the falling-edge of the clock. The test ended at 300 ns and the decoder BIST indicated that the DUT is functional. The Show_Node signal checked the internal test results of nodes C1 to C3, E1 to E3, and E25 and they are all functional. Fig. 5.39 shows the waveform of the decoder BIST testing a DUT with faults. The input test vectors were set up such that the DUT contained faults in nodes C5, C10, C15, E10, E20, and E25. At the end of the test, the decoder BIST indicated that the DUT was faulty and showed that nodes C1, C3, E1, E5 are good and nodes C5, C10, C15, E10, E20, E25 are bad.

Fig. 5.40(a) shows the waveform of the I/O BIST testing a DUT with no fault. Self-test tested at 60 ns and the I/O BIST sent the serial test pattern at 100 ns. The first set of DUT responses was checked at 700 ns and the second, third, and fourth set of responses were checked at 1.1 μs, 1.5 μs, and 1.86 μs respectively. As long as the response is correct, the signal Good_IO stays high. The self-test finishes at 1.86 μs and the BIST indicates that the DUT is functional. Fig. 5.40(b) shows the waveform of the I/O BIST testing a DUT with faults. This time the third set of DUT responses is incorrect. The BIST detected the errors and set the Good_IO to low and stopped the test.

After importing the layout of the BISTs and connecting them to the decoder core and I/O interface, the full decoder chip in test mode was simulated using Cadence Spectre. A clock cycle of 1 μs was used. We first simulated the decoder chip performing the self-test and no errors were injected into the decoder core. Fig. 5.41 shows the simulation results of the functional decoder chip. The results are the same as ones obtained from ModelSim in Fig. 5.38 and Fig. 5.40(a). Then we enabled a signal called ERR1 so that there were faults injected into all the CHECK3_1NG nodes, which means nodes C1, C3, ..., C15 in Table 5.3 were faulty. Then we enabled another signal called ERR2 to make all the EQUALITY3 nodes (nodes E3, E6, ..., E24) contain faults. The full decoder was then simulated again but this time with faults injected into the decoder core. The simulation results are shown in Fig. 5.42. The fault indicated that the I/O interface was good and the decoder core was faulty. Some internal nodes were accessed and nodes C2, C16, E1, E8, E25 were good and nodes C1, C15, E3, E9, E24 were bad.

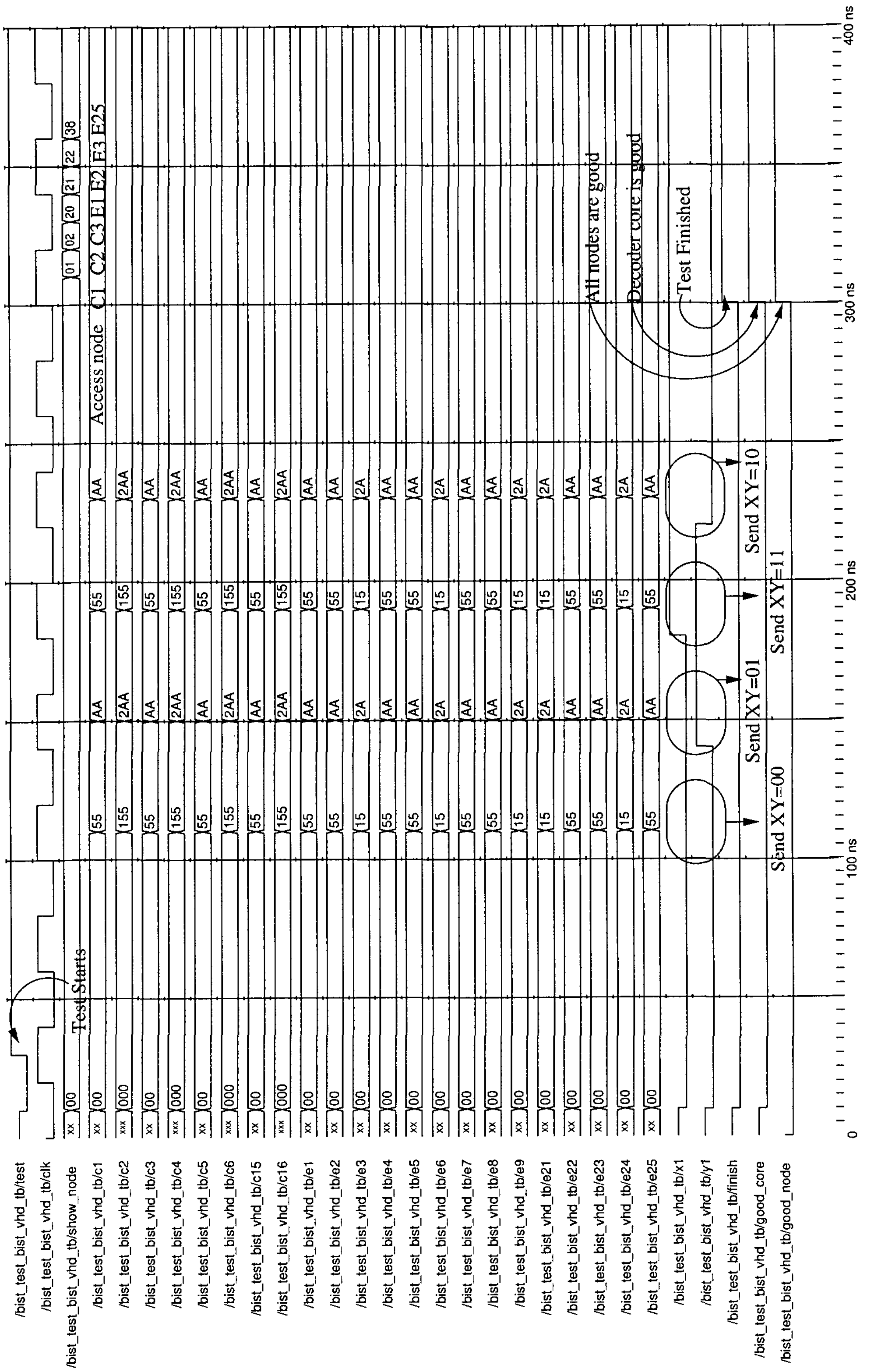


Figure 5.38 Waveform of decoder BIST detecting no fault in the decoder and showing that nodes C1, C2, C3, E1, E2, and E3 are good

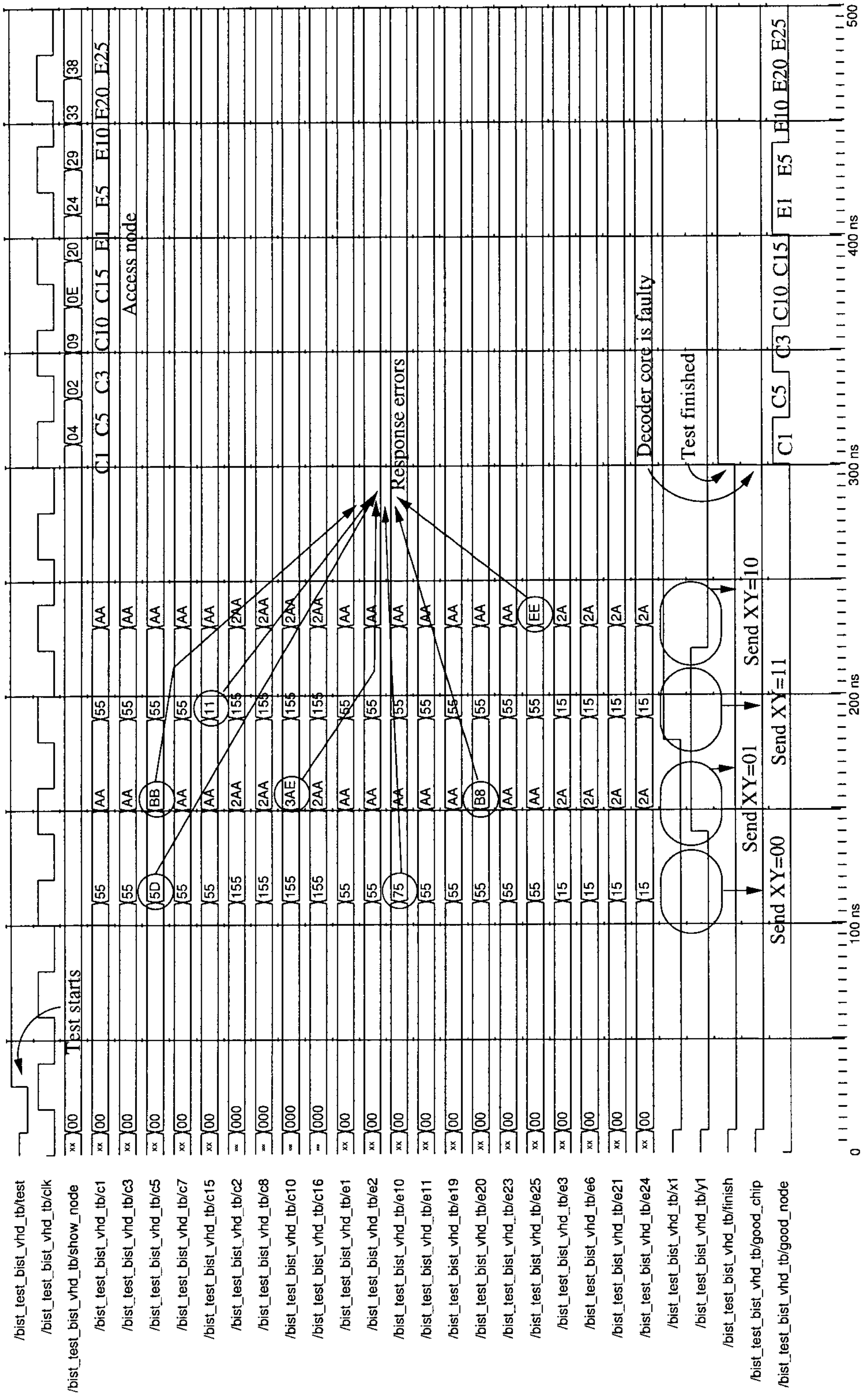


Figure 5.39 Waveform of decoder BIST detecting faults in nodes C5, C10, C15, E10, E20, and E25

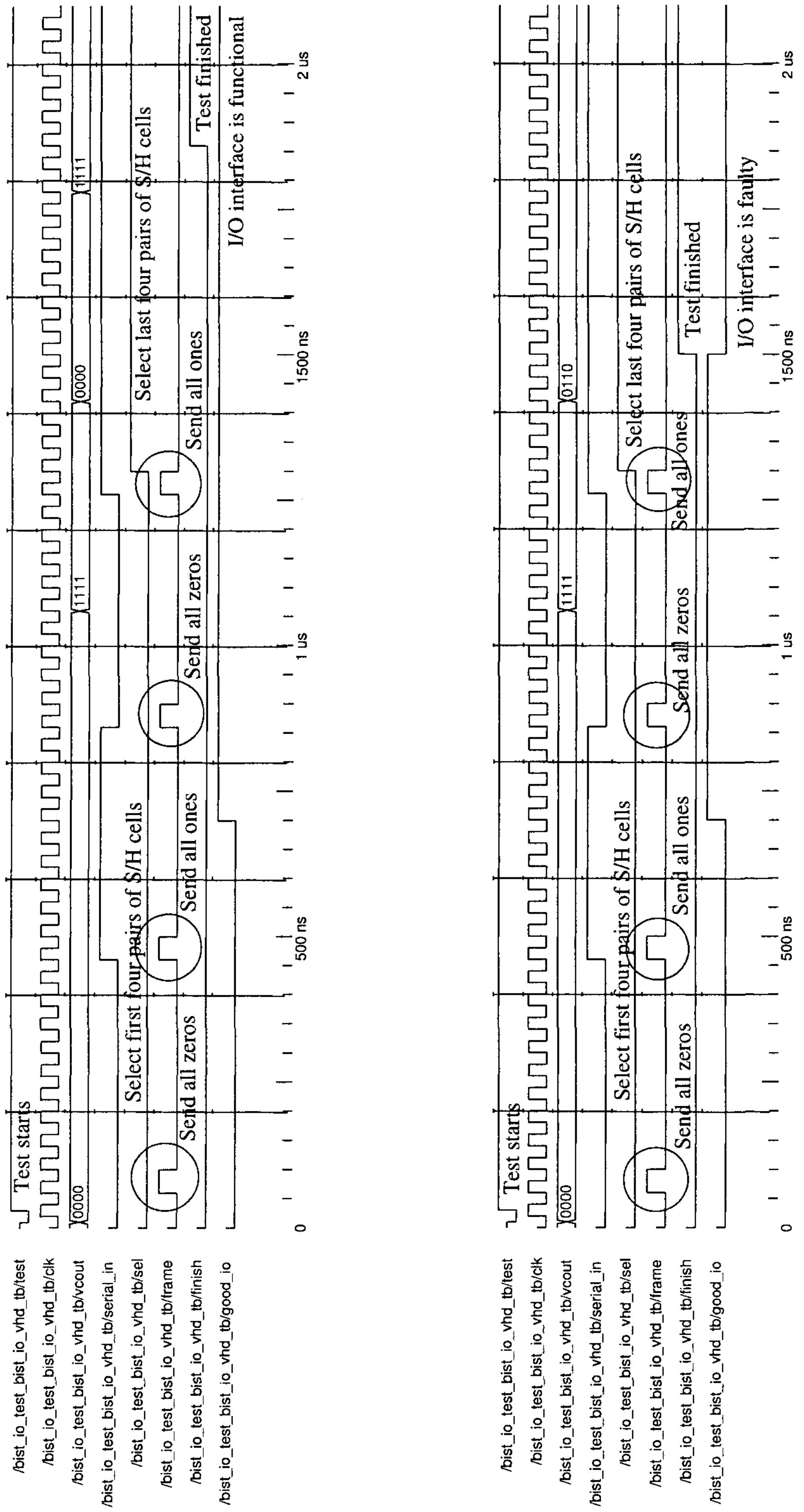


Figure 5.40 Waveform of I/O BIST: (a) detecting no fault in the interface (b) detecting faults in the third set of responses receiving from the interface

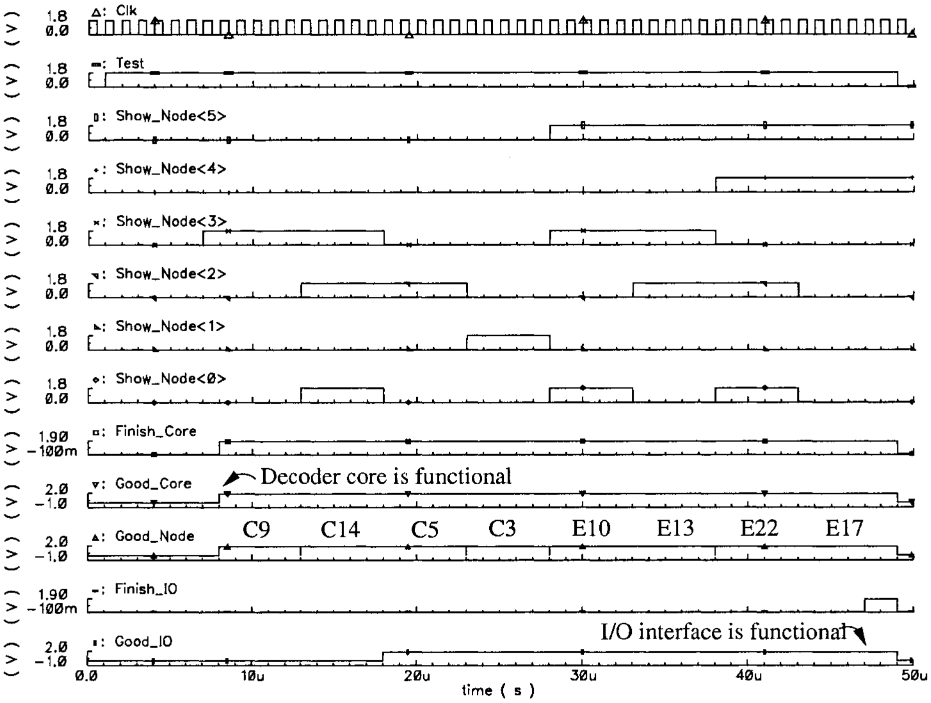


Figure 5.41 Simulation results of full functional decoder chip in test mode showing that the decoder core, I/O interface, and all internal nodes are functional

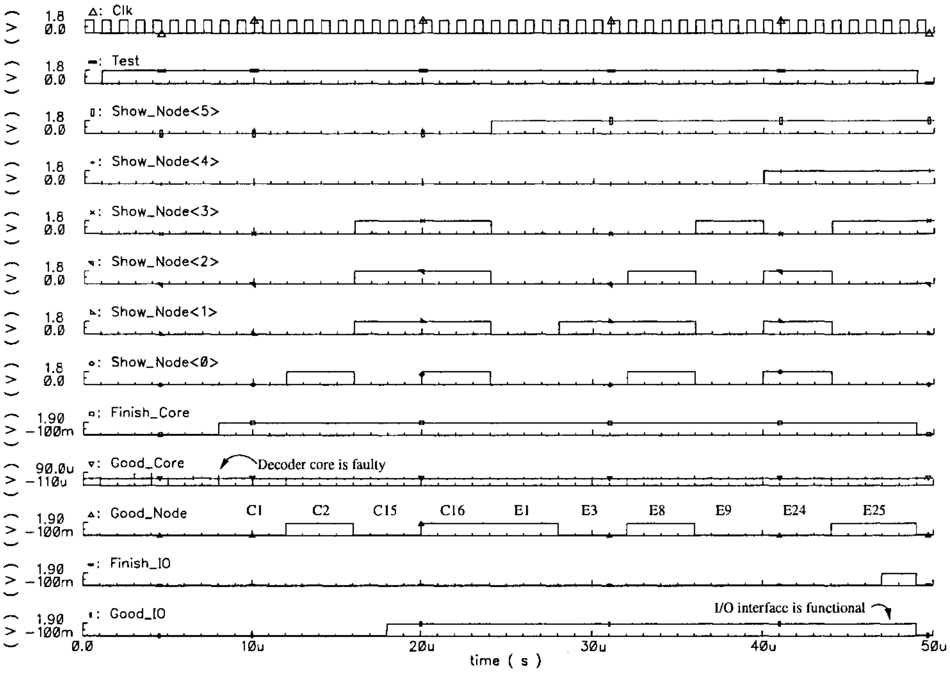


Figure 5.42 Simulation results of full faulty decoder chip in test mode showing that the decoder core is faulty, I/O interface is functional, and some of the internal nodes are bad

5.4.2 Simulations of Analog Decoder

The analog decoding behavior of the decoder core was simulated in Spectre. Probability currents were injected into the decoder inputs and decoded probability currents were observed. The decoder core used a supply voltage of V_{DD} = 1.8 V and unit bias current of I_U = 10 nA. Such a small current was used since we want to test the robustness of the decoder core and carefully examine how the error bit was being corrected. Table 5.5 shows the probability values of the currents being injected into the decoder core. The probability currents represent a received codeword from the channel, which is 00**0**01101. The received codeword contained an error at the third bit which is in **bold**. The output currents are shown in Fig. 5.43. The output currents u1<0> and u1<1> represent the probabilities that the first received bit was a 1 or 0, u2<0> and u2<1> represent the probabilities of the second bit, and so on. We can observe from Fig. 5.43 that the third bit represented by u3<0> and u3<1> was corrected from 0 to 1. Other currents also settled to an equilibrium state. The simulation could not give us an accurate estimation of how fast the decoder can run. In analog decoding, time required for the currents to settle depends on the input probabilities, the error pattern, and the unit bias current.

Table 5.5 Probability values of input currents used for simulating the decoder core

Bit no.	p(0)	I_0 (nA)	p(1)	I_1 (nA)
1	0.9	9	0.1	1
2	0.8	8	0.2	2
3	0.6	6	0.4	4
4	0.7	7	0.3	3
5	0.1	1	0.9	9
6	0.2	2	0.8	8
7	0.7	7	0.3	3
8	0.4	4	0.6	6

After connecting the I/O interface to the decoder core, an analog decoding simulation was run with the full decoder. The simulation results are shown in Fig. 5.44. The serial inputs to the full decoder have probability of $p(1)$ equals 0.8 or 0.2 depending on whether the incoming bit is a 1 or 0 and $p(0)$ = 1- $p(1)$. Three codewords are shifted serially into the decoder. The first decoded outputs appear on the falling edge of the 17th cycle and they are presented in parallel through DOUT1 to DOUT4 or shifted serially through DOUT1. The decoded outputs appear every nine cycles. The extra cycle is due to the I/O pipeline and decoder reset. A bias current of 1 μA and a clock rate of 1 MHz were used giving a decoding rate of 444 kbps. The three codewords being simulated were 01**0**10100, 0**1**010111, and **0**0001011 where the error bits are shown in **bold**. The decoded output bits were 0111, 0001, and 1000 as shown in Fig. 5.44.

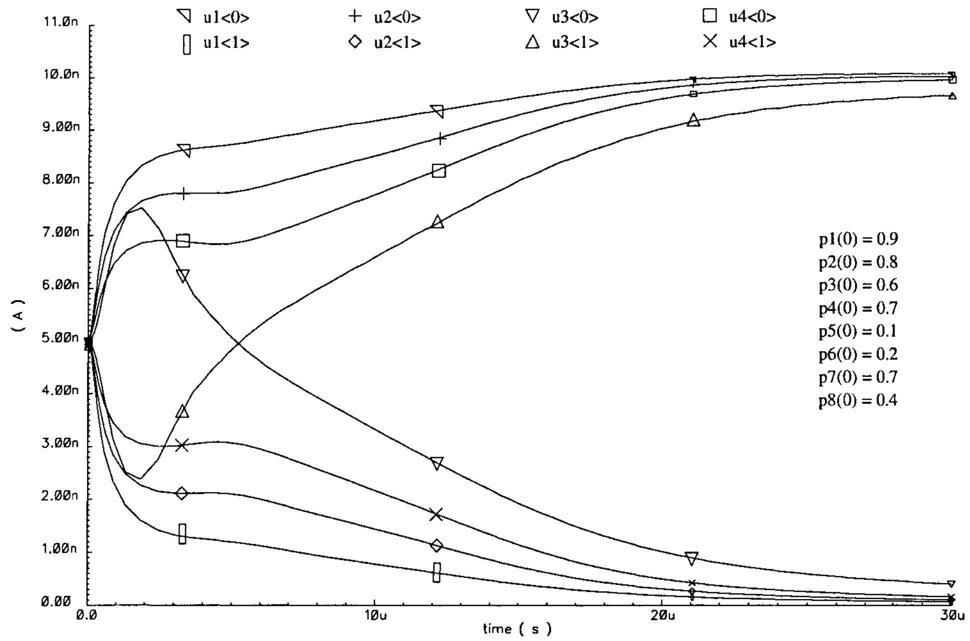


Figure 5.43 Analog decoding simulation of the decoder core with I_U = 10 nA showing an error correction on bit3

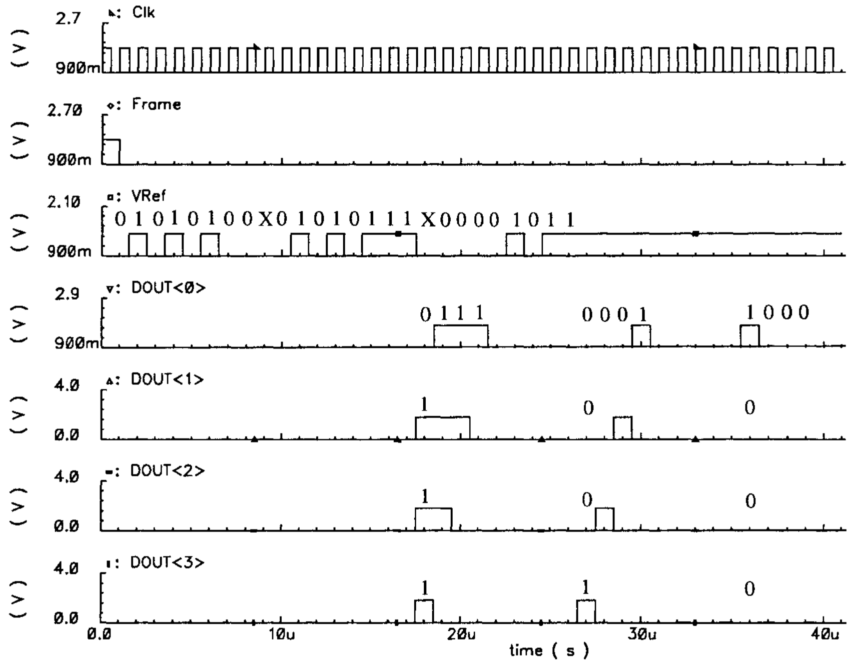


Figure 5.44 Simulation of the full decoder with $I_U = 1$ μA showing the decoding process of three codewords

5.5 Chapter Summary

In this chapter, a digital BIST scheme for analog iterative decoders using the sum-product algorithm was proposed. The proposed BIST scheme was applied on an (8, 4) Hamming decoder. Implementation details of the Hamming decoder chip including the decoder core, decoder BIST, I/O interface, and I/O BIST were given and simulations in ModelSim and Cadence Spectre were provided. In the next chapter, we will present the IC test results.

Chapter 6

Testing

This chapter describes the testing of the decoder chip. Two different types of tests were performed to verify the chip: a self-test and a BER test. We will describe the general test setup and equipment and give details about the self-test and BER test along with test results.

6.1 Fabrication of Prototype

The (8, 4) Hamming decoder chip with BIST was designed and fabricated in a 1P6M TSMC 0.18 μm process with assistance from CMC Microsystems. Fig. 6.1 shows a die photo of the decoder chip.

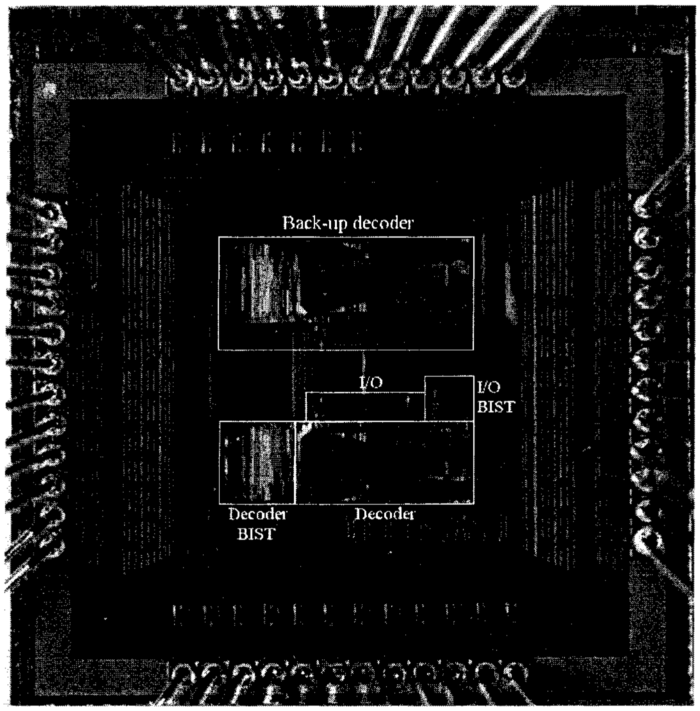


Figure 6.1 Die photo of (8, 4) Hamming decoder with BIST (ICFAAMY1)

6.2 General Test Setup and Equipment

The general test set-up for testing the decoder chip is shown in Fig. 6.2. The test set-up contains a PC with RedHat 9 Linux platform that controls the prototype test procedures, collects test results and displays results on screen. The Keithley 236 Source Measure Unit is used to generate small unit bias currents in the range of nA to µA. The oscilloscope and multi-meter are used for probing signals. The decoder chip is located on a PCB board connected to a test support board and the test support board is connected to an FPGA as shown in Fig. 6.3. Fig. 6.4 shows how the testing PC, FPGA, test support board and PCB board communicate with each other. The test support board plays an important role since it provides communications between the PCB and FPGA through a USB interface device and also between FPGA and DUT through some electronic devices (e.g. buffer, DAC, amplifier, etc.).

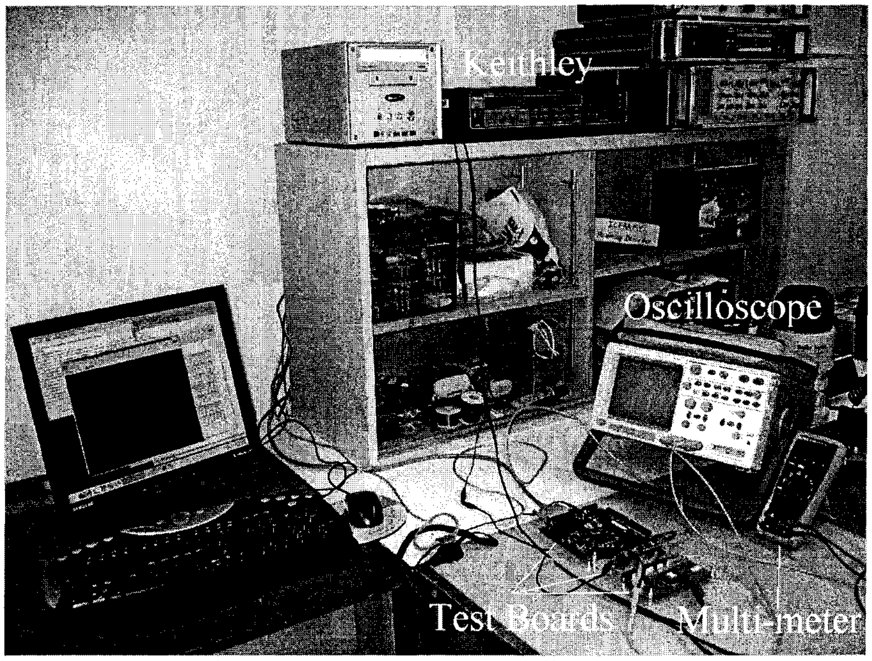


Figure 6.2 Picture of test set-up: PC, Keithley unit, FPGA board, test support board, PCB, multi-meter, and oscilloscope

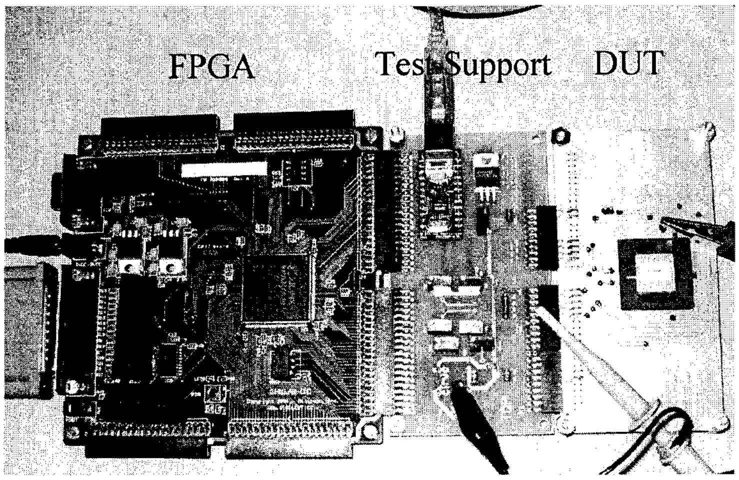


Figure 6.3 Picture of FPGA, Test Support, and DUT boards

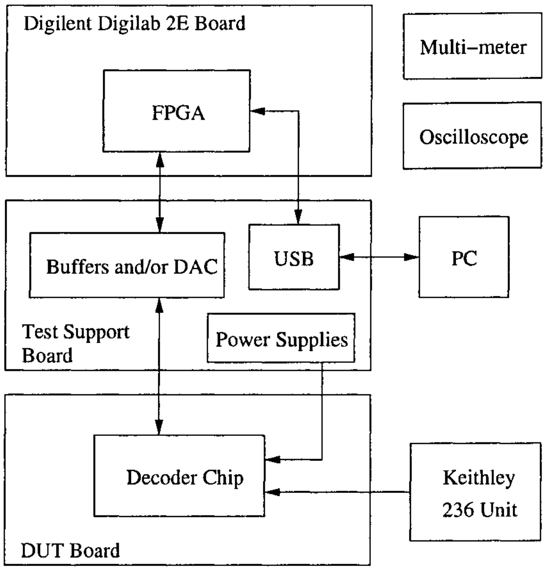


Figure 6.4 General test-setup block diagram for ICFAAMY1

6.3 Self Test

Since the (8, 4) Hamming decoder chip was implemented with built-in test circuits, we can detect any catastrophic faults within the chip easily and we only need a simple test program and VHDL program to control the test flow. The PC first sends a start signal to the FPGA through a USB interface and the FPGA then enables the TEST signal of the decoder chip to start self-test. The FPGA waits for the self-test to finish and reads the test results from the chip. Then the test results are sent from the FPGA back to the PC and the results are displayed on the PC monitor.

6.3.1 Test Program on PC

The PC test program for the self-test is a very simple program written in C++. It sends a start signal to the FPGA and then waits for the test results to come back and displays them on the screen. The test program shows on the screen whether the decoder core and I/O interface are functional and it also indicates the result of each internal 3-edge bi-directional node including C1 to C16 and E1 to E25.

6.3.2 FPGA Controller Board

The FPGA controller board is a Digilent Digilab 2E board which contains a Xilinx Spartan 2E FPGA chip with 200K gates [66]. The test controller is written in VHDL, and is designed in a FSM fashion. The FPGA controller basically performs four main tasks: receiving a control signal from PC, applying a control signal to decoder, receiving test results from the decoder, and sending test results back to the PC. Since the FPGA provides a clock rate of 50 MHz and the decoder chip may not be able to perform self-test at such a high speed, a slower clock is created in the VHDL code to control the self-test of the decoder chip. Moreover, the USB interface can send only one byte of data at a time. Therefore, the FPGA first sends the test results of the internal nodes C1 to C8, followed by C9 to C16, E1 to E8, E9 to 16, E17 to E24, then finally E25, core results and I/O interface results.

6.3.3 Test Support Board

The test support board is located between the FPGA board and DUT board as shown in Fig. 6.3. The role of the board is to provide USB communications between the FPGA board and PC and to provide voltage supplies for the decoder. The support board can provide up to four variable voltages to the DUT. The voltages are generated by an amplifier AD8544 [64] connected as a voltage follower and each of them can be adjusted by a potentiometer ranging from $0.02V_{DD}$ to $0.98V_{DD}$ with small currents in mA. For decoder self test, only one voltage is needed which is 1.8 V used for operating the core circuit of the decoder chip. An adjustable voltage supply which supports larger currents is also provided by an LM317 voltage regulator set to 3.3 V in order to drive the I/O ring of the decoder chip. Digital signals are also buffered using SN74LV125A digital driver chips [68]. Through the buffers, twelve digital signals can be sent from the DUT to the FPGA and vice versa. To communicate between the FPGA board and the PC, the test support board includes a DLP245M USB-to-FIFO converter device [67]. The FPGA controller uses four handshaking signals to control the read and write operations of the USB device. The device contains 384 bytes of FIFO transmit buffer and 128 bytes of FIFO receive buffer.

6.3.4 DUT Board

The decoder chip has a pin grid array (PGA) package with 68 pins and a socket for this package type is added to the DUT board such that the chip can become removable. Power supply and ground lines are routed to the test support board. Switches are added for the ERR1 and ERR2 signals so that we can turn on/off the error injection option of the decoder chip.

6.3.5 Test Procedures

Performing the self-test only requires the 3 steps listed below:

1. Adjust the I/O ring supply to 3.3 V and core supply to 1.8 V. Verify the voltages and ground on the DUT board by probing the signals using multimeter.

2. Program FPGA device with the appropriate .bit file generated from the VHDL code.

3. Run the C++ test program and observe results on screen.

6.3.6 Test Results

The decoder chip is capable of performing self-test at a clock rate of 12.5 MHz. With this clock rate, the self-test has a run time of 5 μs. The results of the first self test are copied from the PC and listed below:

```
1 byte read,    Node C8 downto C1 is: 11111111
1 byte read,    Node C16 downto C9 is: 11111111
1 byte read,    Node E8 downto E1 is: 11111111
1 byte read,    Node E16 downto E9 is: 11111111
1 byte read,    Node E24 downto E17 is: 11111111
1 byte read,    Good_Core, Good_IO, and Node E25 is: 101
```

In the first test, the ERR1 and ERR2 signals were turned off so that no faults were injected into the chip. The result shows that Good_Core is 1, which means the decoder core is good. The test results of all the internal 3-edge bi-directional nodes are 1; therefore, all the nodes are good. However, since Good_IO equals 0, it means that the I/O interface is bad. It actually makes sense that the I/O interface is bad since in [39] and [54], Winstead and Nguyen both encountered a charge leakage problem on the S/H capacitors and comparator DC offset problems. Since the I/O BIST does not tell which bits of the decoder outputs are bad, out of the four decoder output bits, some bits may work fine and some may not.

We then ran the same test but this time we inserted the ERR1 signal and made nodes C1, C3, ..., C15 become faulty. The test results are shown below:

```
1 byte read,    Node C8 downto C1 is: 10101010
1 byte read,    Node C16 downto C9 is: 10101010
1 byte read,    Node E8 downto E1 is: 11111111
1 byte read,    Node E16 downto E9 is: 11111111
1 byte read,    Node E24 downto E17 is: 11111111
1 byte read,    Good_Core, Good_IO, and Node E25 is: 001
```

The decoder BIST is capable of detecting all the faults. We then repeated the test and inserted the ERR2 signal instead to make E3, E6, ..., E24 become faulty. The test results are:

```
1 byte read,    Node C8 downto C1 is: 11111111
1 byte read,    Node C16 downto C9 is: 11111111
1 byte read,    Node E8 downto E1 is: 11011011
1 byte read,    Node E16 downto E9 is: 10110110
1 byte read,    Node E24 downto E17 is: 01101101
```

```
1 byte read,     Good_Core, Good_IO, and Node E25 is: 001
```

The decoder BIST is also able to detect the faults. Finally, we simultaneously enabled ERR1 and ERR2 and ran the test again:

```
1 byte read,     Node C8 downto C1 is: 10101010
1 byte read,     Node C16 downto C9 is: 10101010
1 byte read,     Node E8 downto E1 is: 11011011
1 byte read,     Node E16 downto E9 is: 10110110
1 byte read,     Node E24 downto E17 is: 01101101
1 byte read,     Good_Core, Good_IO, and Node E25 is: 001
```

The decoder successfully detected all faults. Therefore, without running any long BER test, we can easily find out whether a chip is defective or not.

6.4 BER Test

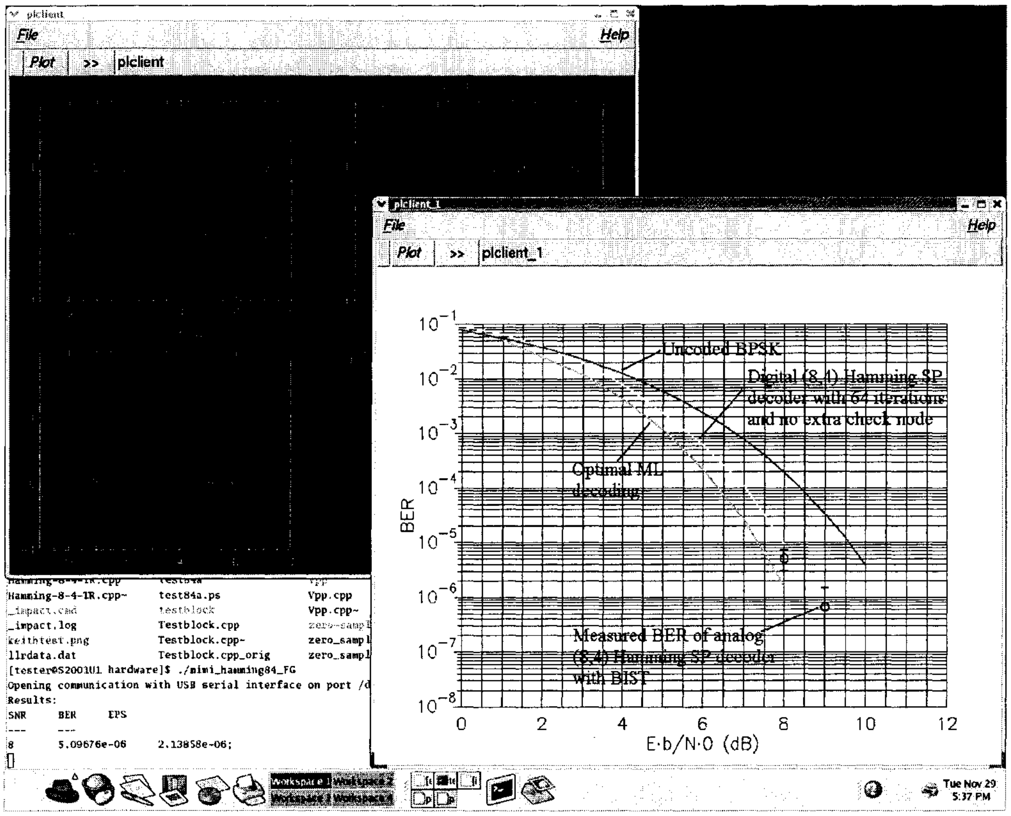
In the last section, we showed that the (8, 4) Hamming decoder is able to test itself and discover any catastrophic faults. Based on the discussion provided in Section 4.5, the performance of a decoder is verified as long as there is no stuck-at fault and a small mismatch variance is met. Therefore, it should not be necessary to perform any lengthy statistical test during the manufacturing process when the decoder chip can perform self-test. However, we are testing the design prototype and this prototype is the very first analog decoder design that contains built-in test circuitry. It is necessary for us to examine how the extra circuitry affects the BER performance of the analog decoder.

The BER test set-up was designed by Winstead and Nguyen in [39] and [54]. The PC first generates channel samples with Gaussian noise and sends them to the FPGA. The FPGA collects the samples and applies them to the DAC to obtain analog channel samples. The analog samples are clocked into the decoder chip and time is provided to allow decoding. Once decoding is finished, the decoded bits are captured by the FPGA controller and sent back to the PC. Finally the PC determines whether the decoded bits are correct or not and calculates the BER.

6.4.1 Test Program on PC

The BER test program is responsible for generating coded bits and Gaussian noise samples, sending codewords with noise to and receiving decoded bits from the FPGA through the USB interface, counting errors, and plotting the results. Fig. 6.5 shows a screen shot of the test results displayed by the test program. The task sequence of the program is listed below [39]:

1. Plot BER curves of the ideal uncoded BPSK, optimal ML decoding, and digital (8, 4) Hamming sum-product decoder.
2. Send information including start of test, test speed, and block length of code.
3. Generate a random source word and encode it using the generator matrix.
4. Modulate codeword in BPSK and add AWGN according to SNR.
5. Convert AWGN modulated symbols to LLR.
6. Convert LLR to DACCODE values.
7. Send DACCODE values to FPGA and wait for decoding results.
8. Receive decoded bits from FPGA and compare with the source word to determine bit errors.
9. Update BER and plot data on graphs.
10. Repeat step 3 to 12 until required number of errors met for each SNR point.



```
Hamming-8-4-1R.cpp~     test84a.ps            Vpp.cpp
_impact.cmd             testblock             Vpp.cpp~
_impact.log             Testblock.cpp         zero-sampl
keithtest.png           Testblock.cpp~        zero_sampl
llrdata.dat             Testblock.cpp_orig    zero_sampl
[tester@S2001U1 hardware]$ ./mini_hamming84_FG
Opening communication with USB serial interface on port /d
Results:
SNR     BER         EPS
---     ---
8       5.09676e-06     2.13858e-06;
```

Figure 6.5 Screen capture of test results displayed by the BER test program

6.4.2 FPGA Controller Board

The FPGA controller acts like a bridge between the PC and the DUT. It receives control information from the PC and generates synchronized digital control signals (e.g. DAC clock, decoder clock, and frame) at appropriate times. It also receives DACCODE values from the PC and stores them into an on-board RAM. The stored values are then passed onto the DAC on the test support board. The controller waits for decoding to take place and then sends the decoded bits back to the PC.

6.4.3 Test Support Board

The test support board for the BER test is very similar to the one used in self-test. Besides having the USB device, variable voltage sources, and buffers, the test support board also includes a digital-to-analog converter AD9764 [65] which has

14-bit resolution and it can operate at up to 125 MS/sec. The DAC takes the 14-bit sample inputs and generates differential output currents. The currents are converted to voltages using resistive loads and they are passed through an analog buffer AD8138 which allows control of peak-to-peak amplitude and common-mode voltage through potentiometers. In order to run the BER test, we need a 3.3 V for the I/O ring of the decoder chip, a 1.8 V for the core circuit of the chip, 0.3 V for V_{refN} and 1.5 V for V_{refP}.

6.4.4 DUT Board

The DUT board contains a PGA socket for the decoder. Power supply and ground lines are routed to the test support board. Posts are used so that unit bias currents can easily be fed into the decoder. Test points and probes are added for signal probing and debugging.

6.4.5 Test Procedures

When performing the BER test, some parameters can be varied to obtain better decoding performance. The parameters include bias current, peak-to-peak amplitude and common-mode voltage for the differential voltages on the test support board, and decoder test speed. The general procedures of running the BER test are [39]:

1. Adjust I/O ring supply to 3.3 V, core supply to 1.8 V, V_{refN} to 0.3 V, and V_{refP} to 1.5 V. Verify the voltages and ground by probing signal using multi-meter.

2. Adjust differential voltages to full swing (1.8 V) and the common-mode voltage close to 1.8 V since this voltage will be divided equally between the input sample and hold capacitors. The voltages can be verified using an oscilloscope.

3. Set CLK_DIV = 63 in test program to get the slowest decoder testing speed of 155 kSps. See Table 6.1 for testing speeds.

4. Allow bit errors to be counted and wait for BER results and graphs.

Table 6.1 CLK_DIV and corresponding test speed and decoding speed [39]

CLK_DIV	Test Speed (Sps)	(8, 4) Decoded Bit Rate (bps)
0	8.33 M	3.702 M
1	4.54 M	2.018 M
2	3.12 M	1.387 M
4	1.92 M	0.853 M
8	1.087 M	0.483 M
16	581 k	258 k
32	301 k	133.8 k
63	155 k	68.9 k

6.4.6 Test Measurements

By running the BER test, we have observed that two out of four decoded bits perform as expected and the other two bits, which produce a flat BER around 0.5, appear to suffer from excessive comparator input offsets. The failure is simply due to a poor design choice in the comparators of the decoder. This problem is also addressed in [39] and [54] and any number of known solutions can be used in future generations of chips. For example, current comparators which employ input-offset canceling mechanisms may be used to counter the natural mismatch effects in semiconductor processing. The offset problem actually agrees with the self-test results of the I/O BIST in Section 6.3.6. This indicates that the I/O BIST has performed a correct diagnosis.

Since two of the comparators function with a tolerable offset, the test results of the corresponding bits provide a good indication of the decoder's overall performance. This is because each bit-position has a strong correlation with every other position; therefore, the performance of one bit can represent the performance of the full decoder. If the decoder is not working, none of the bits will produce meaningful results. So, we have used two out of four output bits to measure the BER.

We first ran BER tests with a constant unit bias current $I_U = 5$ μA and vary the test speed to see how it affects the performance of the decoder. The BER tests were run at 155 kSps, 1.087 MSps, and 8.33 MSps. The BER results are shown in Fig. 6.6. The curves indicate that a decoder running at a slower speed produces better BER performance and the BER moves towards ML. This is reasonable because a slower clock speed allows the decoder more time to converge and settle at a correct probability values. We then ran the BER test again and this time we kept the test speed constant at 155 kSps and varied the unit bias current. Fig. 6.7 shows the resulting BER curves. With a larger bias current, the decoder performs better since a larger I_U accelerates the settling time of the probability currents in the decoder. However, there is an upper bound for $\mathrm{I_U}$ to keep the decoder in subthreshold mode. So, it is appropriate to keep the bias current below approximately 10 μA.

By observing the BER curves, we know that if we want to run the decoder at faster speeds, we need to increase the bias current to maintain the decoder performance. We can also see that the extra test circuitry does not affect the performance of the (8, 4) Hamming decoder. In fact, the performance of the (8, 4) analog Hamming decoder with BIST gives a better BER than the digital decoder using the same code but without the extra parity check nodes. The curve produced by running the test at 155 kSps with I_U = 5 μA is only 0.3 dB to 0.4 dB from the ML curve. Although only two bits are measured from the analog decoder, its BER curve gives a significant indication that the analog implementation of the (8, 4) Hamming decoder can potentially outperform the digital implementations. Table 6.2 shows a summary of the decoder. The power consumption was obtained by measuring the current flowing through the core voltage supply and then multiplying it by the supply voltage. The energy per decoded bit was calculated by dividing the power consumption by the information throughput.

Table 6.2 Summary of (8, 4) Hamming decoder with BIST

Technology	TSMC 0.18 μm 1P6M 1.8 V
Analog area (with interfaces)	0.072 mm^2
Decoder BIST area	0.036 mm^2
I/O BIST area	0.006 mm^2
Total circuit area	0.138 mm^2
Clock speed	8.3 MSps
Decoding rate	3.7 Mbps
Self test run time	5 μs @ 12.5 MHz
Power	13 mW @ I_U = 4 μA, SNR = 8 dB, Speed = 3.7 Mbps
Energy/decoded bit	3.5 nJ/bit

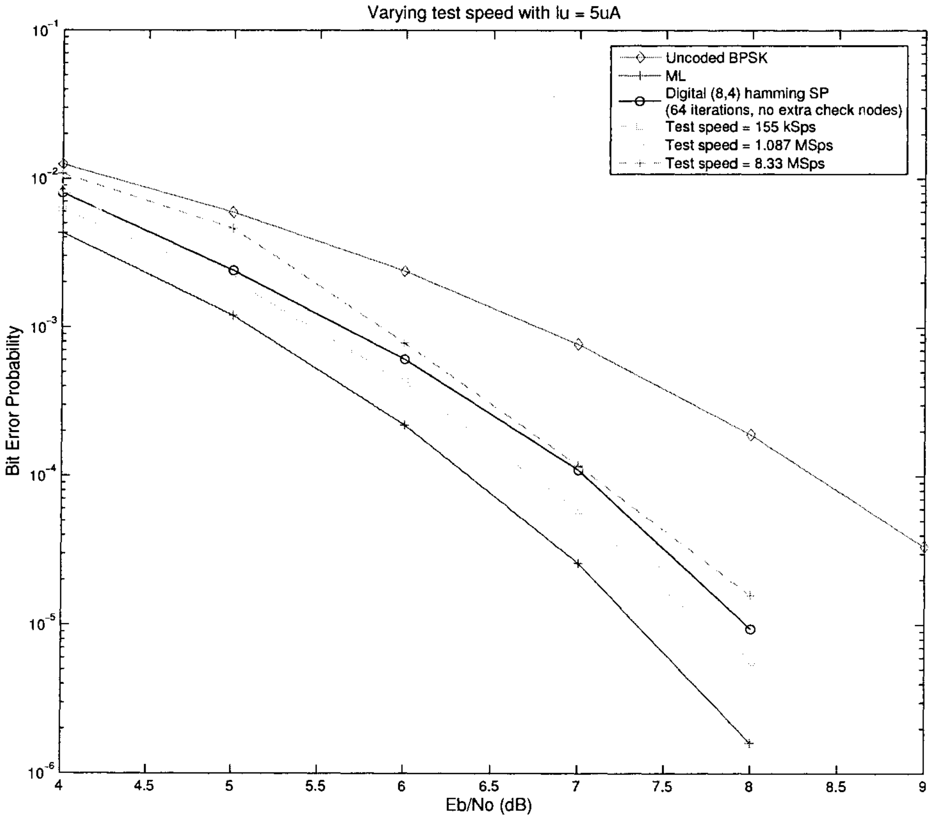


Figure 6.6 BER measurements with varying test speed

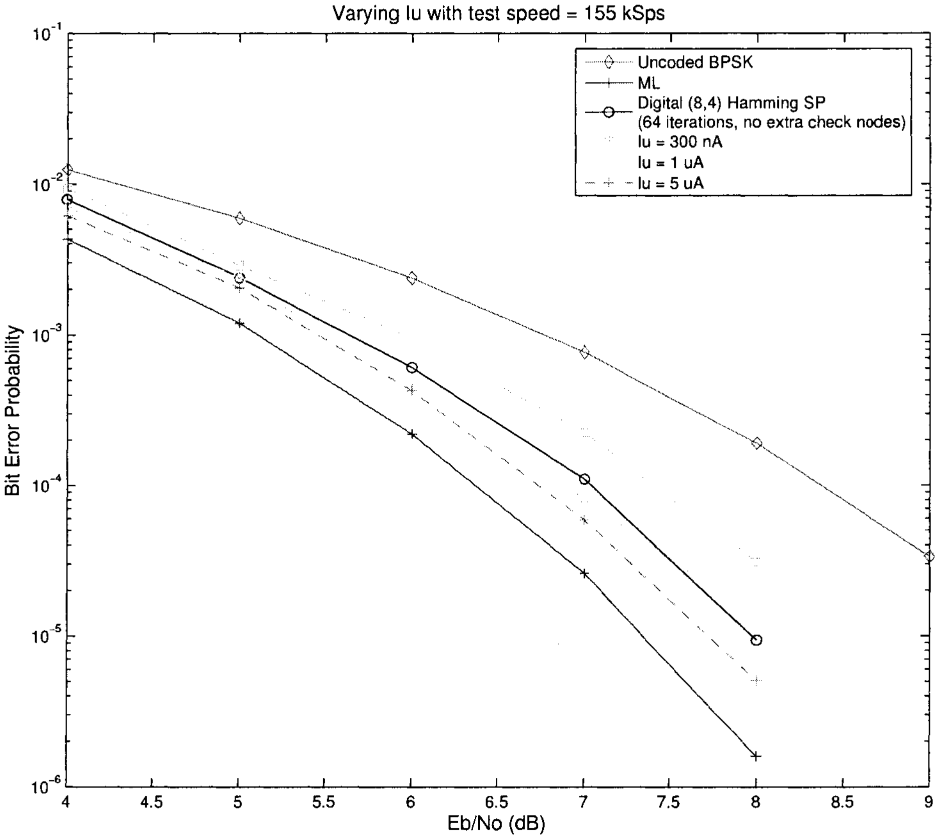


Figure 6.7 BER measurements with varying bias current I_U

6.5 Chapter Summary

In this chapter, we have described the testing of the analog decoder with BIST. Two different types of test were performed: self-test and BER test. The test setups of both tests were explained in detail and the self-test results and BER measurements were presented. In the next chapter, we conclude this thesis, provide recommendations for improvement, and discuss possible future work.

Chapter 7

Conclusions and Future Work

7.1 Conclusions and Contributions

The objective of this thesis has been to demonstrate the feasibility of testing analog iterative decoders in the digital domain with the use of built-in test hardware. A digital BIST scheme was proposed for testing sum-product analog decoders and the suggested design was implemented using a small (8, 4) extended Hamming analog decoder as a proof-of concept. The same testing scheme can be applied to sum-product analog decoders with bigger code sizes.

With the built-in test circuits, the (8, 4) extended Hamming analog decoder can perform testing by itself. We only need a simple test setup to control the self-test and read the results. We have shown that the decoder core BIST is capable of detecting catastrophic errors and the I/O BIST is capable of detecting comparator offset errors. Compared to running the BER test in minutes or even hours, the self-test can be done in microseconds, which greatly reduces the testing time. Since setup for BER test is also very complicated, we can save a great deal of time by eliminating the BER test setup. Therefore, by largely shortening the testing time of analog decoders, testing cost can significantly reduced during the manufacturing process since time means money.

A BER test was run to ensure that the extra test circuits and the modifications of the original decoder structure do not affect the performance of the decoder. The decoder can be clocked at 8.3 MHz giving a decoding rate of 3.7 Mbps. The test results have proven that the decoder is very robust even with the extra test circuitry added and decoder modifications, and we have also shown that the decoder can give a better BER than a digital decoder operating on the same code and decoding algorithm.

The energy consumed by each decoded bit is measured at 3.5 nJ/bit at 8 dB. This is relatively small compared to the (8, 4) Hamming decoder built by Winstead *et al.* [56], which has an energy consumption of 45 nJ/b. However, it is very difficult to have a fair comparison with other similar (8, 4) Hamming decoders since they are built using different technologies and I/O interfaces.

The main tradeoff of having test circuits inside the decoder chip is the hardware overhead. The BIST area occupied one-third of the total circuit area and the area of the decoder itself also increased quite seriously. The hardware overhead is approximately 50%. However, when implementing the (8, 4) Hamming decoder with BIST, we did not aim for minimum area since our main goal was to prove that analog decoders can be tested easily in the digital domain with simple built-in circuits. We have also added extra test features into the decoder, such as error injection and internal test result reading. These features can be eliminated to reduce the silicon area.

There are several potential ways to improve the area overhead problem. For example, digital BIST circuits can be designed with purely structural VHDL code instead of behavioral VHDL to minimize the number of standard cells being used, or they can also be designed using other methods instead of using VHDL such as RTL design. We can also reduce the silicon area by removing all the switches used for changing the V_{refN} and V_{refP} to V_{SS} and V_{DD}. It is because V_{refN} and V_{refP} are in fact connected to the input pins of the chip. The reference voltages are provided directly from power supplies outside of the chip; therefore, instead of feeding V_{refN} and V_{refP}, we can feed V_{SS} and V_{DD} during the decoder self test. The suggested methods of minimizing silicon area and eliminating extra test features can potentially improve the overhead to 30%. BIST seems to be economically unacceptable for analog decoders with small codes. However, analog decoders for larger codes require dramatically less silicon area than digital decoders [54]; therefore, the area reduction achieved through analog decoding can significantly outweigh the burden of a BIST. Moreover, for bigger analog decoders, we predict that the silicon area required for extra test circuitry should not increase linearly as the block length of the code. It is because instead of sending test patterns to or receiving test responses from all the unidirectional circuits in parallel, we can use a multiplexing scheme to partially send or receive data from the decoder. Then we can use the same digital BIST circuit to test large decoders.

In terms of designability, it is relatively easy to implement the decoder with built-in test hardware. Basically, we only need to add switches, either pass transistors or transmission gates, to modify the decoder such that it decomposes into an array of small XOR gates during test mode. The design of the BIST is also straightforward since the main tasks of the BIST are to send simple test patterns and analyze the results.

7.2 Future Work

Testability is a key barrier for analog decoders to be deployed in the communications market. With our successful demonstration of the digital BIST approach for analog decoders, such problems can be circumvented. We now propose the following future research topics.

More attention should be drawn on the I/O interface. The interface should be redesigned with more careful consideration and characterization since it can significantly affect the performance of an analog decoder. It is especially crucial to guarantee that the comparators make correct bit decisions. Currently, a few researchers have been looking into how to reduce the leakage currents in the input S/H cells and obtaining a small comparator offset in the output interface.

It is also challenging to provide a small bias current in the range of nA to μA especially for system-on-chip solutions. We were using a Keithley source unit to provide such small bias current for the decoder chip, but when analog decoders come to interface with other receiver components, a methodology should be arrived to produce the bias current. Some research work should be done to address this issue.

In this thesis, we have demonstrated the feasibility of having digital BIST for analog decoders by applying the suggested BIST scheme on a small (8, 4) Hamming decoder. In the future, the same testing idea should be applied to larger decoders and silicon area should be minimized to reduce the area overhead.

Moreover, since the digital BIST scheme proposed in this thesis is mainly for sum-product analog decoders, it may be beneficial to apply similar testing schemes for other decoders using different decoding algorithms. Researchers in our group have also implemented a low voltage analog decoder and have shown that the power usage of the decoder is way less than any other decoder designs. Therefore, future work should also be done on examining how to test low voltage analog decoders with the digital BIST concept.

Bibliography

[1] V. Agrawal, C. Kime, and K Saluja. A tutorial on built-in self-test: Principles. *IEEE Design & Test of Computers*, 10(1):73-82, 1993.

[2] V. Agrawal, K Saluja, and C. Kime. A tutorial on built-in self-test: Applications. *IEEE Design & Test of Computers*, 10(2):69-77, 1993.

[3] A. G. Amat, G. Montorsi, S. Benedetto, D. Vogrig, A. Neviani, and A. Gerosa. An analog Turbo decoder for the UMTS standard. In *IEEE Int. Symp. on Information Theory*, page 296, Chicago, IL, June 2004.

[4] K. Arabi, B. Kaminska, and J. Rzeszut. A new built-in self-test for digital-to-analog and analog-to-digital converters. In *Proc. Int. Conf. Computer-Aided Design*, pages 491-494, Los Alamitos, CA, 1994.

[5] B. Arnold. Built-in-self-test gains ground as ATE time and cost soar. Electronic design. 48(19):117-124, Dec. 2000.

[6] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. on Information Theory*, 20:284–287, Mar. 1974.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error correcting coding and decoding: Turbo codes. In *IEEE Int. Conf. on Communications*, pages 1064–1070, Geneva, Switzerland, May 1993.

[8] M. Burns and G. W. Roberts. *An introduction to mixed-signal IC test and measurement*. Oxford University Press, New York, 2001.

[9] M. Bushnell and V. Agrawal. Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits. Kluwer Academic Publishers, Norwell, MA, 2000.

[10] A. Chatterjee. Concurrent error detection and fault tolerance in linear analog circuits using continuous check-sums. *IEEE Trans. VLSI Systems*, 1(2):138-150, June 1993.

[11] A. Chatterjee, B.C. Kim, and N.Nagi. DC built-In self-test for linear analog circuits. *IEEE Design & Test of Computers*, pages 26-33, 1996.

[12] A. Chatterjee and N. Nagi. Design for testability and built-in self-test of mixed-signal circuits: A tutorial. In *Proc. VLSI Design Tenth Int. Conf.*, pages 388-392, Jan. 1997.

[13] I. D. Dear, C. Dislis, A. P. Ambler, and J. Dick. Economic Effects in Design and Test. *IEEE Design & Test of Cornputers,* 8(4):64-77, 1991.

[14] G. D. Forney Jr. The Viterbi algorithm. In *Proc. of the IEEE*, volume 61, no. 3, pages 268–278, Mar. 1973.

[15] G. D. Forney Jr. Codes on graphs: normal realizations. *IEEE Trans. on Information Theory*, 47: 520-548, Feb. 2001.

[16] R. G. Gallager. *Low-Density Parity-Check Codes*. MIT Press, 1963.

[17] V. Gaudet and G. Gulak. A 13.3Mbps 0.35µm CMOS analog Turbo decoder IC with a configurable interleaver. *IEEE J. of Solid State Circuits*, 38(11):2010–2015, Nov. 2003.

[18] B. Gilbert. A precise four-quadrant multiplier with subnanosecond response. *IEEE J. of Solid State Circuits*, 3(4):365–373, 1968.

[19] B. Gilbert. Translinear circuits: a proposed classification. *Electronics Letters*, 11(1):14–16, 1975.

[20] B. Gilbert. A monolithic 16-channel analog array normalizer. *IEEE J. of Solid State Circuits*, 19(6):956–963, 1984.

[21] J. Hagenauer. Decoding of binary codes with analog networks. In *Proc. 1998 Information Theory Workshop*, pages 13–14, San Diego, CA, Feb. 1998.

[22] J. Hagenauer and M. Winklhofer. The analog decoder. In *IEEE Int. Symp. on Information Theory*, page 145 Cambridge, MA, Aug. 1998.

[23] R. W. Hamming. Error detecting and error correcting codes. *Bell Sys. Tech. J.*, 29:147–160, Apr. 1950.

[24] S. Hemati, A. H. Banihashemi, and C. Plett. A high-speed analog min-sum iterative decoder. In *IEEE Int. Symp. on Information Theory*, pages 1768-1772, Sept. 2005.

[25] D. A. Johns and B. Zand. High-speed CMOS analog Viterbi detector for 4-PAM partial-response signaling. *IEEE J. of Solid State Circuits*, 37(7):895–903, July 2002.

[26] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. on Information Theory*, 47:498–519, Feb. 2001.

[27] B. P. Lathi. *Modern digital and analog communication systems*. Oxford University Press, New York, 1998.

[28] S. Lin and D. J. Costello. Error Control Coding. Prentice Hall, Englewood Cliffs, NJ, 2004.

[29] H.-A. Loeliger, M. Helfenstein, F. Lustenberger, and F. Tarkoy. Probability propagation and decoding in analog VLSI. In *IEEE Int. Symp. on Information Theory*, page 146, Cambridge, MA, Aug. 1998.

[30] H.-A. Loeliger, F. Tarkoy, F. Lustenberger, and M. Helfenstein. Decoding in analog VLSI. *IEEE Communications Magazine*, pages 99–101, April 1999.

[31] F. Lustenberger. *On the design of analog VLSI iterative decoders*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Nov. 2000.

[32] F. Lustenberger, M. Helfenstein, H. A. Loeliger, F. Tarköy, and G. S. Moschytz. An analog VLSI decoding technique for digital codes. In *Proc. IEEE Int. Symp. on Circuits and Systems*, volume 2, pages 424 - 427, June 1999.

[33] F. Lustenberger, M. Helfenstein, H.-A. Loeliger, F. Tarköy, and G. S. Moschytz. All-Analog Decoder for a Binary (18, 9, 5) Tail-Biting Trellis Code. In *Proc. ESSIRC 1999*, pages 362-365, Duisburg, Germany, Sept. 1999.

[34] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In *Cryptography and Coding. 5th IMA Conference (ed. C. Boyd), no. 1025 in Lecture Notes in Computer Science*, pages 100–111. Springer, 1995.

[35] D. J. C. MacKay and R. M. Neal. Near Shannon limit performance of low density-parity check-codes. *Electron. Letter*, volume 32, pages 1645-1646, Aug. 1996.

[36] K. Maggard and C. Stroud. Built-in self-test for analog circuits in mixed-signal systems. *In Proc. IEEE Southeast Regional Conf.*, pages 225-228, 1999.

[37] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. An analog 0.25μm BiCMOS tailbiting MAP decoder. In *IEEE Int. Solid-State Circuits Conf.*, pages 356-357, Feb. 2000.

[38] N. Nagi, A. Chatterjee, and J. A. Abraham. A signature analyzer for analog and mixed-signal circuits. In *Proc. Int. Conf. Computer Design*, Los Alamitos, CA, 1994.

[39] N. Nguyen. *Implementation of Sub-1V Analog Decoders.* MSc thesis, University of Alberta, Edmonton, AB, Sept. 2004.

[40] N. Nguyen, C. Winstead, V. C. Gaudet, and C. Schlegel. A 0.8V CMOS analog decoder for an (8, 4, 4) extended Hamming code. In *IEEE Int. Symp. on Circuits and Systems*, volume 1, pages 1116–1119, May 2004.

[41] C. Schlegel and L. Perez. *Trellis and Turbo Coding*, IEEE/Wiley, 2004

[42] C. Schlegel. *Trellis coding*. IEEE Press, New York, 1997.

[43] C. E. Shannon, "A mathematical theory of communication. *Bell Systems Tech Journal*, 27:379-423, 623-656, July 1948.

[44] C. Stroud, P. Karunaratna, and E. Bradley. Digital components for built-in self-test of analog circuits. In *Proc. IEEE int. ASIC Conf.*, pages 47-51, 1997.

[45] P. Sweeney. *Error Control Coding*. John Wiley & Sons, West Sussex, England, 2002.

[46] S. Tabatabaei and A. Ivanov. A built-in current monitor for testing analog circuit blocks. In *IEEE Int. Symp. on Circuits and Systems,* volume 2, pages 109-114, June 1999.

[47] E. Teraoka *et al.* A built-in self test for ADC and DAC in a single-chip speech CODEC. In *Proc. Int. Test Conf.*, pages 791-796, Los Alamitos, CA, 1993.

[48] R. Togneri and C. J. S. deSilva. *Fundamentals of information theory and coding design.* Chapman & Hall/CRC, New York, 2002.

[49] M. F. Toner and G. W. Roberts. A BIST scheme for a SNR, gain tracking and frequency response test of a sigma-delta ADC. *IEEE Trans. Circuits and Systems*, pages 1-15, Jan. 1995.

[50] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. on Information Theory*, 13:260–269, April 1967.

[51] D. Vogrig, A. Gerosa, A. Neviani, A. Amat, G. Montorsi, and S. Benedetto. A 0.35-µm CMOS analog turbo decoder for the 40-bit rate 1/3 UMTS channel code. *IEEE J. of Solid State Circuits*, 40(3): 753 - 762, Mar. 2005.

[52] N. Wiberg, H.-A. Loeliger, and R. Kötter. Codes and iterative decoding on general graphs. *Europ. Trans. Telecommunications*, volume 6, pages 513–525, Sept/Oct. 1995.

[53] N. Wiberg. *Codes and decoding on general graphs*. PhD thesis, University of Linköping, Sweden, 1996.

[54] C. Winstead. *Analog Implementation of a Product Decoder.* PhD thesis, University of Alberta, Edmonton, AB, Aug. 2004.

[55] C. Winstead, J. Dai, W. J. Kim, S. Little, Y.-B. Kim, C. Myers, and C. Schlegel. Analog MAP decoder for (8, 4) Hamming code in subthreshold CMOS. In *Proc. Advanced Research in VLSI Conference*, pages 132–147, Salt Lake City, UT, March 2001.

[56] C. Winstead, J. Dai, S. Yu, C. Myers, R. R. Harrison, and C. Schlegel. CMOS analog MAP decoder for (8, 4) Hamming code. *IEEE J. of Solid State Circuits*, 39(1):122–131, Jan. 2004.

[57] C. Winstead, N. Nguyen, V. Gaudet, and C. Schlegel. Low-voltage CMOS translinear circuits for analog decoders. In *Proc. Int. Symp. on Turbo Codes and Related Topics*, Brest, France, Sept. 2003.

[58] C. Winstead, N. Nguyen, V. Gaudet, and C. Schlegel. Low-voltage CMOS circuits for analog iterative decoders. Accepted to appear in *IEEE Trans. on Circuits and Systems* I, 2005

[59] C. Winstead and C. Schlegel. Density evolution analysis of device mismatch in analog decoders. In *IEEE Int. Symp. on Information Theory*, page 293, Chicago, IL, June 2004.

[60] M. Yiu, V. Gaudet, C. Schlegel, C. Winstead. Digital built-in self-test of CMOS analog iterative decoders. In *IEEE Int. Symp. on Circuits and Systems*, volume 3, pages 2204-2207, May 2005

[62] S. Yu. *Design and test of error control decoders in analog CMOS*. PhD thesis, University of Utah, Salt Lake City, Dec. 2003.

[63] S. Yu, C. Winstead, C. Myers, C. Schlegel, and R. R. Harrison. An analog decoder for (8, 4) Hamming code with serial input interface. University of Utah, March 2002.

[64] ---, Analog Devices. AD8541/AD8542/AD8544 Data Sheet: General-Purpose CMOS Rail-to-Rail Amplifiers, 2003.

[65] ---, Analog Devices. AD9764 Data Sheet: 14-bit, 125 MSPS TxDAC D/A Converter, 1999.

[66] ---, Digilent. Digilab 2E Reference Manual: Digilab 2E FPGA Development Board, 2002.

[67] ---, DLP Design. DLP-USB245 User Manual: USB to FIFO Parallel Interface Module, 2002.

[68] ---, Texas Instruments. SN74LV125A Data Sheet: Quadruple Bus Buffer Gates with 3-State Outputs, 2003.