**University of Alberta**

# Implementation of Sub-1V Analog Decoders

by

**Nhan Duc Nguyen** 

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
Fall 2004

*Your file Votre référence*
*ISBN: 0-612-95826-4*
*Our file Notre référence*
*ISBN: 0-612-95826-4*

Nothing in life is to be feared.
It is only to be understood.
—Marie Curie

To my family and Yoko
for their love, encouragement, and support.

# Acknowledgements

I would like to say many thanks to Dr. Vincent Gaudet and Dr. Christian Schlegel for their guidance, encouragement, funding, and in helping to make this possible. I am indebted to Dr. Gaudet for always taking time out of his busy schedule to deal with my questions and concerns whenever they come up.

I would like to thank Chris Winstead for coming up with the idea for this thesis. His guidance and enthusiasm from day one helped me to understand much about this research area.

I would like to thank the committee members Dr. Bruce Cockburn and Dr. Ioanis Nikolaidis for their helpful comments and suggestions.

I would like to extend gratitude to my colleagues Edmund Fung and David Li. Our various discussions about the universe and microcosmos created much laughter. I would like to thank Siavash for answering questions related to communications and his ever cheerful presence. I would like to thank the people in the VLSI lab who helped me with various mundane questions about Cadence: Tyler Brandon, John Koob, Craig Joly, Dan Leder, Raewadee Parmmukh, and Sue Ann Ung.

I would like to acknowledge the computer guys Paul Greidanus and Jacob Bresciani for keeping the network up and running. I would like to thank Robert Hang from the HCDC lab for helping with Xilinx related questions.

Thank you to those on the fourth and fifth floor of this fine building for putting up with my meddling and disturbance as I talk with the one sitting next to you while you were busy trying to concentrate on an important deadline.

This project was made possible by Canadian Microelectronics Corporation grants 0302CF-ICFAANN1 and 0304CF-ICFAANN2, NSERC, iCORE, and the Mary Louise Imrie Graduate Student Award.

# Table of Contents

# List of Tables

# List of Figures

# List of Symbols

## Acronyms

| | |
|---|---|
| ACS | Add Compare Select |
| ADC | Analog-to-Digital Converter |
| APP | A Priori Probability |
| AWGN | Additive White Gaussian Noise |
| BCJR | Bahl Cocke Jelinek Raviv |
| BER | Bit Error Rate |
| BiCMOS | Bipolar-Complementary Metal Oxide Semicondutor |
| BJT | Bipolar Junction Transistor |
| BL | Block Length |
| BPSK | Binary Phase Shift Keying |
| BSC | Binary Symmetric Channel |
| CC | Convolutional Code |
| CCW | Counterclockwise |
| CMOS | Complementary Metal Oxide Semiconductor |
| CW | Clockwise |
| DAC | Digital-to-Analog Converter |
| DRC | Design Rule Checking |
| DUT | Device Under Test |
| FEC | Forward Error Correction |
| FIFO | First-In First-Out |
| FPGA | Field-Programmable Gate Array |
| FSM | Finite State Machine |
| GUI | Graphical User Interface |
| I/O | Input Ouput |
| KCL | Kirchoff's Current Law |
| KVL | Kirchoff's Voltage Law |
| LDPC | Low Density Parity Check |
| LLR | Log-Likelihood Ratio |
| LVS | Logic Versus Schematic |
| MAP | Maximum a Posteriori |
| ML | Maximum Likelihood |
| MPSK | M-ary Phase Shift Keying |
| MUX | Multiplexer |

| | |
|---|---|
| NMOS | N-type Metal Oxide Semiconductor |
| PC | Personal Computer |
| PMOS | P-type Metal Oxide Semiconductor |
| QAM | Quadrature Amplitude Modulation |
| RAM | Random-Access Memory |
| RF | Radio Frequency |
| SNR | Signal-to-Noise Ratio |
| SR | Set Reset |
| TL | Translinear Loop |
| TS | Test Support (board) |
| USB | Universal Serial Bus |
| VHDL | VHSIC Hardware Description Language |

## Electrical

| | |
|---|---|
| $AVDD$ | analog decoder supply voltage |
| $DVDD$ | digital I/O supply voltage |
| $I_d$ | drain current |
| $I_f$ | forward current component in weak inversion |
| $I_r$ | reverse current component in weak inversion |
| $I_U$ | unit current representing probability 1 |
| $I_{\square}(V_g)$ | square transistor current when $V_{gs} = 0$ |
| $PVDD$ | pad supply voltage |
| $U_T$ | thermal voltage kT/q, equal to 25.9mV at 300K |
| $V_{DD}$ | supply voltage |
| $V_{ds}$ | drain-to-source voltage |
| $V_{gd}$ | gate-to-drain voltage |
| $V_{gs}$ | gate-to-source voltage |
| $V_{SS}$ | zero potential, ground |
| $V_T$ | threshold voltage |
| $V_U$ | unit voltage |
| $W/L$ | transistor width to length ratio or size ratio |

# Chapter 1

# Introduction

## 1.1 Motivation

Demand for data transmission and storage over the years has spawned new technologies for real time applications. Some of the well-known technologies include the third generation (3G), IEEE 802.11, and Bluetooth wireless standards. Mobile and fixed devices that operate on these networks need reliable, fast data pipes that are also power efficient.

Shannon [60] showed that, in a noisy environment, it is possible to provide protection for information bits by proper coding. A greater volume of information bits can be transmitted by increasing the code complexity. This complexity often grows quadratically or even exponentially as the Shannon limit is approached, as observed by [44]. Powerful codes that approach the limit are Turbo [7] and low density parity check (LDPC) [22, 47] type codes. These belong to a field of coding known as *probabilistic*, in which *soft* information processing techniques are used [43]. Decoders that operate on these types of codes deal with soft information, which is best represented by real numbers. In addition, the construction of the decoder has feedback loops where calculations are done in iterations. These two characteristics of probabilistic coding make digital implementations challenging. Thus, researchers have been looking into using analog circuits to perform such tasks [29, 42, 44, 50, 74, 69, 24, 3].

Analog decoding offers many advantages compared to its digital counterpart. The circuits do not switch from rail to rail, which means they do not dissipate as much dynamic power; however, static power due to leakage current

is still present. There is continuous time processing—voltages and currents in the decoder propagate throughout the network and come to some steady state. Clocking is only needed for input-output (I/O) integration with digital blocks. Decoding latency is largely determined by resistive-capacitive values in transistors and is not expected to change significantly with increased block size. Since signal processing is done in the current domain, sums are achieved by tying wires together and products are performed by using less than ten transistors (compare this to digital adders and multipliers). One only needs to replicate a multiplier circuit many times to implement probability propagation or sum product modules. These are placed in parallel to form the decoder network. Factor graphs and trellises, which are graphical forms of code representation, can be mapped directly into circuits. The growth of complexity with larger codes becomes almost linear. The design philosophy is very much like a digital design flow.

Currently, state-of-the-art analog decoders operate on small codes as proof of concept designs. Larger size decoders are being implemented as part of ongoing research. In order for analog decoders to gain industry acceptance, there are challenges to address. Analog decoders that work on practical length codes have to be designed. These codes and their bit rates have to meet current industry standards. There is the desire for added programmability to provide greater flexibility. Interfaces, which affect the ability to get bits into and out of the decoder, need to be improved. As the design gets larger, the efficient production of larger decoder chips will depend on their testability. Efficient testing methods for the chip, which can include design for testability (DFT) or built in self test (BIST), have still yet to be developed.

As manufacturing processes advance, designs have to operate at lower supply voltages to compensate for smaller transistor dimensions and lower gate oxide thicknesses. This project focuses mainly on addressing the concerns of lowering supply voltages so that current designs can be scaled to future processes. An added effect of lowering the supply voltage is lower power consumption. At the time of this writing, a paper on tackling power consumption is the second most cited within the IEEE Journal of Solid State Circuits [11]

despite being published in 1992. Power consumption becomes an even greater challenge as the industry moves to developing powerful radio handsets and implantable devices. To address the above concerns, two small proof of concept decoders are presented. These decoders operate at sub-1V supply voltages and are capable of being manufactured using advanced processes. They also dissipate very little power.

## 1.2 Thesis Outline

This thesis is presented in six chapters. Chapter 2 presents an overview of a communication system by describing its general structure, channel models, and types of codes, decoding algorithms, and codes on graphs. Chapter 3 describes previous analog decoding work leading to our low voltage sum product circuit. Chapter 4 presents the design of two small sub-1V analog decoders. Chapter 5 describes a test setup along with measurement results for the two decoders mentioned in Chapter 4. The core innovations of this thesis are found in Chapters 4 and 5. Chapter 6 concludes the thesis and presents future research problems.

# Chapter 2

# Background on Channel Coding

Channel coding is concerned with the protection of information bits to be transmitted. From this perspective, a communication system can be described as containing three parts: encoder, channel, and decoder as shown in Figure 2.1. The modulator, physical transmission medium, and demodulator are contained within the channel. Information bits $\mathbf{u}$ from the source are encoded to $\mathbf{x}$ which contains redundancy. Coded bits $\mathbf{x}$ are modulated for transmission, and demodulated at the receiver to $\mathbf{y}$. These steps inevitably distort the signal of interest. The total distortion is often modeled by the addition of white Gaussian noise $\mathbf{N}$. The demodulated signal $\mathbf{y}$ is decoded to an estimate of the original signal $\hat{\mathbf{u}}$.

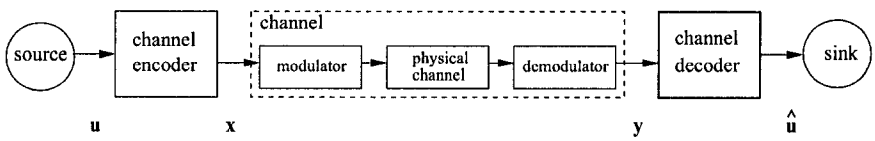



Figure 2.1: A communication system from the coding perspective

## 2.1 Channel

### 2.1.1 Additive White Gaussian Noise

The total noise incurred in the channel $\mathbf{N}$ adds to the modulated symbols by

$$\mathbf{y} = \mathbf{x} + \mathbf{N} \tag{2.1}$$

With zero mean and average power of $N_o$ within the transmission bandwidth, its probability density function is given by

$$p(N) = \frac{1}{\sigma_N \sqrt{2\pi}} \exp\left(\frac{-N^2}{2{\sigma_N}^2}\right) \tag{2.2}$$

where $\sigma_N$ is the noise variance and is related to its power by

$${\sigma_N}^2 = \frac{N_o}{2} \tag{2.3}$$

An individual received symbol $y$ at the channel output has the function

$$p(y|x) = \frac{1}{\sigma_N \sqrt{2\pi}} \exp\left(\frac{-(y-x)^2}{2{\sigma_N}^2}\right) \tag{2.4}$$

### 2.1.2 Demodulation

The value of the received symbol is declared using a demodulator. In *hard decision* demodulation, the probability of bit error due to AWGN is

$$p_b(\frac{E_b}{N_o}) = \frac{1}{2} erfc\left(\sqrt{\frac{R \cdot E_b}{N_o}}\right) \tag{2.5}$$

where $R$ is the rate of the code (defined in the next Section), $E_b$ is the source bit energy, $N_o$ is the one-sided noise power spectral density, and $erfc(\cdot)$ is the complimentary error function. The ratio $E_b/N_o$ is the normalized signal to noise ratio (SNR). When $R = 1$, (2.5) becomes the uncoded bit error rate (BER) for binary phase shift keying (BPSK) and is used as the base for comparisons to coded systems.

In *soft decision* demodulation, a decision and its reliability are given. The output can be represented as a log likelihood ratio (LLR).

$$LLR = log\frac{p(y_k|x_k = 0)}{p(y_k|x_k = 1)} \tag{2.6}$$

where $y_k$ and $x_k$ are the $k^{th}$ received symbol and code symbol. Its decision is $sign(LLR)$ and its reliability $|LLR|$. Soft demodulation, when used with coding, provides a BER improvement of approximately 2dB [61] over hard demodulation.

## 2.2 Codes

The type of coding this thesis deals with is forward error correcting (FEC), where, the receiver tries to identify and correct errors as it receives each code word.

The *rate* of a code is defined as

$$R = \frac{k}{n} \tag{2.7}$$

with $n$ is the length of the codeword in bits, $k$ is the number of information bits, and $n - k$ is the number of redundant parity bits.

The *capacity* represents the maximum amount of information that can flow across a channel. An AWGN channel has a capacity of [60]

$$C = W \cdot log_2\left(1 + \frac{S}{N}\right) \tag{2.8}$$

where $W$ is the bandwidth of the channel, $S = R \cdot E_b$ is the average signal power, and $N = N_o \cdot W$ is the total noise power.

Capacity, rate, and code length are related through Shannon's *Channel Coding Theorem* [60], which says: **(i)** for $R < C$, the error rate can be lowered by increasing the code length $n$, and **(ii)** for $R \geq C$, the error rate *cannot* be lowered.

A code which maps $\mathbf{u}$ to $\mathbf{x}$ such that $\mathbf{u}$ is visible in $\mathbf{x}$ is called *systematic*. A systematic code word could have information $u_i$ and parity bits $p_i$ organized as

| $u_k$ | ... | $u_1$ | $p_{n-k}$ | ... | $p_1$ |
|---|---|---|---|---|---|

An important indication of error control is the minimum distance $d_{min}$, the minimum difference in bit positions between valid code words. A greater distance increases the chance of decoding the correct transmitted word as shown in Fig. 2.2. In this example, a noise corrupted received word $\mathbf{y}$ is decoded to $\mathbf{x}_2$ because it is contained in $D_2$. Linear block codes such as Hamming codes (Section 2.2.1) are capable of correcting $\lfloor (d_{min} - 1)/2 \rfloor$ errors and detecting up to $(d_{min} - 1)$ errors but cannot do both simultaneously. In

fact the BER for linear block codes can be approximated as [5]

$$P_b \leq \frac{1}{2} \sum_{i=d_{min}}^{n} \left( \sum_{j=1}^{k} \frac{i}{k} \beta_{i,j} \right) erfc \left( \sqrt{iR \frac{E_b}{N_o}} \right) \tag{2.9}$$

where $\beta_{i,j}$ is the input output *enumerating* function, indicating the number of codewords of weight j generated by data words of weight i. This is referred to as the $d_{min}$ asymptote.

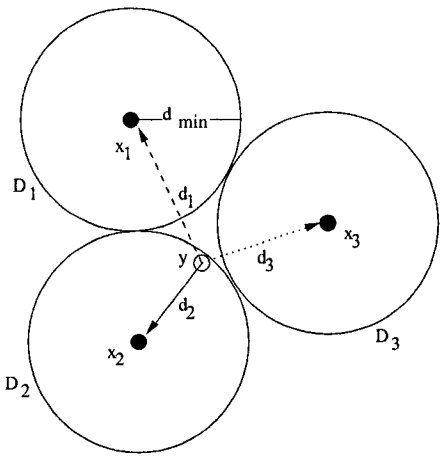



Figure 2.2: Decoding and $d_{min}$: $d_2 < d_{min}$ allowing $y$ to be decoded as $x_2$

### 2.2.1 Hamming Codes

Hamming developed the first error control code. It first appeared in [60] and was formally published in a separate article [32]. The parity checking of an $(n, k, d_{min})$ Hamming code is described by an $\mathbf{H}$ matrix. Its $n$ columns are constructed of $2^{n-k} - 1$ non-zero vectors of length $n - k$. Each row describes one parity check equation, where the bits included in the check are marked with '1'. The columns can be swapped in order to form

$$\mathbf{H} = [\ \mathbf{P}^T \mid \mathbf{I_{n-k}}\ ] \tag{2.10}$$

where $\mathbf{P}$ is the section of the matrix corresponding to parity checks, and $\mathbf{I}$ is an identity matrix. With the $\mathbf{H}$ matrix in the above form, a generator matrix $\mathbf{G}$ can be written as

$$\mathbf{G} = [\ \mathbf{I_k} \mid \mathbf{P}\ ] \tag{2.11}$$

Note that $\mathbf{G}$ and $\mathbf{H}$ are orthogonal

$$\mathbf{G} \cdot \mathbf{H}^T = 0 \tag{2.12}$$

Now the generator matrix $\mathbf{G}$ can be used for encoding $2^k$ source words of length $k$ to $2^k$ systematic code words of length $n$:

$$\mathbf{x} = \mathbf{u} \cdot \mathbf{G} \tag{2.13}$$

while the parity check matrix $\mathbf{H}$ can be used for decoding received word $\mathbf{y}$

$$\mathbf{s} = \mathbf{y} \cdot \mathbf{H}^T \tag{2.14}$$

When $\mathbf{y}$ contains one error, its *syndrome* $\mathbf{s}$ is non-zero giving the location of the bit in error. If $\mathbf{y}$ is correct, its syndrome will be zero. In the case of multiple errors, it is possible for $\mathbf{y}$ to turn into another valid codeword rendering the error *undetectable*. Since there are $2^k - 1$ valid codewords, there will also be $2^k - 1$ undetectable errors.

### Extending the Code

The Hamming family of codes have $d_{min} = 3$, capable of correcting one hard error. To simultaneously correct 1 error and detect 2 errors, the distance can be increased to 4 by adding an extra parity to check all $n$ bits in the $\mathbf{G}$ matrix:

$$\mathbf{G_{ext}} = \begin{bmatrix} & & p_1 \\ & \mathbf{G} & \vdots \\ & & p_k \end{bmatrix} \tag{2.15}$$

where $p_i$, $i = \{1, .., k\}$ is added to maintain even parity – if there are an odd number of 1s in the row, then $p_i$ is also 1, otherwise $p_i$ is 0. Consequently, the parity check matrix has to be extended from $\mathbf{H}$ to $\mathbf{H_{ext}}$ by

$$\mathbf{H_{ext}} = \begin{bmatrix} & & & 0 \\ & \mathbf{H} & & \vdots \\ & & & 0 \\ 1 & 1 & .. & 1 \end{bmatrix} \tag{2.16}$$

## 2.2.2 Convolutional Codes and Trellises

The mapping of source word to codeword can be done over time through the use of memory elements. This type of mapping is called *convolutional* encoding

and was first developed by [19]. A new variable, the *constraint length* $\nu$, is used to describe the total number of delay stages in the encoder. An $R = 1/2$, $\nu = 2$ encoder is shown in Fig. 2.3. Source bits $\mathbf{u}$ enter from the left into memory elements which cause delay $D$. Modulo 2 addition is done to give coded bits, which are multiplexed to form $\mathbf{x}$.

It is easy to visualize this encoding using a finite state machine (FSM) with $2^{\nu}$ states, as shown in Fig. 2.4. The states are given by the contents of the delay elements. Unfolding the FSM in time will produce a *trellis* as shown in Fig. 2.5. The trellis describes all possible code sequences that can be generated by the encoder. Each node in the trellis represents a state ($S \in \{0, .., 3\}$ in this case). Each edge represents a valid state transition showing the input that caused that transition along with the corresponding output bits. There are two edges leaving each state. The upper and lower edges represent inputs of '0' and '1' respectively. The encoding process starts in the all zero state and is brought back to the all zero state with $L$ *termination* bits.

It is important to note that Hamming codes can be encoded with a *tail-biting trellis* [10]. Such a trellis is time variant and its ending states are connected to its initial states. A valid codeword starts and ends in the same state.

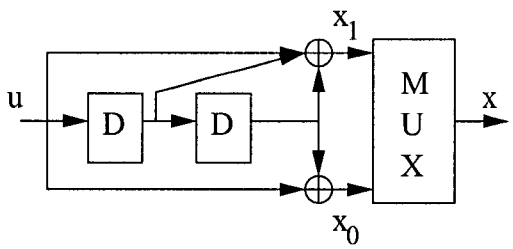



Figure 2.3: A simple $R = 1/2$, $\nu = 2$ convolutional encoder

### 2.2.3 Low Density Parity Check codes

An important class of codes called LDPC codes [22, 47] have gained much interest due to their incredible BER performance. An LDPC code is described by a very sparse $m$ x $n$ $\mathbf{H}$ matrix. It is *regular* if there are exactly $j$ number of ones in its columns and $k$ number of ones in its rows. To get a sparse $\mathbf{H}$

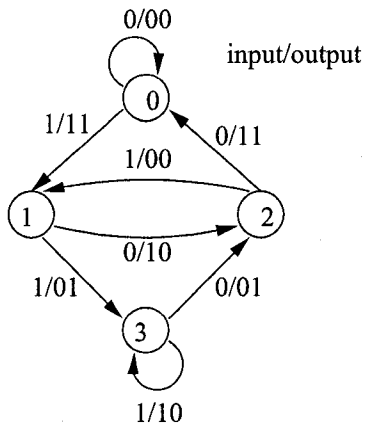



Figure 2.4: A finite state machine describing possible state transitions and input / outputs

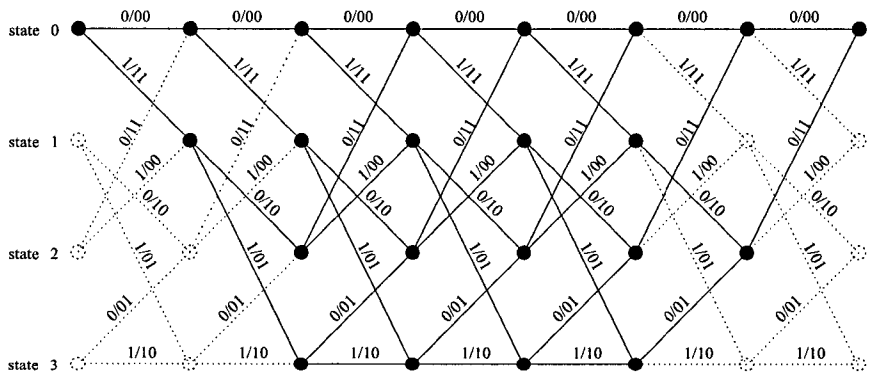



Figure 2.5: Trellis representation of a convolutional code with $n = 5$ and $L = 2$

matrix, conditions such as $j < k$, $j, k \ll n$, and $n > 1000$ are imposed. To get better performing codes, the $\mathbf{H}$ matrix is made irregular by varying the number of ones in each row and column. The best currently existing code is an $R = 1/2$, $n = 10^7$ irregular LDPC code reaching within 0.04 dB of the Shannon limit [12].

### 2.2.4 Turbo Codes

It is also possible to concatenate [39] two codes to form a hybrid third. A parallel concatenation encoding scheme is presented in Fig. 2.6(a). Information bits $\mathbf{u}$ are encoded to $\mathbf{x} = \{\mathbf{u}, \mathbf{p_1}, \mathbf{p_2}\}$ using two convolutional encoders, CC, separated by an interleaver $\pi$. The *interleaver* is used to randomize the order of incoming bits, spreading potential burst errors across its whole frame length.

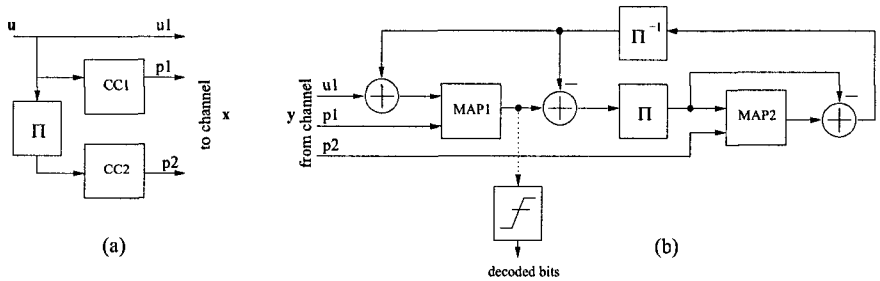



Figure 2.6: Parallel Turbo (a) encoder (b) decoder

A corresponding decoding architecture, called a *Turbo decoder* [7], is made up of two decoders (running the MAP algorithm explained in the next Section), interleaver, and de-interleaver are shown in Fig. 2.6(b). *Intrinsic* bits $\mathbf{u}$ and $\mathbf{p_1}$ are used by MAP1 to calculate *extrinsic information* for MAP2. This information is combined with intrinsic bits $\mathbf{p_2}$ to form extrinsic information for MAP1. The exchange of information between the two MAP decoders give rise to the term *iterative decoding* [31]. This powerful feedback technique continues until convergence or until a given amount of iterations has passed. The sharing of information results in BERs approaching the Shannon Limit.

## 2.3 Decoding on a Trellis Graph

### 2.3.1 Maximum Likelihood

The maximum likelihood (ML) decision rule is based on choosing a word $\mathbf{x}$ that will maximize the sequence probability $p(\mathbf{y}|\mathbf{x})$. The Viterbi Algorithm [66, 40] performs such computations on a decoding trellis using add, compare, and select (ACS) operations.

### 2.3.2 Maximum a Priori

To maximize the probability of individually choosing the correct coded bits $x_k$, the maximum a priori (MAP) decision rule performs

$$\hat{x}_k = \arg\max_x p(x_k|\mathbf{y}) \tag{2.17}$$

The BCJR algorithm [4] does such computation on a decoding trellis. It is a soft input soft output [6] algorithm which can be described as a function of states $s$.

The *a posteriori* probability (APP) of arriving in state $q$ at time $k$ is given by

$$p(s_k = q|\mathbf{y}) = p(s_k = q, \mathbf{y})/p(\mathbf{y}) \tag{2.18}$$

The above joint probability can be rewritten as

$$\begin{aligned} \lambda_k(q) &= p(s_k = q, \mathbf{y}^-, \mathbf{y}+) && (2.19)\\ &= p(s_k = q, \mathbf{y}^-)p(\mathbf{y}^+|s_k = q, \mathbf{y}^-) && (2.20)\\ &= \underbrace{p(s_k = q, \mathbf{y}^-)}_{\alpha_k(q)}\underbrace{p(\mathbf{y}^+|s_k = q)}_{\beta_k(q)} && (2.21) \end{aligned}$$

where vector $\mathbf{y}$ is broken into $\mathbf{y}^+$ and $\mathbf{y}^-$ meaning $\mathbf{y}^- = \{y_1, .., y_k\}$ and $\mathbf{y}+ = \{y_{k+1}, .., y_n\}$. Note that (2.19) changes to (2.20) by the property that if $s_k$ is known, events after time $k$ do not depend on $\mathbf{y}^-$. The forward metric through the trellis is $\alpha_k$ and the backward metric is $\beta_k$. These metrics depend on previously accumulated values and are calculated recursively as

$$\alpha_k(q) = \sum_{(p,q)} \alpha_{k-1}(p)\gamma_k(p, q) \tag{2.22}$$

and

$$\beta_k(p) = \sum_{(p,q)} \beta_{k+1}(q)\gamma_{k+1}(p,q) \tag{2.23}$$

where the sum is taken over all states connecting $p$ and $q$.

The channel metric $\gamma_k(p,q)$ is calculated as

$$\gamma_k(p,q) = p(s_k = q|s_{k-1} = p)p(y_k|s_{k-1} = p, s_k = q) \tag{2.24}$$

where the transitional probability $p(q|p)$ is the *a priori* information and is usually assumed to be equal for all transitions coming out of state p. The conditional probability $p(y_k|p,q)$ is actually (2.4) since the transition pair $p,q$ is weighted by $x_k$.

The output APP of an information bit is calculated as

$$p(u_k = 0|\mathbf{y}) = \sum_{S_0} \lambda_k(q)/p(\mathbf{y}) \tag{2.25}$$

$$p(u_k = 1|\mathbf{y}) = \sum_{S_1} \lambda_k(q)/p(\mathbf{y}) \tag{2.26}$$

where $S_0 = \{(s_k = p, s_{k+1} = q) : u_k = 0\}$ is the set of state transitions from $p$ to $q$ caused by an input of one and $S_1 = \{s_k = p, s_{k+1} = q) : u_k = 1\}$ is the set of transitions caused by a zero. The output is decoded as '0' if $p(u_k = 0|\mathbf{y}) > p(u_k = 1|\mathbf{y})$, otherwise it is '1'.

The algorithm is summarized as:

1. Initialize $\alpha_0(0) = 1$, $\alpha_0(p) = 0$ for $p \neq 0$, $\beta_0(0) = 1$, $\beta_0(q) = 0$ for $q \neq 0$.

2. For each $y_k$ received, compute $\gamma_k(p,q)$ and iterate $\alpha_k$ increasing $k$.

3. When the whole code word $\mathbf{y}$ has been received, iterate $\beta_k$ decreasing $k$.

4. Calculate $\lambda_k$ to decode information bits.

The MAP decision rule is different from ML because it can accommodate different a priori probabilities $p(q|p)$. When all input code bits are equiprobable, MAP and ML are equivalent [4].

## 2.4 Decoding on a Factor Graph

Much has been mentioned about decoding on the trellis representation of a code. Another form of representation, which is widespread due to LDPC codes, are called factor graphs [62, 67, 41]. In a factor graph, larger global functions which depend on many variables are broken down into products of smaller local functions which depends only a subset of those variables. The **H** matrix of a code can be translated into a factor graph with only variable and check nodes, as shown in Fig. 2.7(a). This is because the rows describe which variables are involved in parity checking while the columns indicates how many parity checks one variable is involved in. An edge is drawn from variable node $i$ to check node $j$ only if $h_{i,j} = 1$.

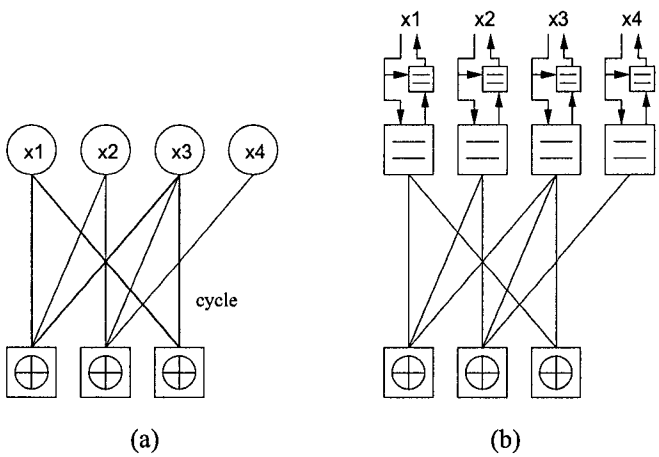



Figure 2.7: A factor graph (a) is transformed into (b) for implementation

The function of variable nodes is easier explained using equality constraints. This is because the output of a variable node depends on all its neighbors agreeing to some value. Not only that, intrinsic information has to be included to correctly arrive at a reliable decision. Both of these concerns are taken care of when the variable node is transformed into two equality nodes as shown in Fig. 2.7(b).

Now, given two input probability distributions $p_X = \{p_x(0), p_x(1)\}$ and

$p_Y = \{p_y(0), p_y(1)\}$ the equality function performs

$$\begin{bmatrix} p_z(0) \\ p_z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_x(0)p_y(0) \\ p_x(1)p_y(1) \end{bmatrix} \tag{2.27}$$

where a constant factor $\gamma$ is used to ensure $p_z(0) + p_z(1) = 1$.

The check function imposes even parity by:

$$\begin{bmatrix} p_z(0) \\ p_z(1) \end{bmatrix} = \begin{bmatrix} p_x(0)p_y(0) + p_x(1)p_y(1) \\ p_x(0)p_y(1) + p_x(1)p_y(0) \end{bmatrix} \tag{2.28}$$

Decoding on the graph follows the sum product algorithm [41] which, in essence, has one simple rule:

> The message passed from node $a$ to node $b$ on edge $e$ is a function of the messages sent to $a$ by its neighbors excluding $b$

Message passing from node to node can happen in discrete time and information is exchanged similar to that of a Turbo decoder. A typical algorithmic approach might be to:

1. Initialize the equality nodes with messages (intrinsic information)

2. Pass messages from equality nodes to check nodes (extrinsic information 1)

3. Calculate parity node information and pass messages from check nodes to equality nodes (extrinsic information 2)

4. Calculate equality node information and include intrinsic information on outgoing equality messages to arrive at decoded bits

5. Do the above for a fixed number of iterations or until there is convergence

Decoding performance on a factor graph is said to be optimal if the graph has no cycles otherwise it is non-optimal since marginal probabilities cannot be calculated [41].

## 2.5 Chapter Summary

In this chapter, important aspects about channel coding were discussed. A communication system from the coding perspective was presented and topics such as AWGN, demodulation, codes, and decoding were discussed. Decoding algorithms forming the basis of the implementations in Chapter 4 were also described. We now proceed to a description of current state-of-the-art implementations of decoders.

# Chapter 3

# Background on Analog Decoding and Circuit Theory

This chapter begins with a brief survey of analog decoder implementations to understand possible architectures, speeds, power dissipation, and robustness of analog decoding. Then the subject is narrowed down to introduce the core circuits used in the implementation of this thesis. Concerns such as minimum supply voltage, comparison of ideal to realized, and analog effects will be addressed.

## 3.1 Analog Decoding Definition

We first define analog decoding by using Fig. 3.1. In a traditional receiver system, as shown in (a), demodulated channel outputs $\mathbf{y}$ are quantized using an analog-to-digital converter (ADC). These bits are then decoded using a digital decoder. We can instead perform sample and hold (S/H) of $\mathbf{y}$ values and pass these analog values into an analog decoder where decoding operation is determined by the transient response of the circuits. Analog outputs are compared to arrive at digital decoded bits, as shown in (b). In (a) and (b), both decoders perform the same decoding algorithm. The analog / digital boundary in (b) is pushed further downstream.

The reasons why we might consider doing this includes smaller silicon area, and the elimination of the ADC. Even when we compare the S/H units, analog decoder, and comparators as a whole against the digital decoder, we still

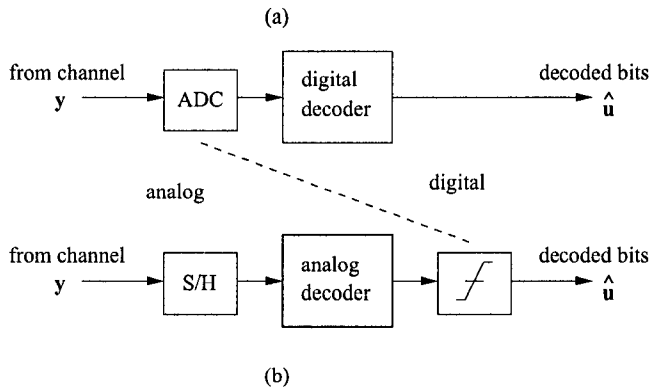



Figure 3.1: A comparison of (a) receiver structure using digital decoders and (b) receiver structure using analog decoders

Table 3.1: Energy efficiency comparison between existing digital (top 2) and analog decoders (bottom 4). ADC figures were not included for digital decoders.

| Who | Technology | Power | Throughput | Energy/ decoded bit |
|---|---|---|---|---|
| Blanksby et al. [9] | 0.16$\mu$m CMOS | 690mW@1.5V | 500 Mbps | 1.26 nJ/b core |
| Bickerstaff et al. [8] | 0.18$\mu$m CMOS | 292mW@1.8V | 2.048 Mbps | 142 nJ/b core |
| Gaudet et al. [25] | 0.35$\mu$m CMOS | 185mW@3.3V | 13.3 Mbps | 13.9 nJ/b core, IO, pads |
| Winstead et al. [70] | 0.5$\mu$m CMOS | 45.2mW@3.3V | 1 Mbps | 45 nJ/b core, IO |
| Moerz et al. [50] | 0.25$\mu$m BiCMOS | 20mW@3.3V | 160 Mbps | 0.125 nJ/b core |
| Amat et al. [3] | 0.35$\mu$m CMOS | 10.3mW@3.3V | 2 Mbps | 5 nJ/b core, IO, pads |

potentially take up less space. The ADC adds extra area. It is power hungry (consuming on the order of hundreds of mW) and is potentially a throughput bottleneck.

It is hard to make a direct comparison between existing analog and digital decoders since they are implemented with different technologies, have different functionalities, and operate on different codes with different algorithms. However, we are interested in making a rough comparison in terms of energy efficiency. We use the metric energy per decoded bit. That is, the amount of energy required to decode one information bit. Table 3.1 examines this metric for existing decoders. The top two decoders are digital and the bottom four are analog. The technology, power consumption, and throughput achieved are shown for each decoder. The energy per decoded bit is simply the power divided by the throughput. We observe that there is indication for analog decoders to be more energy efficient. However, what really tips the scale in analog's favor is when ADC power consumption is included in digital decoding schemes. It is predicted that analog decoders consume up to two orders of magnitude less in power for the same throughput [44].

## 3.2 Previous and Related Work

### 3.2.1 Analog Viterbi Decoding

Pioneering work was done in the late 1970's to build Viterbi decoders using analog circuits. Acampora constructed an analog Viterbi decoder [2, 1] from discrete components operating on a constraint length $\nu = 3$, $R = 1/2$ convolutional code. As shown in Fig. 3.2, the decoder takes two analog matched filter outputs and stores them by sample and hold. Each value gets converted into a differential pair to represent branch metrics. These values were added to existing path metrics (stored by capacitors), compared, and selected (ACS) using analog circuits. Survivor paths are stored in digital memory. There was a direct mapping from the time invariant trellis section into analog circuits. The construction of the decoder made use of microwave devices available at the time. The decoder was tested at 50Mbps with estimated speeds of upwards

to 200Mbps. The error correcting capability was 1dB from ML at a BER of $10^{-7}$.

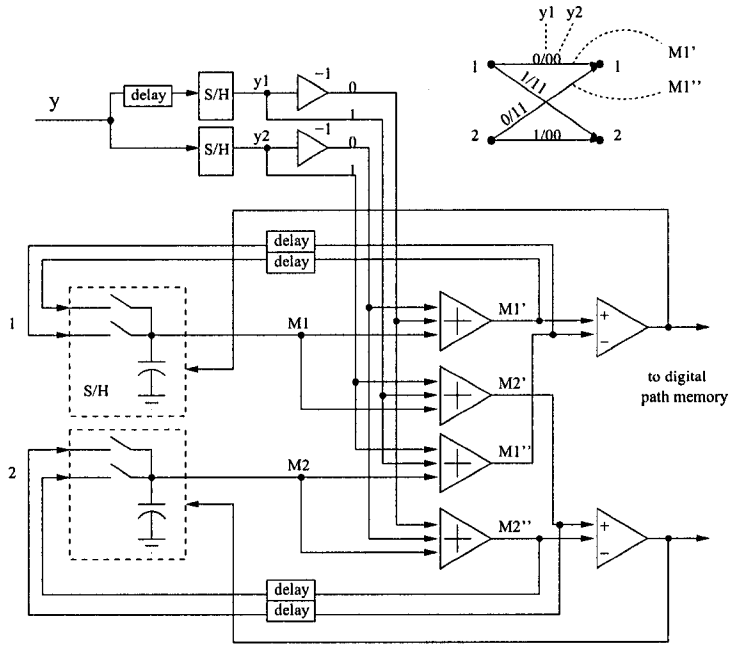



Figure 3.2: A trellis section mapped into circuits used for analog Viterbi decoding shown by Acampora [2]

In the early 1990's work was done to apply analog decoding to digital magnetic recording devices. An analog CMOS Viterbi detector was designed by Matthews et al. [48] for decoding class-IV partial response analog inputs. A partial response channel is one in which the output is a weighted sum of the present and previous channel inputs. If the weighted sum is a polynomial with two specific terms then it is called a dicode. A class-IV decoder is actually made up of two time interleaved independent dicode decoders. The architecture of the detector features analog path metric update and digital path survivor storage. There was use of differential signaling, master and slave S/Hs, summers and comparator blocks. The design was fabricated in $2\mu$m CMOS and the chip was tested at over 40Mbps. Its performance when compared to ideal (Viterbi Algorithm) is about 0.5dB off at a BER of $10^{-7}$. The power drawn from a 5V supply was 89mW.

A few years later, Shakiba et al. [59, 58] extended the work done by Matthews by incorporating dynamic threshold level adaptation to a BiCMOS

class-IV partial response decoder. The threshold levels adjust based on the history of the received signal minimizing the effects of noise. By eliminating analog feedback and an intermediate S/H stage in the path update, speeds of up to 100MSps per dicode were achieved giving an overall decoding rate of 200Mbps. At this speed, the BER was 1dB away from the Viterbi bound at a BER of $10^{-6}$. The decoder consumed 30mW from a supply of 3.3V.

### 3.2.2 Analog Networks

In the late 1990's efforts were made to adapt analog circuits to a new generation of codes. These codes, which include LDPC and Turbo codes, relied on iterative decoding algorithms.

Hagenauer [29] was one of the earliest to endorse the use of analog circuits for soft in soft out decoding. In such a decoder, messages can be LLRs represented as differential voltages. Using the nodes of a factor graph as the starting point, a parity check involving two LLRs is described by a 'boxplus' operation,

$$L(u_1) \boxplus L(u_2) = 2\tanh^{-1}\left[\tanh\left(\frac{L(u_1)}{2}\right)\tanh\left(\frac{L(u_2)}{2}\right)\right] \tag{3.1}$$

where $L(u_i)$ is defined by (2.6) in Section 2.1 and $u_i$ are statistically independent random variables. A variable node on the other hand performs a summation of two input LLRs. The variable node output (i.e. the decoded bits) can include intrinsic information as

$$L(x|y) = L_c y + L(x) \tag{3.2}$$

where $L_c = 4E_s/N_o$ is the channel state information and $L(x)$ is the extrinsic information provided by other bits. Note that these are just log domain versions of (2.27) and (2.28) from Sec. 4.1.

Both of the above equations can be realized by the generic decoder building block shown in Fig. 3.3, where the connectivity in the $\Sigma$ block decides circuit functionality. For example, the connectivity 'sum' will allow two LLRs to be added, while connectivity 'boxplus' performs

$$\Delta V_{LLR3} = 2U_{T3}\tanh^{-1}\left[\tanh\left(\frac{\Delta V_{LLR1}}{2U_{T1}}\right)\tanh\left(\frac{\Delta V_{LLR2}}{2U_{T3}}\right)\right] \tag{3.3}$$

Normalization of metrics is done by (1) diode loads and (2) voltage level shifters as indicated in Fig. 3.3.

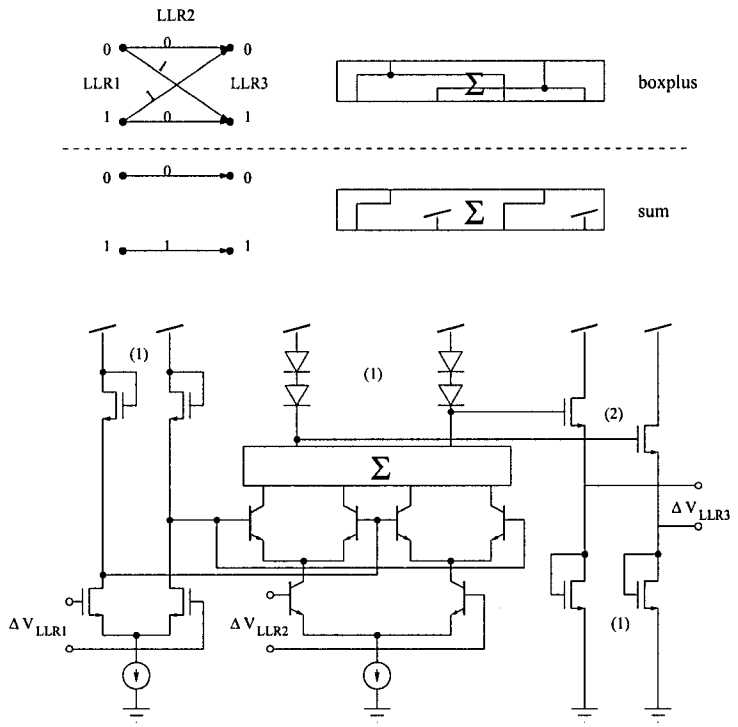



Figure 3.3: Generic decoder building block operating on LLR voltages [50]

A BiCMOS implementation of these ideas was reported in [50]. The analog core operated on an $R = 1/2$, $BL = 16$ convolutional code with a predicted parallel decoding speed of 160Mbps. The performance of the decoder was almost identical to its MAP simulation down to BER of $4{\cdot}10^{-5}$. The power drawn for the analog core is 20mW on a 3.3V supply.

Despite this, a possible limitation of this topology might be its LLR voltage dependence on temperature. By comparing (3.1) to (3.3), observe that $L(u_i) = \Delta V_{LLRi}/U_{Ti}$. The term $U_T$ is the thermal voltage and is equal to $kT/q$, where $T$ is the temperature. In a small decoder, $\Delta V_{LLR1}$ and $\Delta V_{LLR2}$ come from close proximity and are more likely to have $U_{T1} = U_{T2}$ giving an accurate result for $\Delta V_{LLR3}$. In a large decoder, however, the origins of input LLR voltages might be separated by larger distances and $U_{T1} \neq U_{T2}$. Thus input LLR voltages are scaled by different $U_T$s according to $\Delta V_{LLR}/U_T$ and the resulting $\Delta V_{LLR3}$ is inaccurate.

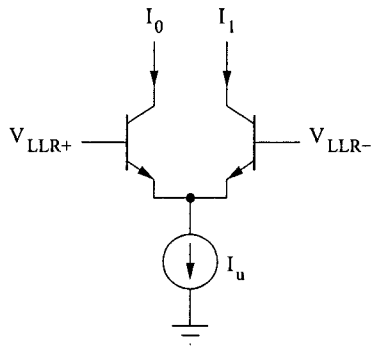



Figure 3.4: LLR voltage to probability current conversion

The alternative is to first convert LLR voltages into probability currents and then perform decoding using these currents. Loeliger et al. [42] was first to endorse such an approach. The conversion circuit mostly used is a differential transconductor [30] as shown in Fig. 3.4. Given an LLR as $\Delta V_{LLR} = V_{LLR+} - V_{LLR-}$, the output probability currents are

$$I_0 = I_u \frac{1}{1 + e^{\frac{\Delta V_{LLR}}{U_T}}} \tag{3.4}$$

$$I_1 = I_u \frac{e^{\frac{\Delta V_{LLR}}{U_T}}}{1 + e^{\frac{\Delta V_{LLR}}{U_T}}} \tag{3.5}$$

These probability currents are related to actual probabilities by $p(0) = I_0/I_u$ and $p(1) = I_1/I_u$, where $I_u$ is the *unit current* used to represent probability 1.

They called the basic decoder building block a *sum-product module* [45]. Shown in Fig. 3.5(a), this module could for instance represent a factor graph node or a trellis section in which two discrete probability masses $p_X, p_Y$ operate to yield a third, $p_Z$. The operation within the box depends on the trellis structure requiring sums and products (hence the name). Now, given a trellis with input nodes **x**, output nodes **z**, and branch metric **y**, the module performs

$$p_Z(z) = \gamma \sum_{x \in X} \sum_{y \in Y} p_X(x) p_Y(y) f(x, y, z) \tag{3.6}$$

where $f(x, y, z)$ is a {0,1} valued function which is equal to 1 only when x and z are connected by an edge y. The factor $\gamma$ is a normalizing factor used to ensure that the probabilities add up to 1.

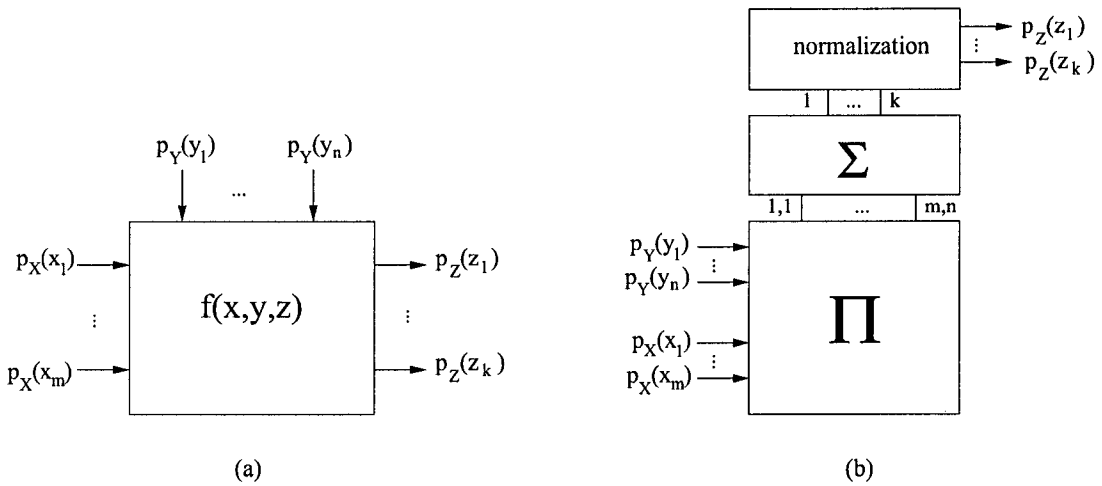



Figure 3.5: The basic decoder building block: (a) a sum-product module [45] and (b) its implementation separated into functions

This module is implemented as three functions: product Π, sum Σ, and normalization. The Σ function is realized by the connectivity between wires. The Π and normalization functions are realized using Gilbert vector multipliers [26] and Gilbert vector normalizers [28], respectively. A vector multiplier using bipolar junction transistors (BJTs) is shown in Fig. 3.6. The output currents drawn from the top can be shown to be

$$I_{i,j} = \frac{I_{x,i} I_{y,j}}{\sum_{j=1}^{n} I_{y,j}} \tag{3.7}$$

The input current (or output current) distribution is normalized to $I_u$ by using the circuit shown in Fig. 3.7. This circuit is one section of the vector multiplier and can be viewed as vector by scalar multiplication. For example, given input vector $I_w$, the outputs $I_x$ are scaled by $I_u$ according to

$$I_{x,i} = I_u \frac{I_{w,i}}{\sum_{i=1}^{m} I_{x,i}} \tag{3.8}$$

Such an operation is like current amplification. The application of such a multiplier and normalizing circuit to decoding is discussed in great detail in [34, 33]. The Gilbert multiplier architecture does not depend on temperature. This architecture, while similar to the core of Fig. 3.3, was reported two years earlier.

Loeliger's research group then went on to build several decoders, as described in [44]. The first working prototype featured a (5,2,3) trellis decoder

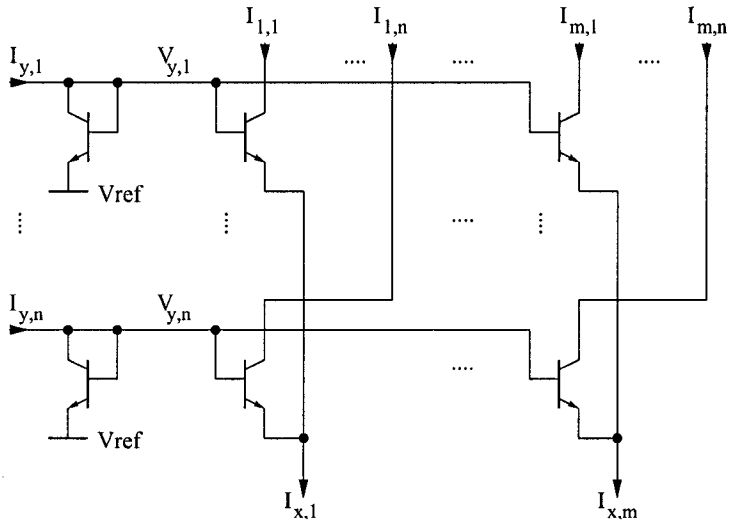



Figure 3.6: A Gilbert vector multiplier [26]

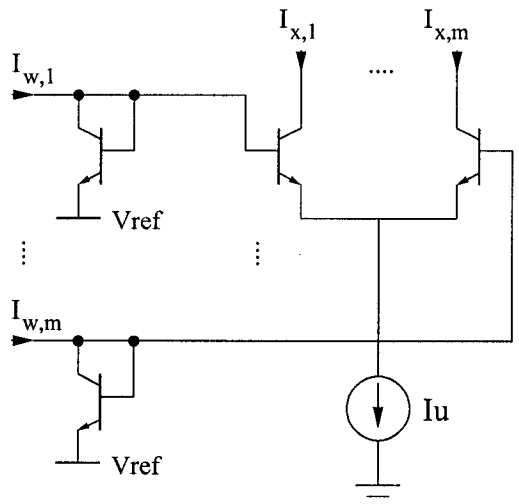



Figure 3.7: A Gilbert vector normalizer [28]

built using discrete transistors on a printed circuit board. The output is a probability value indicated by one LED in a row of LEDs. This effort was followed up by two 0.8$\mu$m BiCMOS implementations of a (18,9,5) tail biting decoder and a (44,22,8) quasi cyclic repeat accumulate decoder, respectively. The (18,9,5) tail biting decoder was estimated to decode at 100Mbps consuming 50mW on 5V supply. Unfortunately both IC designs suffered problems due to the on chip digital-to-analog converter (DAC) and other I/Os used for test support. Although transient simulations would indicate correct analog decoding behaviour, they could not get any measurement results. Years later, they fabricated an (8,4) Hamming decoder in 0.25$\mu$m BiCMOS using only CMOS transistors [21]. The measured BER was almost identical to the simulated ideal down to a BER of $10^{-3}$. This performance held at a supply of 0.7V albeit with much reduced speed. The power consumption was below 50$\mu$W at a 1.8V supply.

As hinted above, it is possible to implement CMOS versions of Figs. 3.6 and 3.7. However, these transistors have to operate in weak inversion with $V_{gs} < V_{th}$. The first working all CMOS analog decoder was built by Winstead et al. [69]. It mapped directly from an (8,4) Hamming tail biting trellis into 0.5$\mu$m CMOS. This design followed the philosophy outlined by Loeliger et al. [42] using sum product modules constructed from Gilbert cell multipliers. The multipliers, in this case, relied on CMOS transistors operating in weak inversion. Another change in this design was the emphasis on moving the troublesome DAC circuitry used for test support to outside the chip. The analog-in-digital-out interface created [76] would go on to become the universal analog decoding interface used by many other designs of the group. Recently, the largest decoder known to date based on a $(16,11)^2$ Turbo product code, has been designed and manufactured in 0.18$\mu$m CMOS [68]. Preliminary testing recorded promising results.

In light of the architecture presented by Loeliger and the success of its CMOS counterpart, an Italian research team has worked to apply these circuits to disk drive channels. A Turbo decoder was designed in 0.18$\mu$m CMOS technology and presented in [74]. The decoder is constructed with an $R = 8/9$,

punctured outer convolutional encoder, a 540 bit interleaver, and a precoded $R = 1$, extended PR-IV channel as an inner code. It was predicted that with a decoding rate of 400Mbps, 500mW would be consumed on a supply of 1.8V. It was not clear whether the decoder was manufactured since no measurement results were reported. Instead, the group focused on another project which used the same architecture to produce an $R = 1/3$, $BL = 40$ UMTS Turbo decoder [3]. The decoder was measured to operate on channel data at 2Mbps. The performance curve was 0.5dB away from ML over a BER range of $10^{-1}$ to $10^{-5}$. When biased on supply 3.3V, it consumed 10.3mW (6.8mW for the analog core).

The decoders mentioned up to this point have implemented the Viterbi [66, 40], Turbo iterative decoding [31], sum product [41], and MAP [4] algorithms. Efforts have also been made to directly translate the Log-MAP [55] algorithms into circuits.

Gaudet and Gulak [24] designed an $R = 1/3$, $BL = 48$, configurable interleaver analog Turbo decoder in 3M1P 0.35$\mu$m CMOS. The idea of a configurable interleaver allows for accommodation of different standards at power up. The interleaver is built by using networks of crossbars [23]. One method of building such interleavers is to decompose a length L interleaver into two factors P and Q. A 3-level crossbar network can then be built using those factors. The crossbar itself is decomposed into a chain of digital shift registers which control a grid of pass transistors. The decoder contains 4-state MAP modules which operate on the Log-MAP algorithm [55]. The basic building blocks are two circuits which operate in weak inversion as shown in Fig. 3.8. The state metric circuit performs the MAX*$(x, y) = ln(e^x + e^y)$ function of the Log-MAP algorithm and the branch metric circuit is a transconductor similar to Fig. 3.4. These blocks are used to map the trellis into circuits as given by the example shown in Fig. 3.9. The a priori and branch metrics are added to previous state metrics. Then a MAX* function is taken. The decoder had a measured decoded throughput of 13.3Mbps limited by test equipment. Its BER was 1.1dB off of simulated MAP at BER of $4{\cdot}10^{-5}$. Its total power consumption including I/O and pads was 185mW on 3.3V supply.

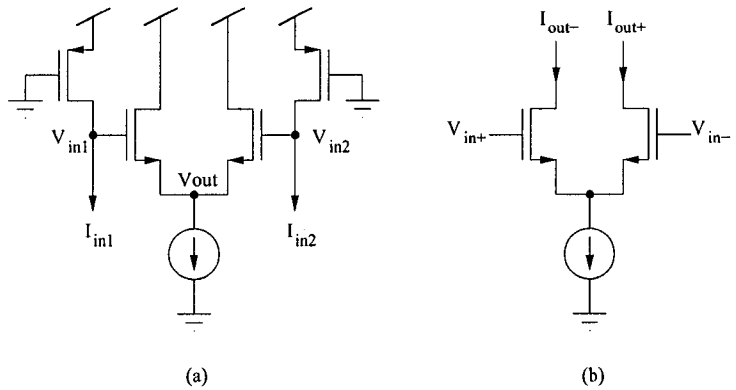



Figure 3.8: The (a) MAX* and (b) a transconductor circuit are the building blocks of a Max-Log-MAP decoder[24]

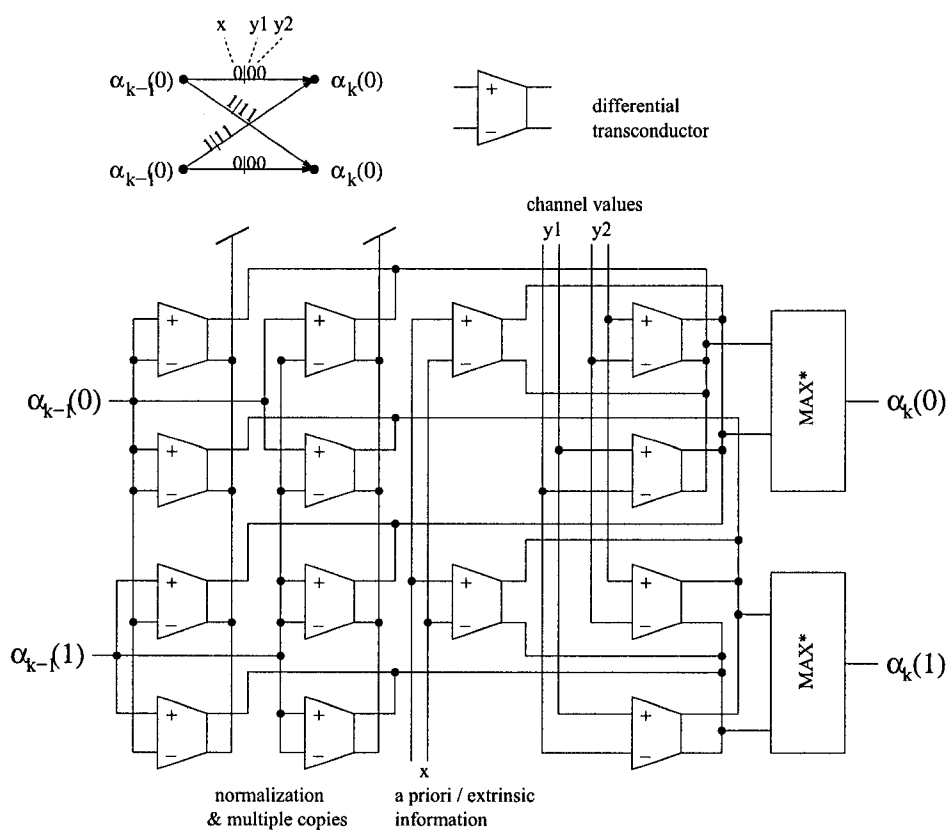



Figure 3.9: The mapping of an example trellis section using MAX* and transconductor circuits [24]

Another way to implement the Log-MAP algorithm is to use weakly inverted multiple input floating gate MOS (MI-FGMOS) transistors [51]. Given a trellis section such as shown on the left of Fig. 3.10, the algorithm needs to calculate the following for the $\alpha$ metric

$$\begin{aligned} \alpha_k(0) &= log[exp\{\alpha_{k-1}(0) + \gamma^u(0) + \gamma^c(0)\} \\ &\quad + exp\{\alpha_{k-1}(1) + \gamma^u(1) + \gamma^c(1)\}] \end{aligned} \tag{3.9}$$

where $\alpha$ is the forward metric, $\gamma^u$ is the source and $\gamma^c$ is the code value for interval $k$ all represented in log probabilities. Such an equation can be realized with the circuit topology as shown on the right of Fig. 3.10. Metric terms are represented as voltages and are connected to an MI-FGMOS which performs a summation and then overall exponential function to yield an output current. One MI-FGMOS is needed for each exponential term. Two currents are then added and are mirrored into a diode connected MI-FGMOS to perform a log operation. The resulting output voltage $V\alpha_k(0)$ is

$$\begin{aligned} V\alpha_k(0) &\propto log[exp\{V\alpha_{k-1}(0) + V\gamma^u(0) + V\gamma^c(0)\} \\ &\quad + exp\{V\alpha_{k-1}(1) + V\gamma^u(1) + V\gamma^c(1)\}] \end{aligned} \tag{3.10}$$

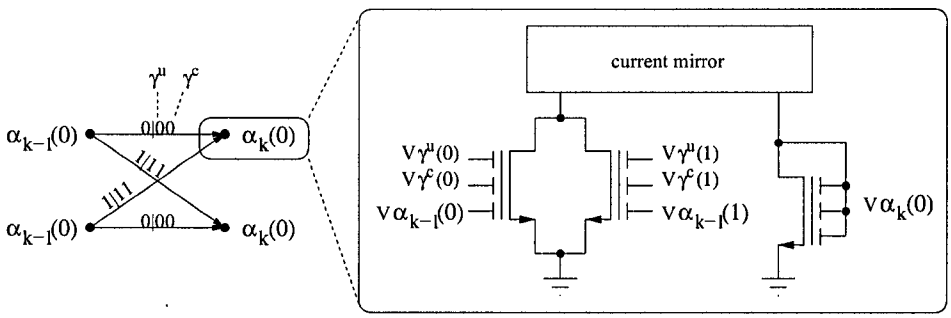



Figure 3.10: Using multiple input floating gate MOS transistors to implement a trellis section performing the Log-MAP algorithm [51]

A single cdma2k 8-state $\alpha$ stage was implemented in 0.5$\mu$m CMOS. The stage was tested by successively applying new log probability values while saving the accumulated $\alpha$ metric. Normalization was provided off chip. A measured speed of 7.8 Mbps was achieved. Power consumption was 4.35mW on a 3.3V supply.

The above presented background to previous analog decoding research. Additional work in the area can be found in [35, 36, 63, 37, 13, 64, 65]. A quick literature review indicates that the architecture proposed by Loeliger et al. is widely adopted. The implementations in this thesis will build on this momentum by modifying the sum product module to operate at low supply voltages. The next section will introduce these circuits and discuss effects such as mismatch, supply voltage, and accuracy when compared to ideal.

## 3.3 Low Voltage CMOS Transistor Behaviour

The construction of sum product modules is easily done in current mode where sums are achieved by tying wires together and products are done using BJTs or CMOS transistors operating in weak inversion. We choose to use CMOS because it is widely available and has a lower cost of manufacturing. Other advantages in using CMOS include low power dissipation, scalability, and easy integration with digital circuits. The drawbacks of CMOS in weak inversion include slower operation and greater mismatch.

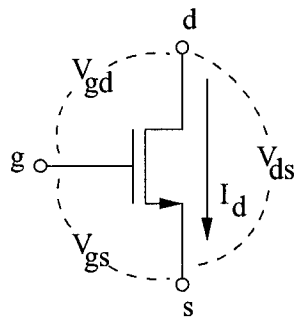



Figure 3.11: A CMOS transistor defining terminals, voltages, and drain current

### 3.3.1 Weak Inversion Modeling

In our analysis, we shall use the transistor model developed by Seevinck et al. [56]. Using Fig. 3.11 as a reference, CMOS *weak inversion* occurs when the gate to source voltage of the transistor is less than its threshold voltage $V_{gs} < V_{th}$. The drain current $I_d$ can be characterized by a forward $I_f$ and reverse $I_r$ component. When its drain to source voltage is $V_{ds} \geq 200mV$, the transistor

is in *saturation* and $I_f$ dominates giving $I_d = I_f$. When $V_{ds} < 200mV$, the transistor is *unsaturated* and $I_d = I_f - I_r$. The forward and reverse current components are

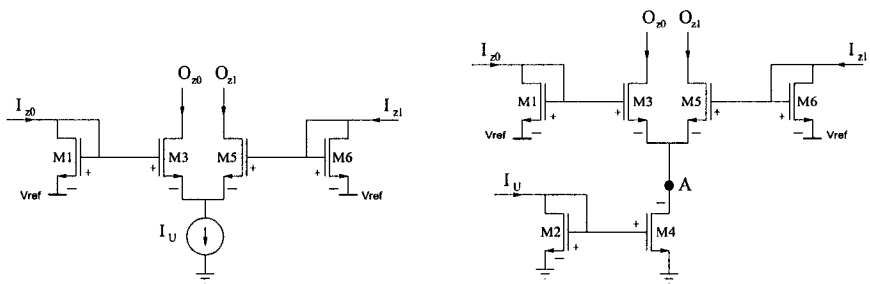
$$I_f = \frac{W}{L} \cdot I_{\square}(V_g) \cdot e^{V_{gs}/U_T} \tag{3.11}$$

$$I_r = \frac{W}{L} \cdot I_{\square}(V_g) \cdot e^{V_{gd}/U_T} \tag{3.12}$$

where $I_{\square}(V_g)$ is the current for a square ($W = L$) transistor when $V_{gs} = 0$. The specifics of $I_{\square}(V_g)$ are described in [56]; note that it represents the *body effect*. $U_T = 25mV$ is the thermal voltage and $W/L$ is the transistor width to length ratio or *aspect ratio*. Solving for $V_{gs}$ and $V_{gd}$,

$$V_{gs} = U_T \cdot ln \left[ \frac{I_f}{\frac{W}{L} \cdot I_{\square}(V_g)} \right] \tag{3.13}$$

$$V_{gd} = U_T \cdot ln \left[ \frac{I_r}{\frac{W}{L} \cdot I_{\square}(V_g)} \right] \tag{3.14}$$




(a) with ideal bias current (b) with bias current transistor

Figure 3.12: Gilbert current multiplier

### 3.3.2 Current Multiplication

The voltage to current characteristics can be used to analyze a CMOS Gilbert current multiplier [57] as shown in Fig. 3.12. Current inputs arrive through diode connected pairs and are mirrored to internal transistors. The requirements for correct operation are that transistors M3 through M5 be in saturation. $V_{ref}$ is needed to lift the common mode voltage of point A so that the

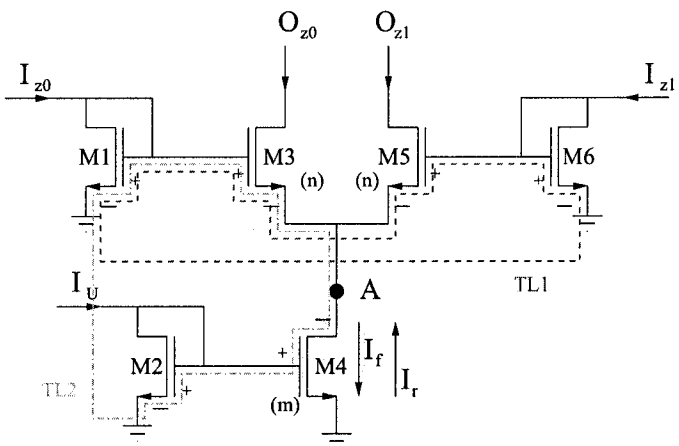



Figure 3.13: Analysis of the low voltage multiplier [56]

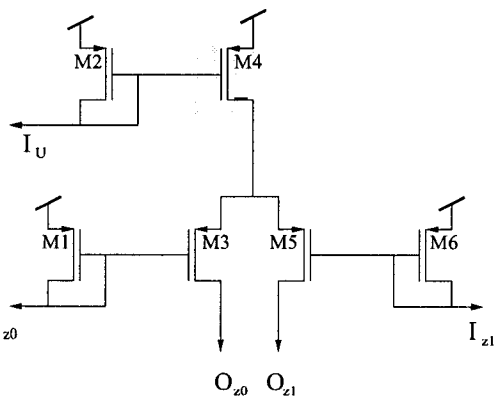



Figure 3.14: P-type low voltage current multiplier

drain of M4 has enough voltage to remain in saturation. When the above requirements are satisfied, a KVL loop can be drawn encircling the gate to source voltages of M1, M3, M5, M6 to form a *translinear loop* (TL). A translinear loop is crucial in allowing us to perform a linear analysis [27] on the voltage and currents traversing that loop. The analysis are as follows

$$\sum_{CW} V_{gs} = \sum_{CCW} V_{gs} \tag{3.15}$$

where clockwise ($CW$) and counterclockwise ($CCW$) indicates the direction of voltage or current flow. With $V_{gs}$ given as (3.13), a summation of logs translates into a product of currents as

$$\prod_{CW} I_d = \prod_{CCW} I_d \tag{3.16}$$

By exploiting this property, and by seeing that $I_u = O_{z0} + O_{z1}$, the output current pair in a Gilbert multiplier (Fig. 3.12(b)) can be shown to be

$$\begin{bmatrix} O_{z0} \\ O_{z1} \end{bmatrix} = \frac{I_u}{I_{z0} + I_{z1}} \begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} \tag{3.17}$$

This circuit is actually a CMOS implementation of the vector normalizer shown earlier in Fig. 3.7 with two inputs and two outputs.

### 3.3.3 Low Voltage Analysis

To reduce the supply voltage, we can remove $V_{ref}$ and replace it with ground to get the configuration shown in Fig. 3.13 [56]. In this case, M4 is going to be *unsaturated* while M3 and M5 will remain in *saturation.*

Now to derive the output current relationship, let all transistors have the same (W/L) ratio or *size ratio* with the exception of M3, M4, and M5 which have ratios of $n$, $n$, and $m$ respectively. Following (3.15), from translinear loop 1 (TL1) we get

$$V_{gs1} + V_{gs5} = V_{gs3} + V_{gs6} \tag{3.18}$$

$$U_{T1} \cdot ln\left[\frac{I_{z0}}{\frac{W}{L} \cdot I_{\square}(V_{g1})}\right] + U_{T5} \cdot ln\left[\frac{O_{z1}}{n \cdot \frac{W}{L} \cdot I_{\square}(V_{g5})}\right] = U_{T3} \cdot ln\left[\frac{O_{z0}}{n \cdot \frac{W}{L} \cdot I_{\square}(V_{g3})}\right] + U_{T6} \cdot ln\left[\frac{I_{z1}}{\frac{W}{L} \cdot I_{\square}(V_{g6})}\right] \tag{3.19}$$

Assuming local *temperature matching* $U_{T1} = U_{T5} = U_{T3} = U_{T6}$,

$$\frac{I_{z0}}{I_{\square}(V_{g1})} \cdot \frac{O_{z1}}{n \cdot I_{\square}(V_{g5})} = \frac{O_{z0}}{n \cdot I_{\square}(V_{g3})} \cdot \frac{I_{z1}}{I_{\square}(V_{g6})} \tag{3.20}$$

Since $V_{g1} = V_{g3}$ and $V_{g5} = V_{g6}$,

$$I_{z0} \cdot O_{z1} = O_{z0} \cdot I_1 \tag{3.21}$$

$$\Rightarrow O_{z1} = \frac{O_{z0} \cdot I_{z1}}{I_{z0}} \tag{3.22}$$

Similarly, the same analysis for TL2 gives

$$V_{gs1} + V_{gd4} = V_{gs2} + V_{gs3} \tag{3.23}$$

$$\frac{I_{z0}}{I_{\square}(V_{g1})} \cdot \frac{I_{r4}}{m \cdot I_{\square}(V_{g4})} = \frac{I_u}{I_{\square}(V_{g2})} \cdot \frac{O_{z0}}{n \cdot I_{\square}(V_{g3})} \tag{3.24}$$

$$\Rightarrow \frac{I_{z0} \cdot I_{r4}}{m} = \frac{I_u \cdot O_{z0}}{n} \tag{3.25}$$

Performing KCL into point A gives

$$I_{r4} + O_{z0} + O_{z1} = I_{f4} = m \cdot I_u \tag{3.26}$$

$$\Rightarrow I_{r4} = m \cdot I_u - (O_{z0} + O_{z1}) \tag{3.27}$$

Combining (3.22), (3.27), (3.25) and solving algebraically, we get

$$\begin{bmatrix} O_{z0} \\ O_{z1} \end{bmatrix} = \frac{m \cdot I_u}{\left(\frac{m}{n}\right) I_u + I_{z0} + I_{z1}} \begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} \tag{3.28}$$

where $n$ is the size ratio of M3 and M4, and $m$ is the size ratio M5. The minimum required supply voltage in such a configuration is the sum of the transistor threshold voltage and the drain source saturation voltage [56].

It must be stressed that in the above derivation, local temperature matching results in a cancellation of the thermal voltage from (3.19) to (3.20). Furthermore, $I_{\square}(V_g)$ terms are cancelled in moving from (3.20) to (3.22) due to common gate voltages thereby eliminating the *body effect*. The result is a robust current multiplier having the same benefits as the regular Gilbert multiplier but with slightly different output currents.

If all size ratios in Fig. 3.13 are made equal (i.e. $m = n = 1$), the difference between (3.28) and (3.17) is the presence of an $I_u$ term in the denominator. Now, let $I_Z = \{I_{z0}, I_{z1}\}$, $I_U = \{I_u, I_{\bar{u}}\}$ form the input probability distributions to a sum product module, and $O_Z = \{O_{z0}, O_{z1}\}$ be its output. When we perform products on two probability masses, the denominator in (3.17) will yield a constant, whereas in (3.28) the denominator will vary with $I_u$. To fix this, an extra term $I_{\bar{u}}$ needs to be added as shown

$$\begin{bmatrix} O_{z0} \\ O_{z1} \end{bmatrix} = \frac{I_u}{I_u + (I_{\bar{u}}) + I_{z0} + I_{z1}} \begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} \tag{3.29}$$

This additional term will show up as extra transistors in the product circuit, and will be dealt with in the next Section.

## 3.4 Low Voltage Sum Product Modules

### 3.4.1 Product Π functions

The construction of a low voltage Π function is an extension of the basic low voltage current multiplier [71]. A low voltage vector multiplier is shown in Fig. 3.15. Extra transistors are needed to keep the denominator of the product constant. The number of extra transistors added depends on the number of symbols used. Two input probability distributions of $M$ symbols each would require $M(M-1)$ extra transistors when compared to the vector multiplier shown in Fig. 3.6. These extra transistors add negligible area when $M$ is small. Notice that mirrored input current $I_{x,m}$ sources are shown to indicate their connectivity to the newly needed redundant transistors. The current output $I_{i,j}$ in the low voltage vector multiplier is given by

$$I_{i,j} = \frac{I_{x,i} I_{y,j}}{\sum_{i=1}^{m} I_{x,i} + \sum_{j=1}^{n} I_{y,j}} \tag{3.30}$$

These individual current branches are intermediate product terms which can be summed by connecting them together. Any unused terms can be tied to $V_{DD}$.

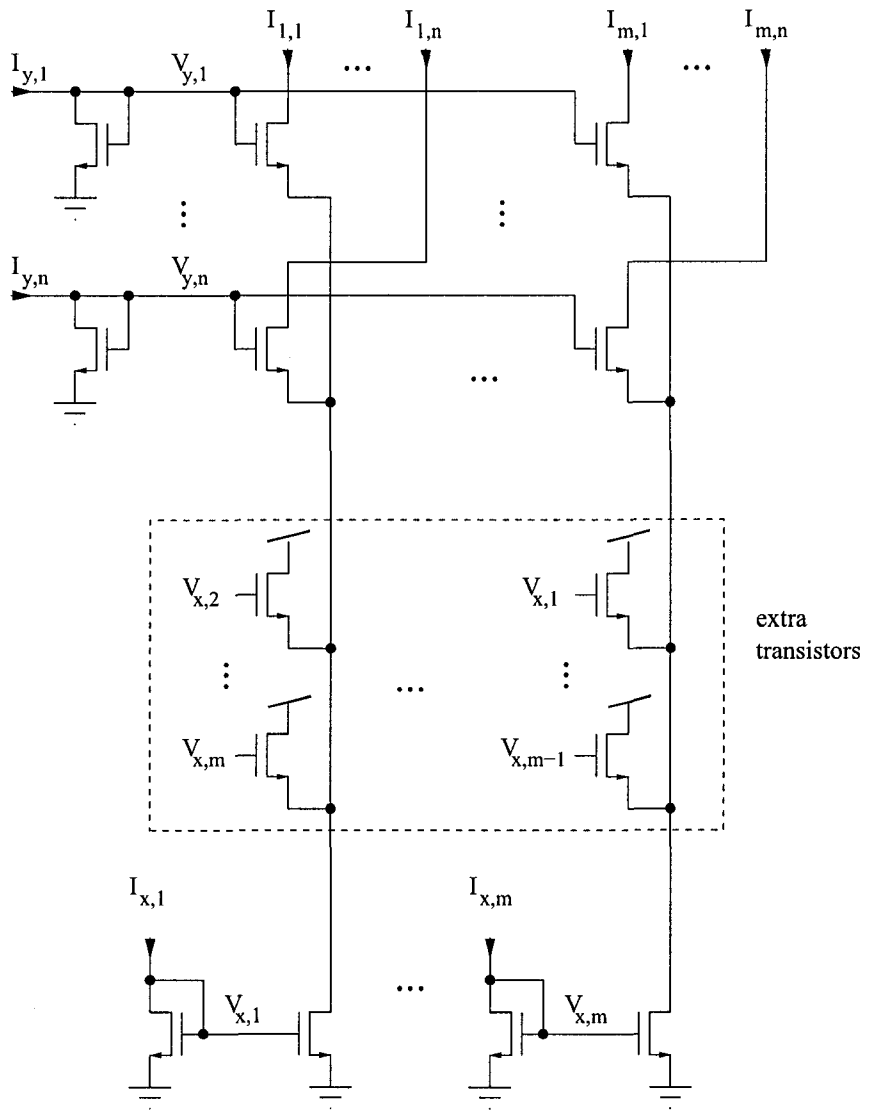



Figure 3.15: Low voltage current vector multiplier with added transistors to create constant denominator [71]

### 3.4.2 Normalizing

As discussed in Sec. 3.2.2, normalization is needed to ensure that the output currents sum to $I_u$ (a current which represents probability 1). In the low voltage sum product module, normalization also needs to perform current amplification since there is heavy attenuation on output currents. This is because the denominator of the product term contains the sum of two probability distributions. We use a P-type low voltage multiplier shown in Fig. 3.14 to perform both functions. Let us analyze its requirements.

The input-output current relationship is still represented by (3.28) where $m$ is the size ratio of M4, and $n$ are the size ratios of M3 and M5. From this, to amplify input currents

$$\frac{m \cdot I_u}{\left(\frac{m}{n}\right) I_u + I_{z0} + I_{z1}} > 1 \tag{3.31}$$

$$mI_u > \frac{m}{n} I_u + I_{z0} + I_{z1} \tag{3.32}$$

the sum of the input currents are at maximum $I_{z0} + I_{z1} = I_u/2$

$$mI_u > \frac{m}{n} I_u + \frac{I_u}{2} \tag{3.33}$$

$$m(1 - \frac{1}{n}) > \frac{1}{2} \tag{3.34}$$

$$m > \frac{n}{2(n-1)} \tag{3.35}$$

Therefore, if we make $m = n$, then any $m, n > 1.5$ will satisfy (3.35). Output currents will be larger than input currents preventing currents from one module to the next from sinking into the noise floor.

### 3.4.3 Factor Graph Nodes

The nodes of a factor graph can be implemented by using the circuit shown in Fig. 3.16. The equality node has output currents,

$$\begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} = k_e \begin{bmatrix} I_{x0} I_{y0} \\ I_{x1} I_{y1} \end{bmatrix} \tag{3.36}$$

and the check node has output currents,

$$\begin{bmatrix} I_{z0} \\ I_{z1} \end{bmatrix} = k_c \begin{bmatrix} I_{x0} I_{y0} + I_{x1} I_{y1} \\ I_{x0} I_{y1} + I_{x1} I_{y0} \end{bmatrix} \tag{3.37}$$

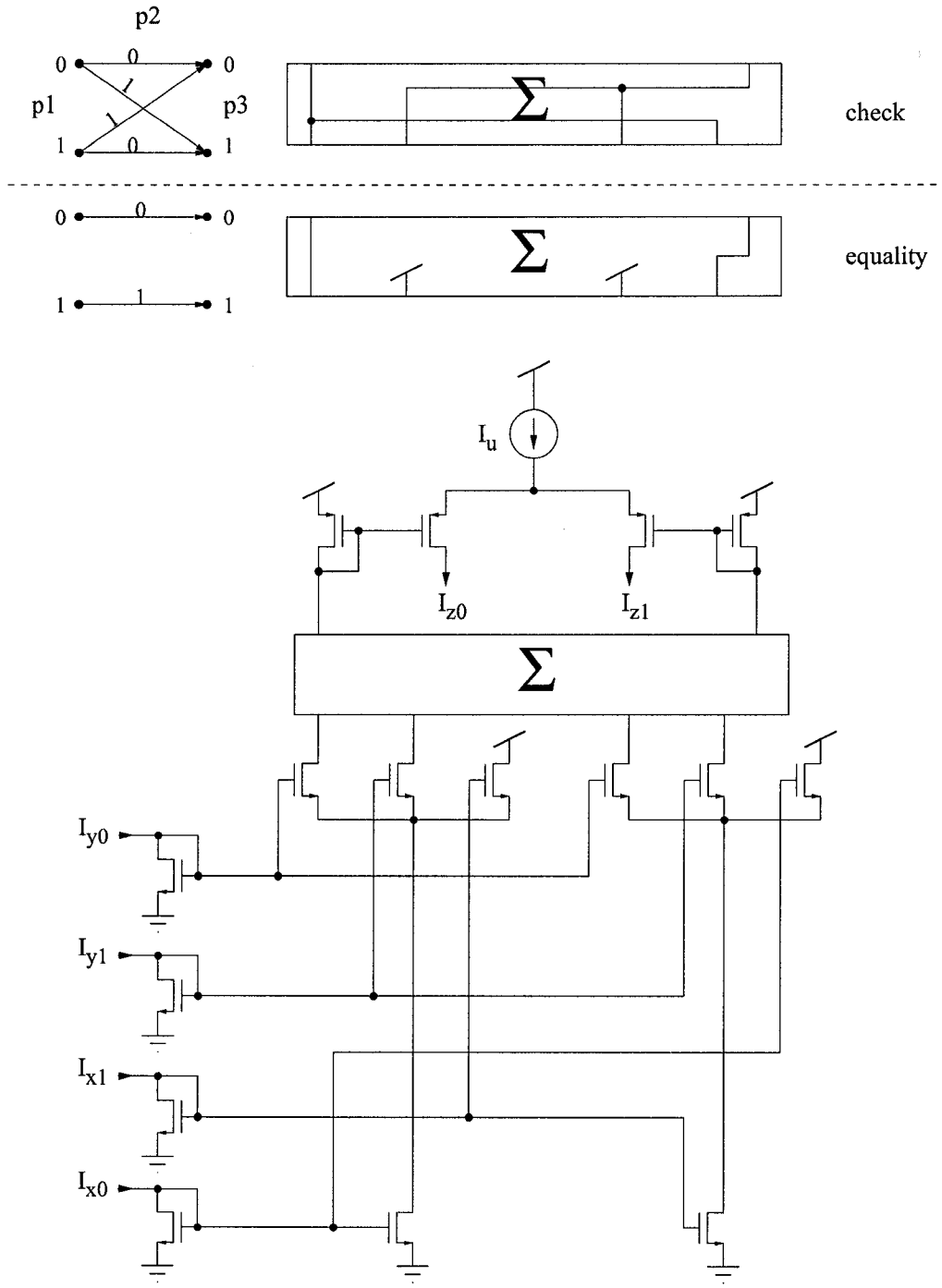



Figure 3.16: Factor graph node in one direction

where $k_e$ and $k_c$ are constants which depends on normalizer transistor sizing.

Two biasing parameters are used to ensure transistors within the node operate in weak inversion: the supply $V_{DD}$ and the unit current $I_u$. A conservative estimate of the minimum required supply $V_{DD}$ for a node constructed using the low voltage multiplier is [72]

$$V_{DD} \geq 0.211 + V_{TH,P} + \frac{U_T}{\kappa} ln\left(\frac{I_u}{100nA}\right) \tag{3.38}$$

where $V_{TH,P}$ is the threshold voltage of a PMOS device, $I_u$ is the unit current, and $\kappa$ is a process dependent parameter. If a node is manufactured in a typical $0.18\mu$m process and is biased at $I_u = 1\mu A$, a supply of $V_{DD} \geq 0.74V$ is needed to ensure correct operation. It was determined that this supply is about 0.4V less than a node constructed with a regular Gilbert multiplier.

In a factor graph decoder, messages are exchanged bidirectionally and nodes often have degree larger than 2. To get bi-directional functionality, 3 uni-directional nodes are placed in parallel. Bi-directional $n$-degree nodes are constructed by placing $n-2$ degree 3 nodes in series. This is summarized in Fig. 3.17 and and follows closely to the description provided by [30].

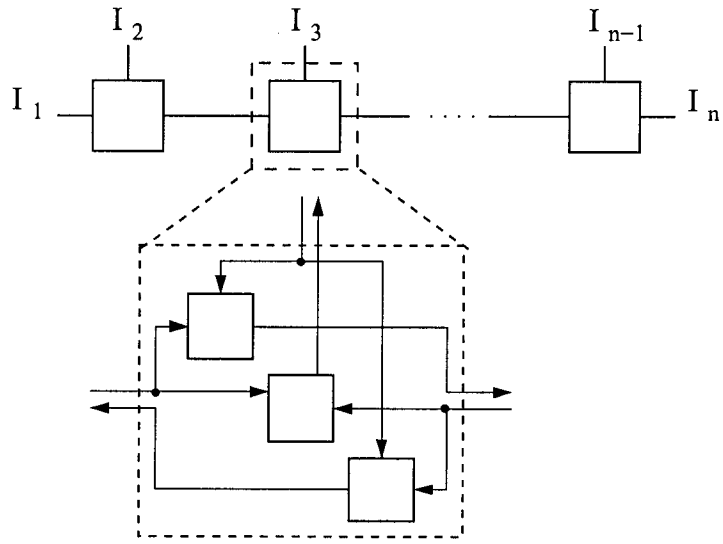



Figure 3.17: The construction of an $n$-degree node

We now examine the difference between the outputs of these circuits and ideal outputs as specified by the sum product algorithm. Even after normal-

ization, the outputs of these modules differ from ideal. This difference varies depending on $V_{DD}$ and $I_u$ assuming transistor sizes are fixed. We can use a 3 degree bi-directional node, as shown in Fig. 3.18, to sweep for current outputs. Each thick line indicates two wires – one for probability 1 and the other for 0. Now using SPICE BSIM v3.1 simulation, for every $I_{x0} = 0.01 \cdot I_u$ to $0.99 \cdot I_u$ we sweep $I_{y0} = 0.01 \cdot I_u$ to $0.99 \cdot I_u$ and vary $I_{x1}$ and $I_{y1}$ accordingly. For every possible product the outputs $I_{z0}$ and $I_{z1}$ are recorded and the difference between circuits and ideal is calculated as

$$\Delta LLR = log\left(\frac{I_{z0}}{I_{z1}}\right) - log\left(\frac{p_z(0)}{p_z(1)}\right) \tag{3.39}$$

where $I_z$ are realized output currents and $p_z$ are ideal output probability values. We plot this difference on a two-dimensional graph and call it the *error surface*. Such surfaces are shown in Figs. 3.19 for the equality node and 3.20 for the check node. These nodes were biased at $V_{DD} = 0.7V, I_u = 0.5\mu A$. The $\Delta LLR$ for the equality node ranges from -1 to +1 and for the check node, from about -0.3 to +0.3. Note that a 0 difference is slightly gray (it is not white) as indicated by the bar on the right side of the graphs.

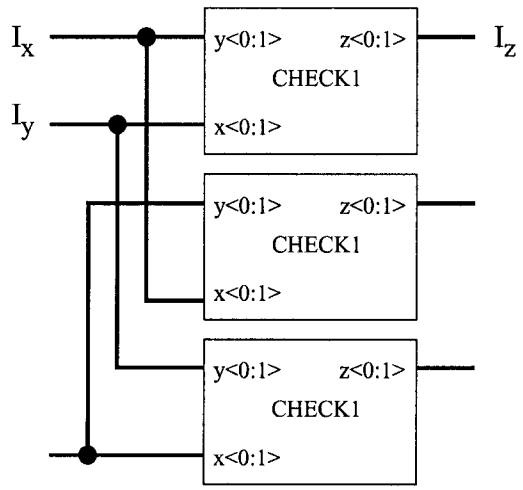



Figure 3.18: An example bi-directional node used for characterizing differences between circuits and ideal

The difference is most pronounced at the edges where realized values cannot keep up with their ideal counterparts (or vice versa). This effect is comparable to digital quantization of soft values. More importantly, the difference

is small in the middle region (where Px=0.5 and Py=0.5) where deviations could potentially flip the decision.

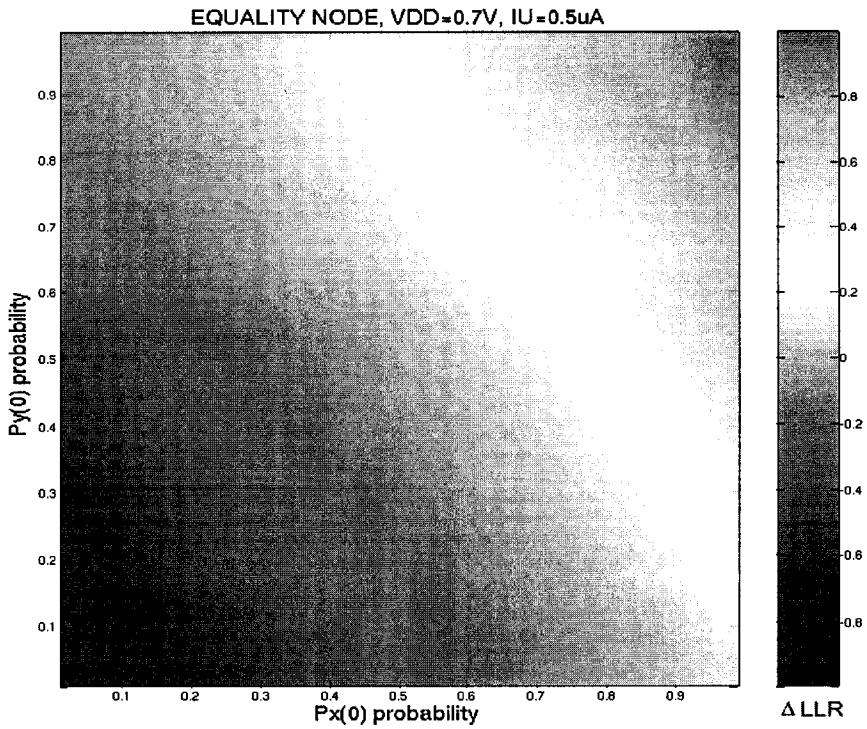



Figure 3.19: Equality node $\Delta LLR$ (VDD = 0.7V, $I_u = 0.5\mu$A)

Observe that the check and equality node have symmetry down both diagonal axes (from Px=0, Py=0 to Px=1, Py=1 and from Px=1, Py=0 to Px=0, Py=1). However, the equality node has negative symmetry for the axis going from Px=1, Py=0 to Px=0, Py=1. By negative symmetry, we mean the difference on one side of the axis is negative on the other. Now look at (3.36), (3.37), and Fig. 3.16. The explanation for this might be that the check circuit uses all intermediate current terms and that the combination of these terms are inherently symmetric. The equality circuit, on the other hand, discards some intermediate terms which are affected by the connection of transistors shown in Fig. 3.16. This is only speculation, however, and more study is needed to understand the causes of these differences.

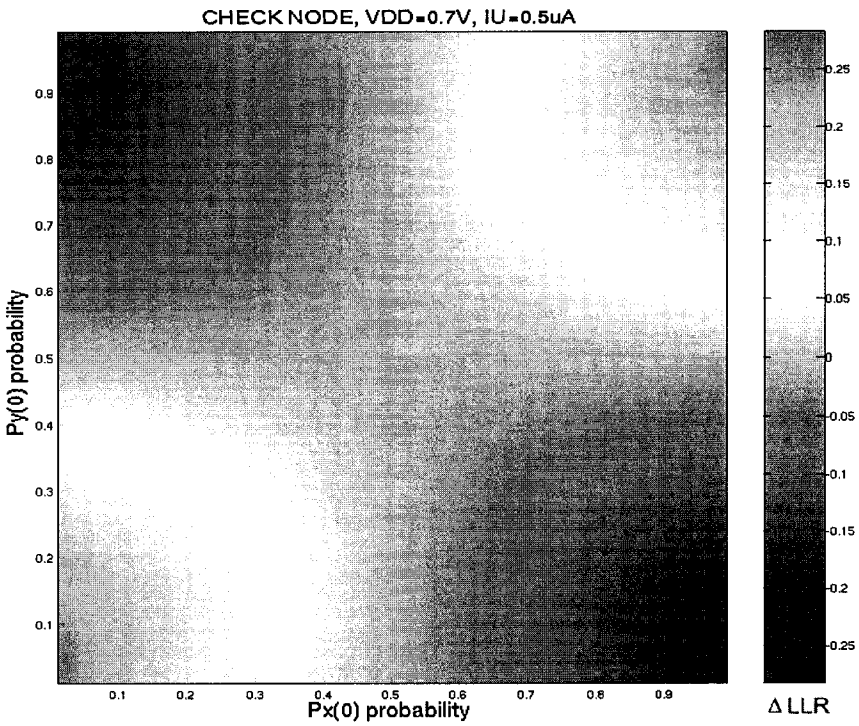



Figure 3.20: Check node $\Delta LLR$ (VDD = 0.7V, $I_u = 0.5\mu$A)

## 3.5 Analog Effects

Recall that the analysis used to arrive at the output equations in Sec. 3.3.3 assumes two things. First, assuming local temperature matching, the $U_T$ terms gets cancelled out in the process thereby eliminating thermal effects. Second, the cancellation of $I_{\square}(V_g)$ terms removes the body effect. These results apply equally to the Gilbert multiplier and its modified low voltage counterpart. Therefore, it might be safe to say that inaccurate sum product operation observed from within the analog decoder is mainly due to LLR differences mentioned in the previous section and *device mismatch.*

Device mismatch happens when two identically drawn transistors are slightly different due to process variations. These differences affect values such as $W$, $L$, $V_{TH}$ and $I_d$. On the subject of mismatch, there has been much discussion. The first argument follows the philosophy of Mead [49] in realizing that analog decoders are very much like neural networks. While each sum product module is less than perfect, overall system-level accuracy is still achieved through par-

allelism and high connectivity. Second, the decoding process can be thought of as a non linear process where dirty bits from the channel get *cleaned* from input to output [42]. Finally, we can think of the decoder as a channel value ADC which, no matter what happens in its intermediate stages, the output will either be a 0 or 1. So unless mismatch is high, its effects will be masked.

In fact, when local matching is good and global matching is bad, the BER performance is still nearly identical to the ideal simulated curve [44, 25, 20]. However, [25] indicated that poor local mismatch could be detrimental to longer length Turbo decoders using that particular architecture.

A specific mismatch study related to decoders constructed with the low voltage multiplier was not done. However, previous mismatch studies will be discussed to give an idea of how mismatch affects BER. In this section, we will give a brief overview of known mismatch studies.

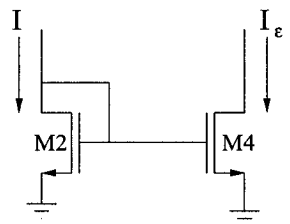



Figure 3.21: Mismatch can be characterized as an error in the mirrored current

In general these studies involved modeling mismatch as an error in the mirrored current as shown in Fig. 3.21. The mirrored current can be written as

$$I_\epsilon = (1+\epsilon) I_{ideal} \tag{3.40}$$

where $\epsilon$ is a random variable used to represent mismatch.

A probability based analysis was done by [44] looking at the possible contribution of errors inside the product function $\Pi$ of a sum product module. These errors include (1) diode connected transistors, and (2) the internal transistors of the Gilbert matrix column not including the bias current sources, as shown in Fig. 3.22. The output current with mismatch $Iz'_{i,j}$ is

$$Iz'_{i,j} = \frac{Ix_i \cdot Iy_j \cdot (1+\epsilon_{i,j})(1+\epsilon_j)}{\sum_{k=1}^{n}(1+\epsilon_{i,j})(1+\epsilon_k) Iy_k} \tag{3.41}$$

where $i = 1, \ldots, m$ and $j = 1, \ldots, n$ are the column and row locations in an $m \times n$ multiplier, and each $\epsilon$ are independent random variables with mean 0 and variance $\sigma_\epsilon$. With $\sigma_\epsilon = 0.15$, the worst simulated (44, 22, 8) decoder deviated 1.75dB from ideal at $BER = 10^{-3}$ [46]. The best decoder performed better than ideal indicating a wide variation.

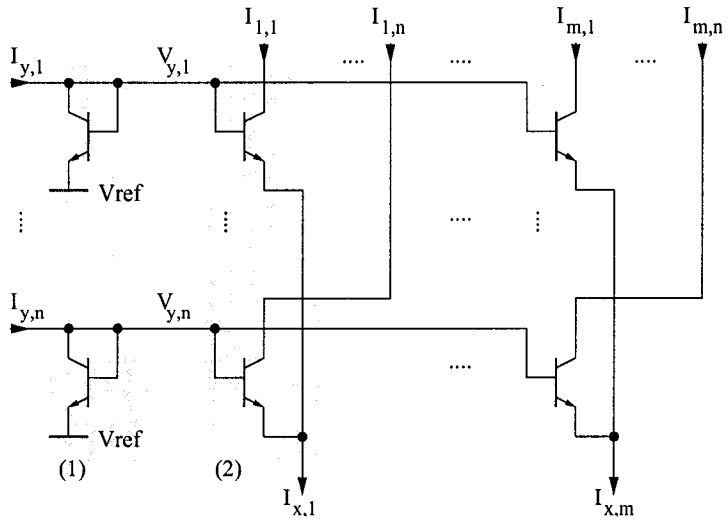



Figure 3.22: Mismatch analysis in the Gilbert vector multiplier [44]

A recent mismatch study done by [73], used similar analysis to that of [46] applied to a $2 \times 2$ CMOS sum product module as shown in Fig. 3.23. Internal transistor column (1) and current source (2) mismatch were included to derive the output:

$$Iz'_{i,j} = \frac{Ix_i \cdot Iy_j \cdot (1+\epsilon_j)(1+\epsilon_{\bar{i},j})}{Ix_i(1+\epsilon_{\bar{i},j}) + Ix_{\bar{i}}(1+\epsilon_{i,j})} \tag{3.42}$$

where $i, j \in \{0, 1\}$ and $\bar{i}, \bar{j}$ are complements to $i, j$. Density evolution analysis was then done to estimate the effects of mismatch on an arbitrarily large LDPC decoder ($n \rightarrow \infty$). With mismatch $\sigma_\epsilon = 0.2$, the performance loss in a very large decoder constructed with sum product modules is around 0.1dB. This loss grows without bound as $\sigma_\epsilon > 0.3$.

An interesting study was done by [20] to measure an (8,4) Hamming decoder constructed with discrete BiCMOS sum product components. Each chip contained either a 3 degree equality or 3 degree check node. While each component varied in performance, the overall decoder BER was almost identical to

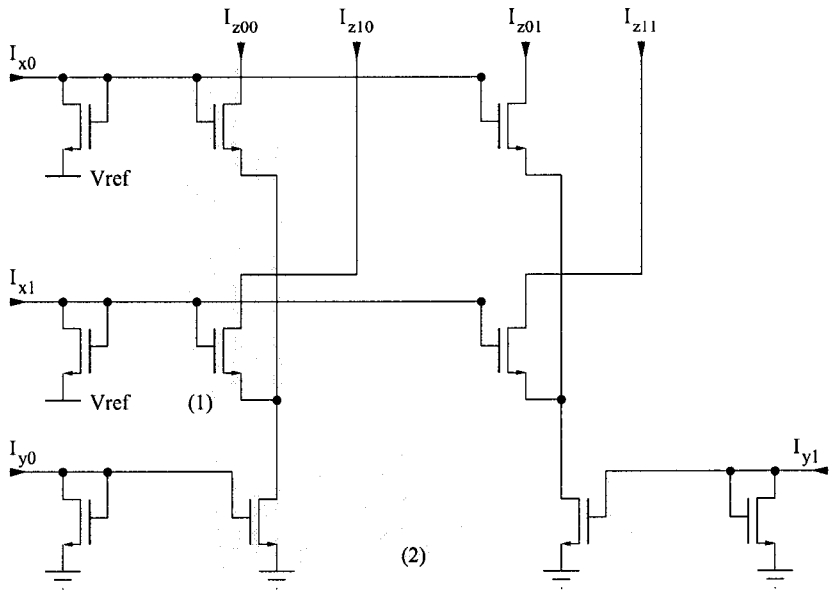



Figure 3.23: Mismatch analysis in a CMOS 2x2 multiplier [73]

ideal down to $BER = 10^{-3}$. In fact, to mitigate mismatch effects, these sum product modules were constructed using large transistor sizes (W = 144$\mu$m and L = 5.6$\mu$m for both PMOS and NMOS devices). These results were later replicated with a 0.18$\mu$m decoder chip [21] with smaller transistor sizes (W = 10$\mu$m, L = 1$\mu$m for NMOS) and (W = 30$\mu$m, L = 1$\mu$m for PMOS).

The effect of mismatch on BER can be summarized as being architecture dependent. Mismatch effects on BER are less noticeable for small decoders (short length codes) but could be performance limiting on longer length decoders.

Additional analysis done by [14] suggests that other analog effects, such as thermal noise, flicker noise, and channel length modulation, are negligible compared to device mismatch. These effects, however, might again become large as the decoder length grows. We therefore say that implementation losses are mainly due to device mismatch, strongly inverted transistors, and I/Os.

## 3.6 Chapter Summary

In this chapter, a background on previous and related analog decoding work was presented. That general description was then narrowed down to the specific architecture used in the implementations of this thesis. Then an example of how to construct factor graph nodes using this particular multiplier were presented. Its minimum supply voltage and comparison to ideal were quantified wrapping up with a discussion of analog effects. We now proceed with two (8,4) Hamming decoder designs.

# Chapter 4

# Implementation

This chapter describes the implementation of two sub-1V analog decoders. The first section presents an (8,4) Hamming decoder based on a factor graph. The second section presents an (8,4) Hamming decoder based on a tail biting trellis. The third section describes the input-output (I/O) circuits used in both of the above decoders. A final section presents SPICE simulations showing the correct operation of both decoders.

## 4.1 An (8, 4) Hamming factor graph decoder

The **G** and **H** matrices of an (8,4) extended Hamming code are shown in (4.1). Their corresponding source word to codeword mapping is shown in Table 4.1. Extra rows in **H** (derived by summing existing rows) were added to increase redundancy in the decoding algorithm. This extra redundancy is needed to drive down the BER at high SNRs [44]. Fig. 4.1 shows simulation results using three belief propagation programs. Programs (1) and (2) are llr_pearl.c and pearl.c, both are from [52]. Program (3) was created by our research group. Programs (1) and (3) perform message passing using LLRs, while program (2) passes actual probabilities. The BER for an 8×4 matrix using program (1) performs worse than uncoded BPSK while the BER for the 8×8 matrices closely follow the ML curve. There is a loss of 0.2dB for programs (1) and (2) at high SNRs, and a loss of 0.4dB for program (3).

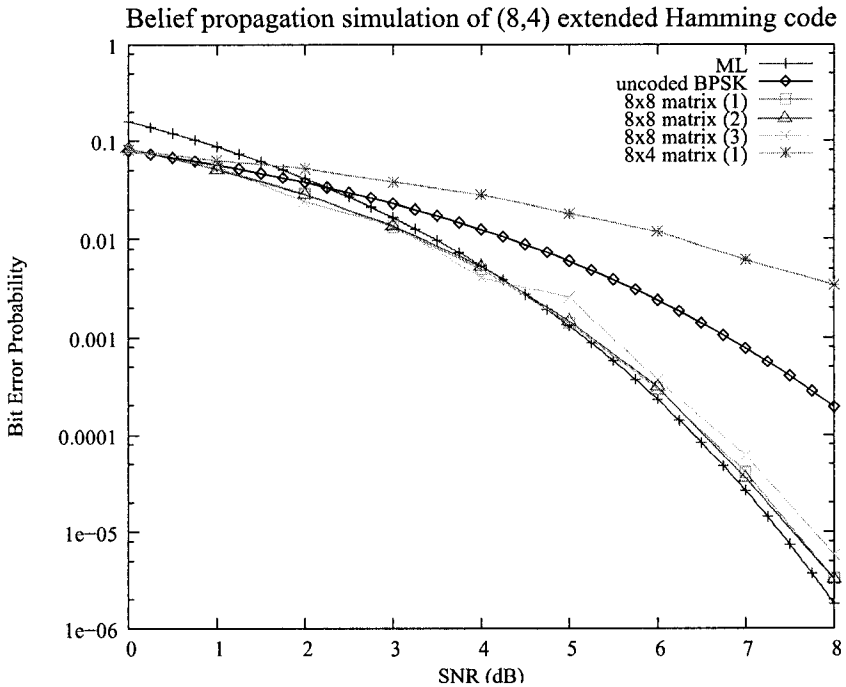



Figure 4.1: Belief propagation simulation of 8x8 and 8x4 **H** matrices of an (8,4) Hamming code using three different programs (1), (2), and (3)

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{4.1}$$

The **H** matrix is translated to the factor graph of Fig. 4.2. Each row of the **H** matrix is represented by a parity check node operating on values coming from several equality nodes representing variables. Larger equality nodes are used to calculate extrinsic information from check nodes, while smaller equality nodes include intrinsic information to calculate decoded outputs. The connections between nodes are bidirectional unless indicated by arrows.

The factor graph is then mapped directly into circuits [53] as shown in Fig. 4.3. The realized decoder is made up of voltage in, voltage out modules or blocks with the exception of EQUALITY1_IOUT. Input differential voltages ($V_{in}(0)$, $V_{in}(1)$) which represent probabilities enter EQUALITY5 and EQUAL-

Table 4.1: Source word to code word mapping of the factor graph decoder

| u | x | u | x |
|---|---|---|---|
| 0000 | 0000 0000 | 1000 | 1000 1011 |
| 0001 | 0001 0111 | 1001 | 1001 1100 |
| 0010 | 0010 1101 | 1010 | 1010 0110 |
| 0011 | 0011 1010 | 1011 | 1011 0001 |
| 0100 | 0100 1110 | 1100 | 1100 0101 |
| 0101 | 0101 1001 | 1101 | 1101 0010 |
| 0110 | 0110 0011 | 1110 | 1110 1000 |
| 0111 | 0111 0100 | 1111 | 1111 1111 |

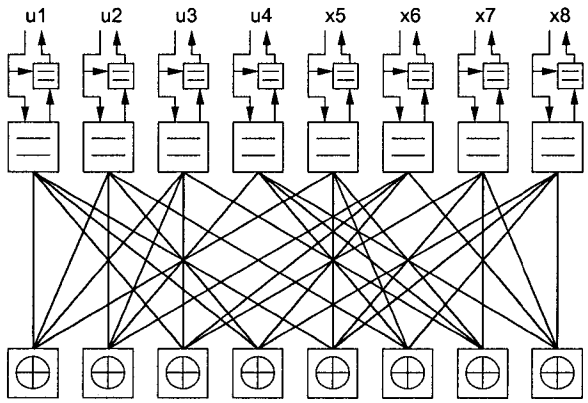



Figure 4.2: The factor graph of an (8, 4) extended Hamming code

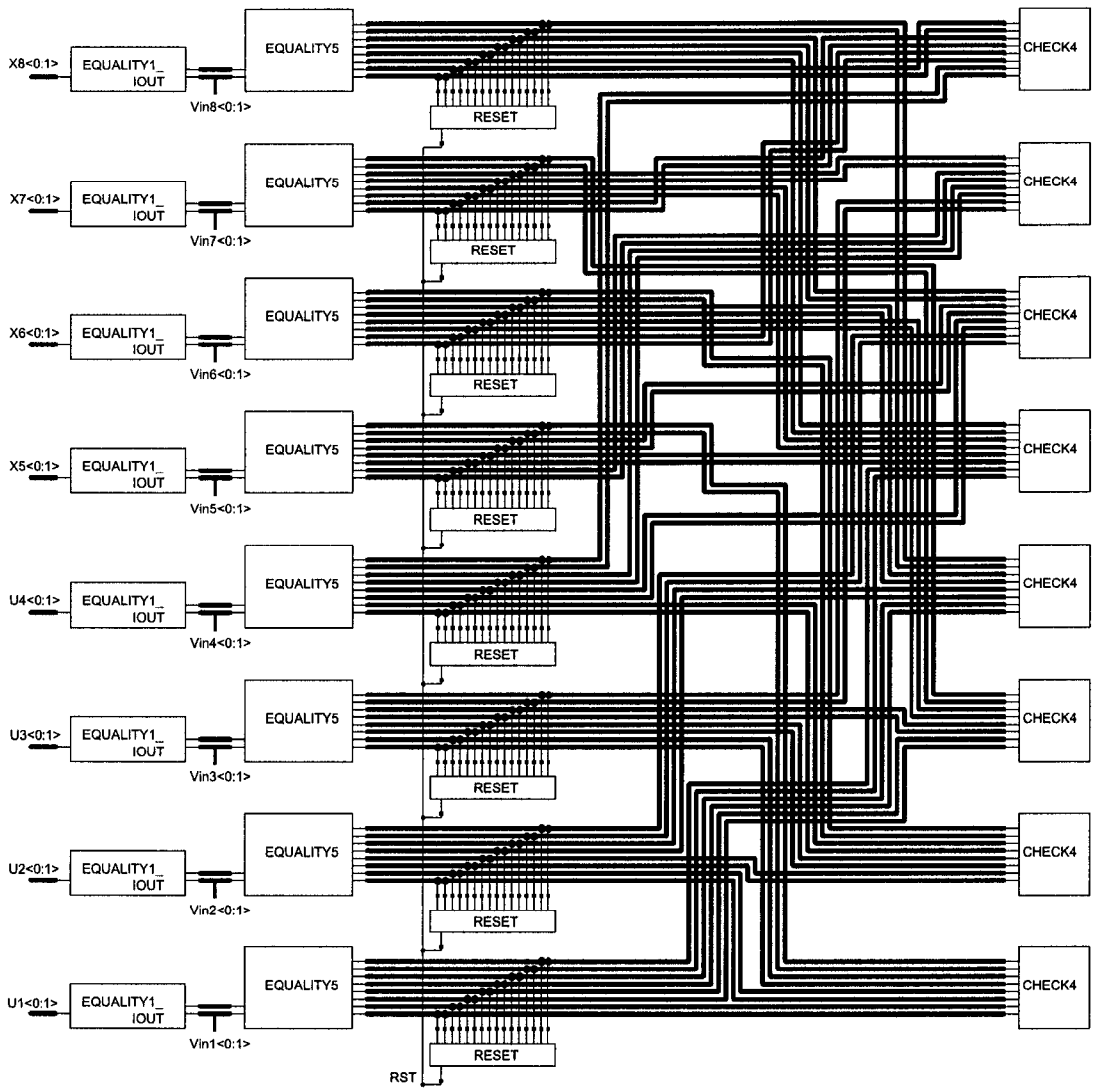



Figure 4.3: The (8, 4) Hamming factor graph decoder

ITY1_IOUT. EQUALITY5 is a degree 5 bi-directional module. CHECK4 is a degree 4 bi-directional module. Decoding takes place by voltage (or current) exchange between EQUALITY5 and CHECK4 modules. Extrinsic values are passed to EQUALITY1_IOUT and included with intrinsic values to form the decoder outputs (coming out of the top). The EQUALITY1_IOUT module is a unidirectional module with current outputs. These currents are fed into output comparators for final bit decisions (see Sec. 4.3). After each codeword has been processed, the interconnections between EQUALITY5 and CHECK4 are initialized by pass transistors inside of the RESET blocks.

We will take a top-down approach in describing the details of each block. The EQUALITY5 node is constructed by placing three EQUALITY3 nodes in parallel, as shown in Fig. 4.9. Each EQUALITY3 node is in turn made up of three unidirectional nodes, EQUALITY1, as shown in Fig. 4.8. The EQUALITY1 node is shown at the transistor level in Fig. 4.7. This circuit is slightly modified from the general structure of Fig. 3.16. The input diode connected transistors are moved to the outputs turning the module into voltage in, voltage out to make connection between nodes easier. Similarly, the CHECK4 and CHECK3 nodes follow the same design procedure and are shown in Figs. 4.6 and 4.5, respectively. The CHECK1 node is shown at the transistor level in Fig. 4.4. The EQUALITY1_IOUT node is simply an EQUALITY1 node without output diode connected transistors, as shown in Fig. 4.10. The RESET block is made up of pass transistors as shown in Fig. 4.11.

Almost all transistors used in the decoding network were sized as $W/L = 0.5\mu m/0.25\mu m$ (slightly larger than the minimum allowable size ratio). An exception to this rule occurs in the normalizers where the mirroring $I_u$ transistor and the two transistors directly beneath it were made 3.5/0.25 to meet the requirement of $m, n = 7 > 1.5$. These transistor sizes were largely influenced by the equality nodes. They have more attenuation due to intermediate current terms being discarded. This is similar to trellis mappings, where larger normalizing ratios are needed for modules which discard many current terms.

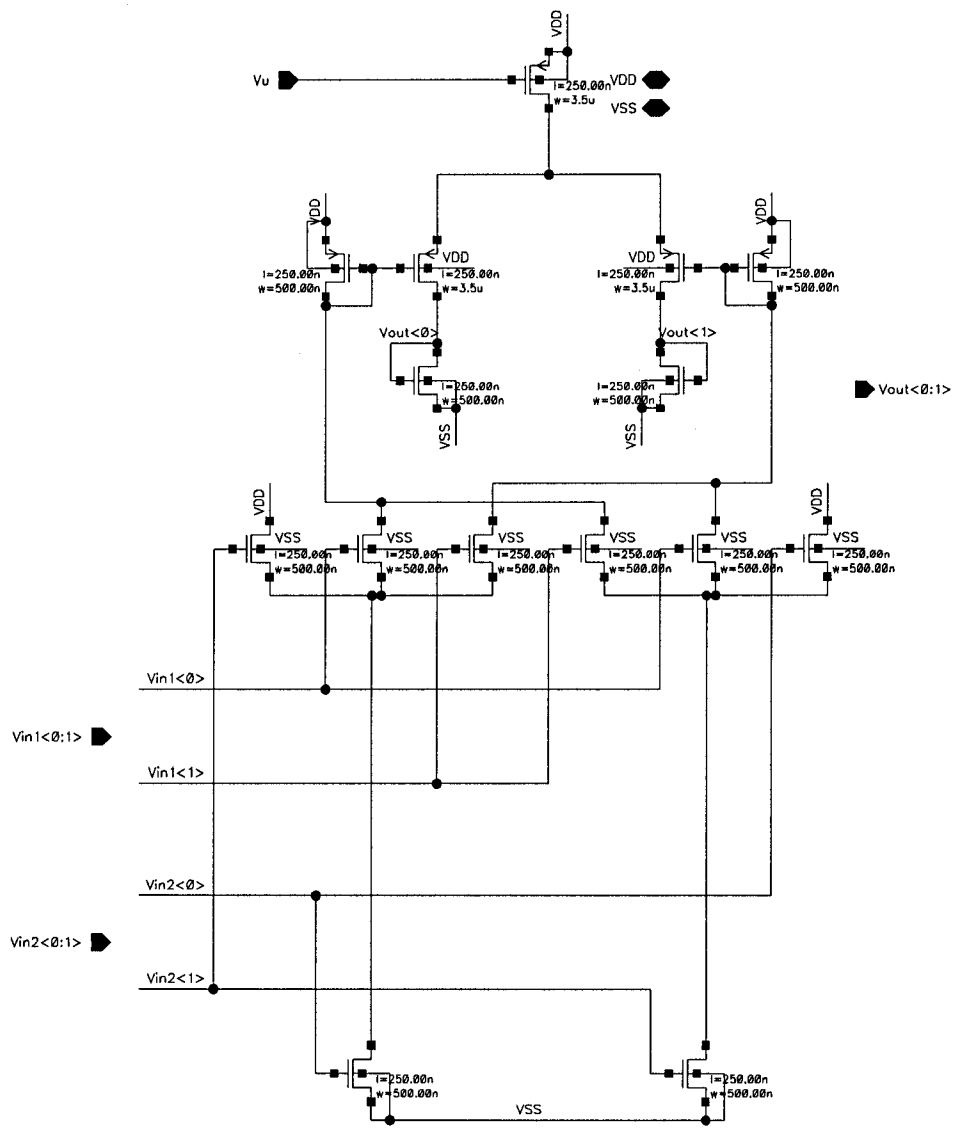



Figure 4.4: CHECK1: unidirectional check node

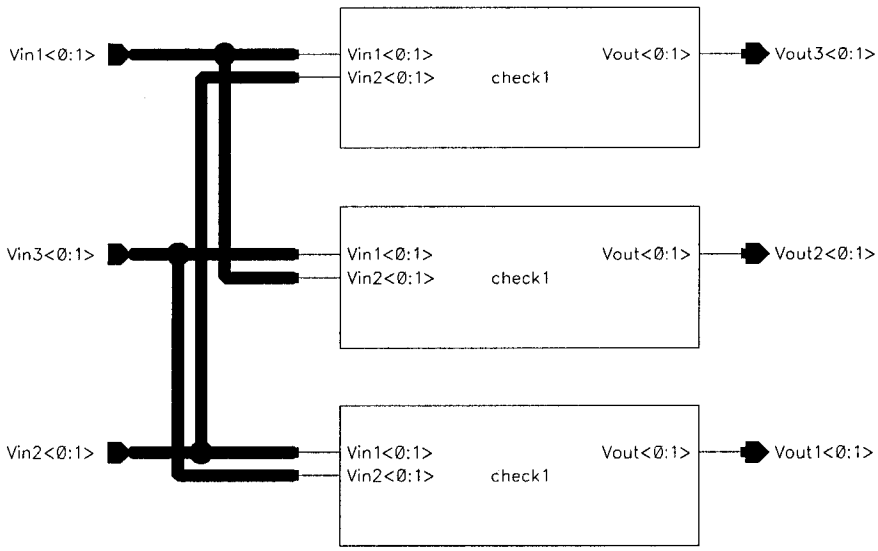



Figure 4.5: CHECK3: bidirectional 3-port check node

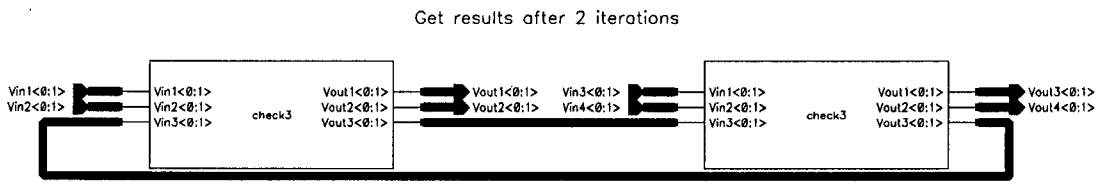



Figure 4.6: CHECK4: bidirectional 4-port check node

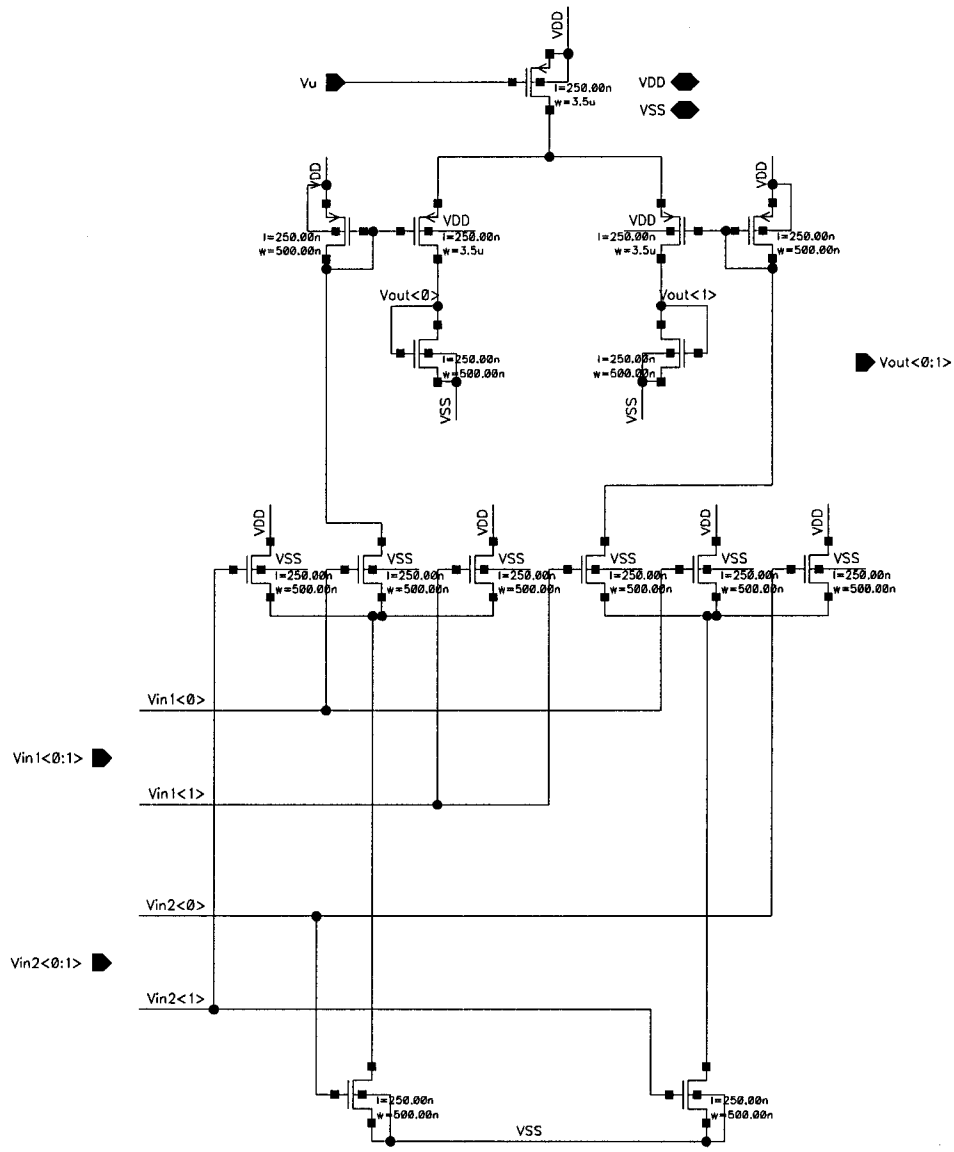



Figure 4.7: EQUALITY1: unidirectional equality node

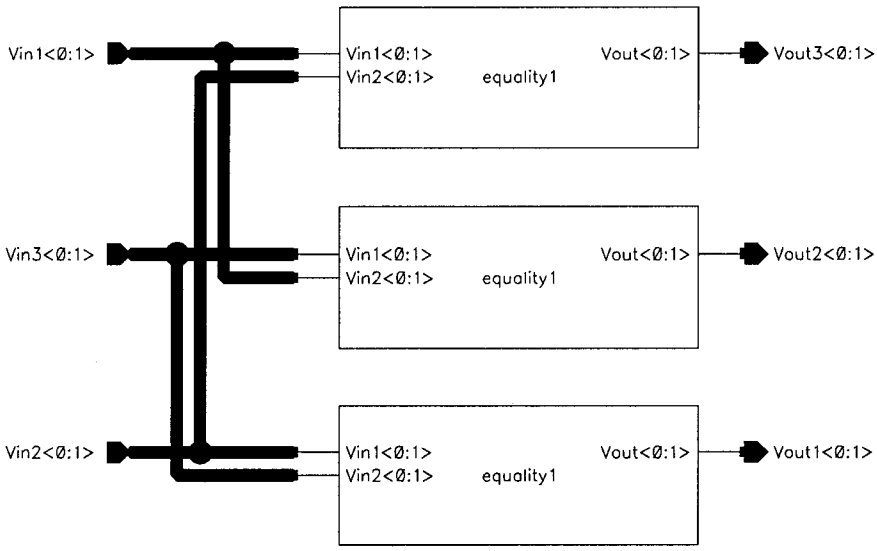



Figure 4.8: EQUALITY3: bidirectional 3-port equality node

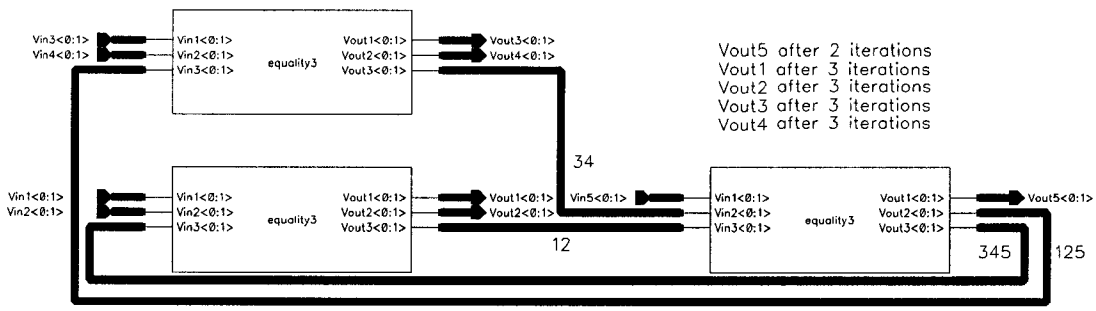



Figure 4.9: EQUALITY5: bidirectional 5-port equality node

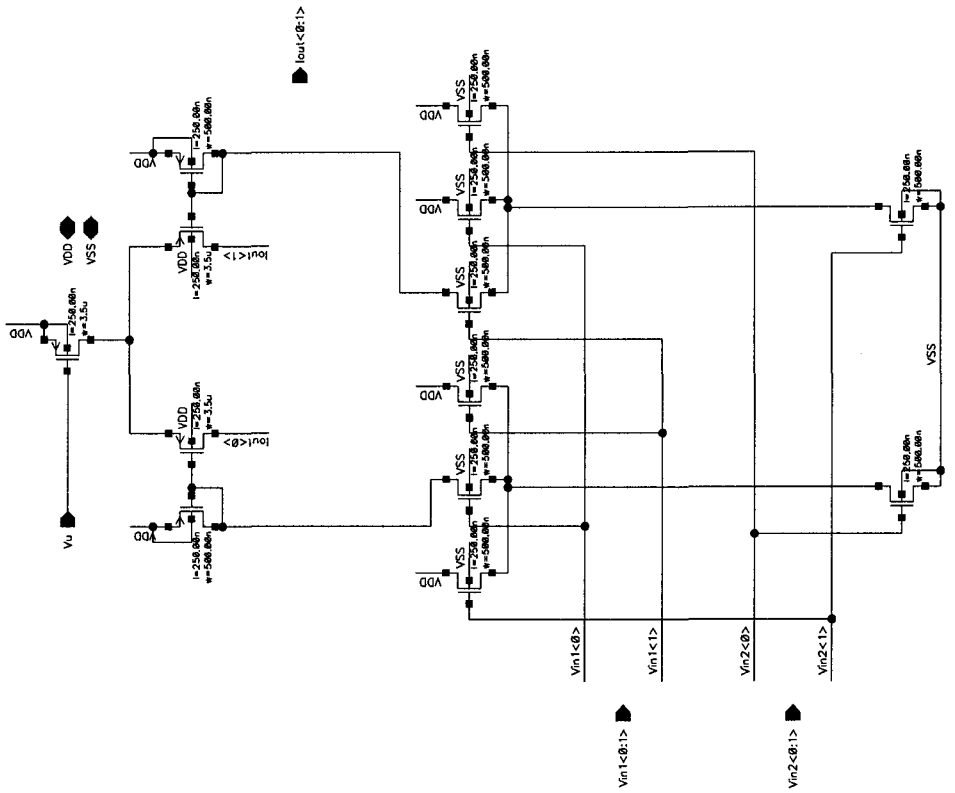



Figure 4.10: EQUALITY1_IOUT: unidirectional equality with current outputs

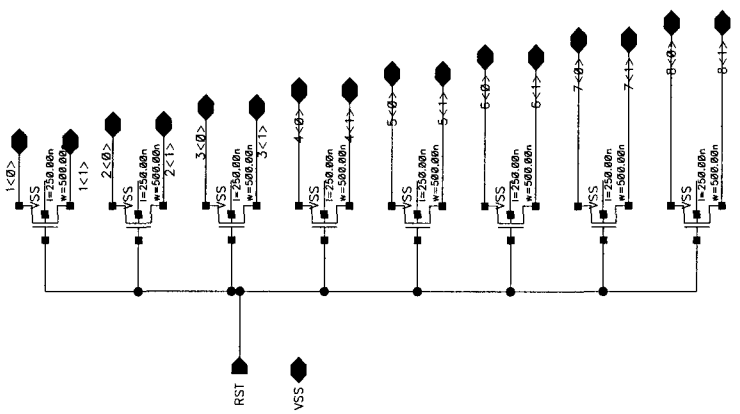



Figure 4.11: RESET: pass transistors used for equalizing probabilities

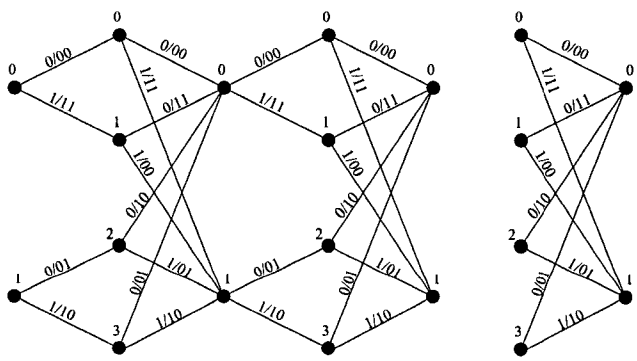
## 4.2 An (8, 4) Hamming trellis graph decoder

The (8, 4) extended Hamming tail biting trellis is shown in Fig. 4.12. Its trellis is time varying. The last trellis section connects to the first section. A valid code word starts and ends in the *same state.* The generator matrix of such a code is non systematic and therefore sourcewords might not show up in codewords as seen in Table 4.2

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{4.2}$$

Table 4.2: Source word to code word mapping of the trellis graph decoder

| u | x | u | x |
|---|---|---|---|
| 0000 | 0000 0000 | 1000 | 1111 0000 |
| 0001 | 0110 0011 | 1001 | 1001 0011 |
| 0010 | 0000 1111 | 1010 | 1111 1111 |
| 0011 | 0110 1100 | 1011 | 1001 1100 |
| 0100 | 0011 0110 | 1100 | 1100 0110 |
| 0101 | 0101 0101 | 1101 | 1010 0101 |
| 0110 | 0011 1001 | 1110 | 1100 1001 |
| 0111 | 0101 1010 | 1111 | 1010 1010 |




(a) trellis

(b) section 2

Figure 4.12: Minimal tail-biting trellis for (8, 4) Hamming code

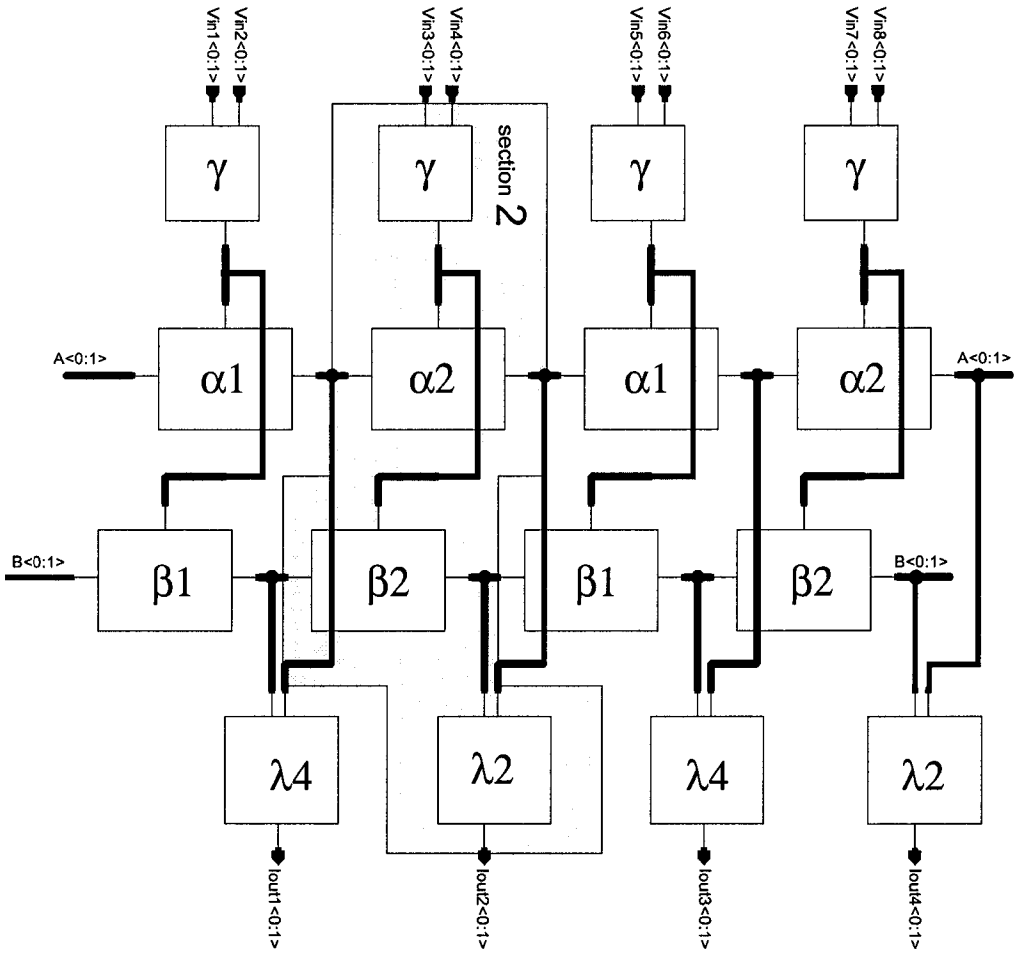



Figure 4.13: Block diagram of an (8, 4) Hamming trellis graph decoder

The construction of this decoder maps directly from trellis into circuits implementing the BCJR algorithm [4]. However, the outputs from a tail biting trellis are considered to be only *approximate* APPs [44].

An overall decoder block diagram is shown in Fig. 4.13. Each trellis section is implemented as four modules ($\gamma$, $\alpha$, $\beta$, and $\lambda$). Each module performs one metric type calculation. Adjacent channel observations $p(y|x)$ come into the top $\gamma$ block and are combined into channel metrics. These channel metrics are used in the recursive calculations of $\alpha$ and $\beta$ values. The outputs of $\alpha$ and $\beta$ blocks are then used to calculate approximate APPs. All blocks are voltage in, voltage out with the exception of the $\lambda$ block, which has current out to feed into output comparators.

Based on (2.24), the channel metric in each trellis section is calculated as

$$\gamma_k(p,q) = p(Y_k|X_k) \tag{4.3}$$

$$= p(y_i|x_i)p(y_{i+1}|x_{i+1}) \quad i = 1,3,5,7 \tag{4.4}$$

where the notation $X$ and $Y$ is used to indicate that the received word actually consists of two bits ($X = x_i, x_{i+1}$ and $Y = y_i, y_{i+1}$) and they need to be combined. The a priori probability is assumed to be equal for all bits and is not included in the calculation. The implementation for this equation is shown in Fig. 4.14. Two adjacent channel values L1<0:1> = $p(y_i|x_i)$ and L2<0:1> = $p(y_{i+1}|x_{i+1})$ are combined to get four distinct output combinations: Y<0> = $\gamma(0,0)$, Y<1> = $\gamma(0,1)$, Y<2> = $\gamma(1,0)$, and Y<3> = $\gamma(1,1)$.

The implementations of $\alpha$ and $\beta$ blocks depend on the specific trellis section. There are only two unique trellis sections and we label them as 1 and 2 since the first is the same as the third and the second is the same as the fourth.

Now given trellis section 2 as shown in Fig. 4.12(b) and correspondingly in Fig. 4.13, the forward recursive metrics, $\alpha_k$, are calculated using (2.22). The resulting two $\alpha_k$ equations for the second trellis section are

$$\alpha_k(0) = \alpha_{k-1}(0)\gamma_k(0,0) + \alpha_{k-1}(1)\gamma_k(1,1) + \alpha_{k-1}(2)\gamma_k(1,0) + \alpha_{k-1}(3)\gamma_k(0,1) \tag{4.5}$$

$$\alpha_k(1) = \alpha_{k-1}(0)\gamma_k(1,1) + \alpha_{k-1}(1)\gamma_k(0,0) + \alpha_{k-1}(2)\gamma_k(0,1) + \alpha_{k-1}(3)\gamma_k(1,0) \tag{4.6}$$

The circuit that implements the above equations is shown in Fig. 4.15. The outputs are Out<0:1> = $\alpha_k(0)$, $\alpha_k(1)$ given inputs of Y<0:3> = $\gamma_k(0,0)$, $\gamma_k(0,1)$, $\gamma_k(1,0)$, $\gamma_k(1,1)$ and In<0:3> = $\alpha_{k-1}(0)$, $\alpha_{k-1}(1)$, $\alpha_{k-1}(2)$, $\alpha_{k-1}(3)$. The connectivity is labeled on each transistor.

The backward recursive metrics, $\beta_{k-1}$, are calculated using (2.23). The resulting four $\beta_{k-1}$ equations for the second trellis section are

$$\beta_{k-1}(0) = \beta_k(0)\gamma_k(0,0) + \beta_k(1)\gamma_k(1,1) \quad (4.7)$$

$$\beta_{k-1}(1) = \beta_k(0)\gamma_k(1,1) + \beta_k(1)\gamma_k(0,0) \quad (4.8)$$

$$\beta_{k-1}(2) = \beta_k(0)\gamma_k(1,0) + \beta_k(1)\gamma_k(0,1) \quad (4.9)$$

$$\beta_{k-1}(3) = \beta_k(0)\gamma_k(0,1) + \beta_k(1)\gamma_k(1,0) \quad (4.10)$$

These equations are realized using the circuit shown in Fig. 4.16. The outputs are Out<0:3> = $\beta_{k-1}(0)$, $\beta_{k-1}(1)$, $\beta_{k-1}(2)$, $\beta_{k-1}(3)$ given inputs of Y<0:3> = $\gamma_k(0,0)$, $\gamma_k(0,1)$, $\gamma_k(1,0)$, $\gamma_k(1,1)$ and In<0:3> = $\beta_k(0)$, $\beta_k(1)$, $\beta_k(2)$, $\beta_k(3)$.

Similar equations of $\alpha_k$ and $\beta_{k-1}$ metrics for trellis section 1 can be mapped into Figs. 4.17 and 4.18 respectively.

Finally, the APPs are calculated according to (2.25) and (2.26). The combined calculations over two states are

$$p(u_i = 0|\mathbf{y}) = \alpha_i(0)\beta_i(0) \quad (4.11)$$

$$p(u_i = 1|\mathbf{y}) = \alpha_i(1)\beta_i(1) \quad (4.12)$$

where $i \in \{1,3\}$. The calculations over four states are

$$p(u_j = 0|\mathbf{y}) = \alpha_j(0)\beta_j(0) + \alpha_j(2)\beta_j(2) \quad (4.13)$$

$$p(u_j = 1|\mathbf{y}) = \alpha_j(1)\beta_j(1) + \alpha_j(3)\beta_j(3) \quad (4.14)$$

where $j \in \{2,4\}$. These equations are realized by the circuits in Figs. 4.19 and 4.20, respectively.

**Transistor sizing**

All transistors in the multiplying matrix were sized to $W/L = 0.5\mu m/0.25\mu m$. Many unused intermediate terms were needed to yield a constant denominator. Normalizing transistors were sized upwards to $n = 10\mu m/0.25\mu m$ and generally $m \neq n$.

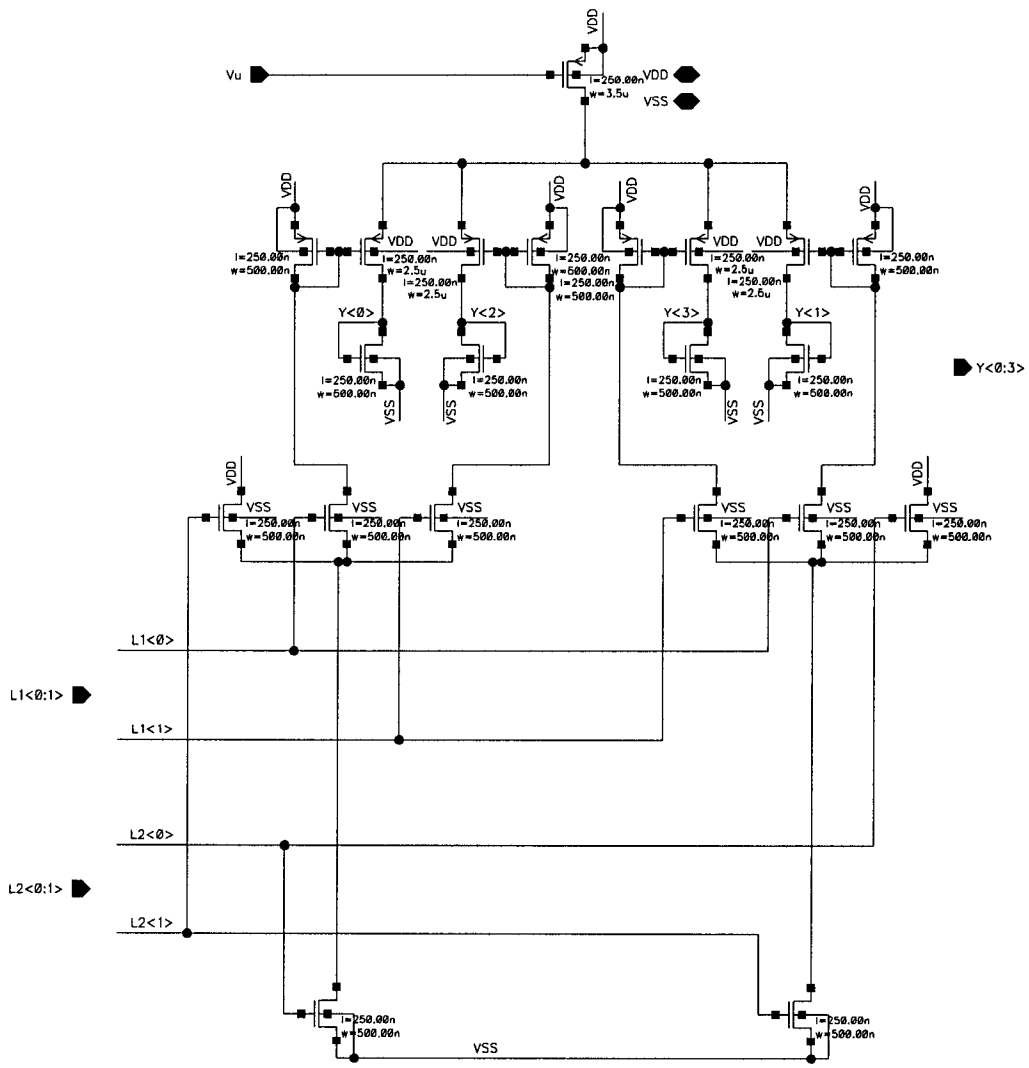



Figure 4.14: $\gamma$ : channel metric calculation block

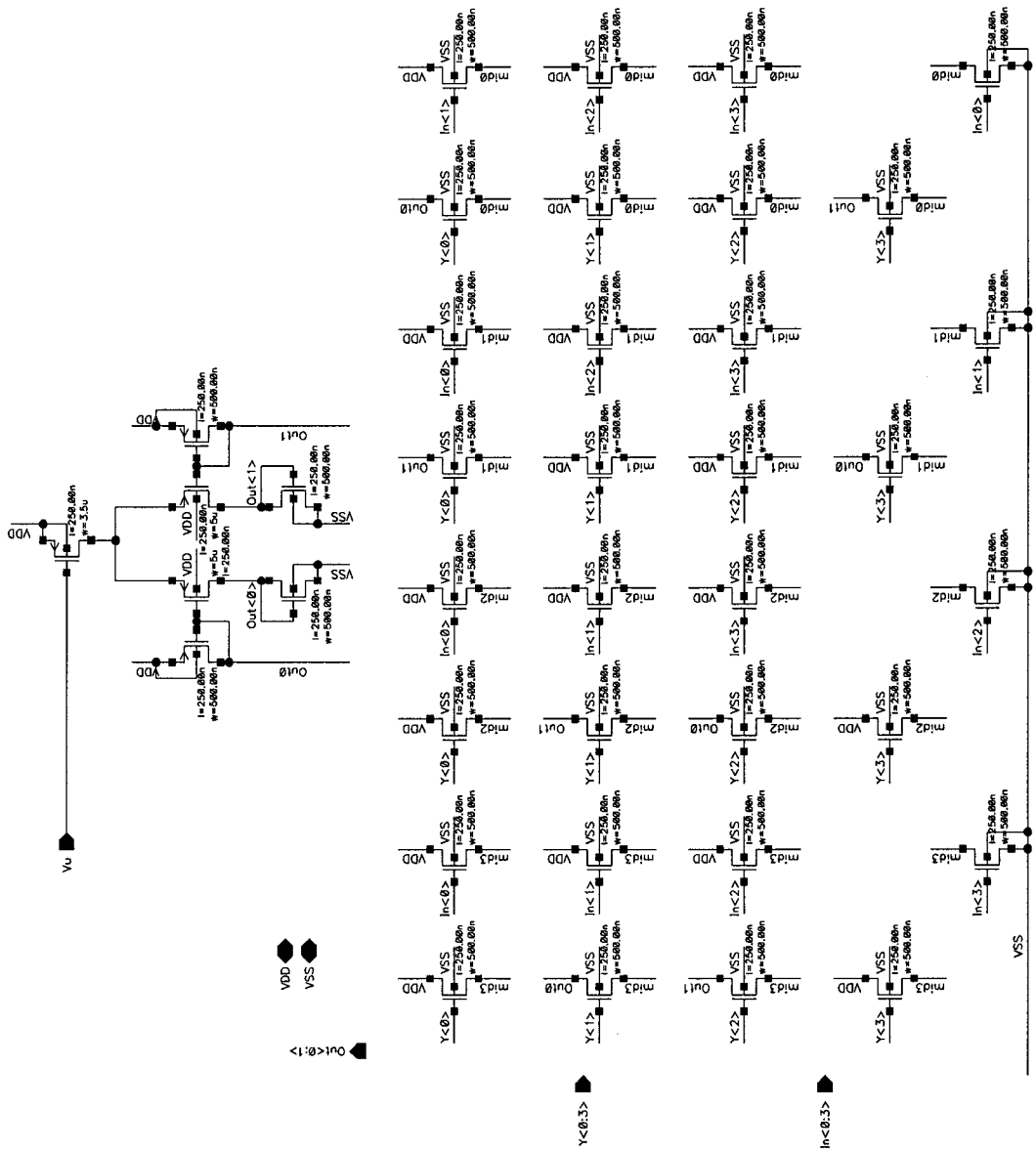


Figure 4.15: $\alpha 2$ : trellis section 2 forward metric

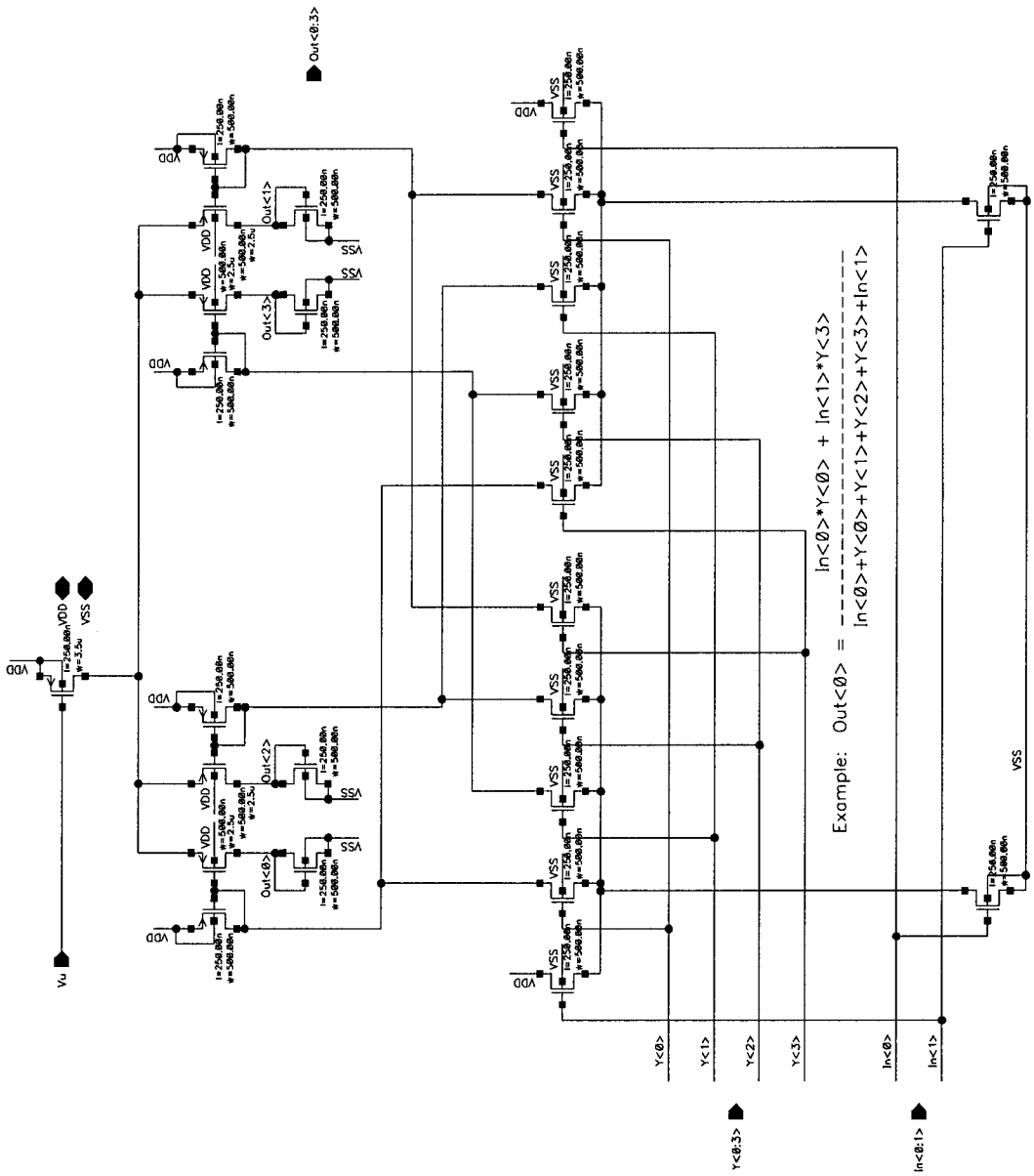



Figure 4.16: $\beta 2$ : trellis section 2 backward metric

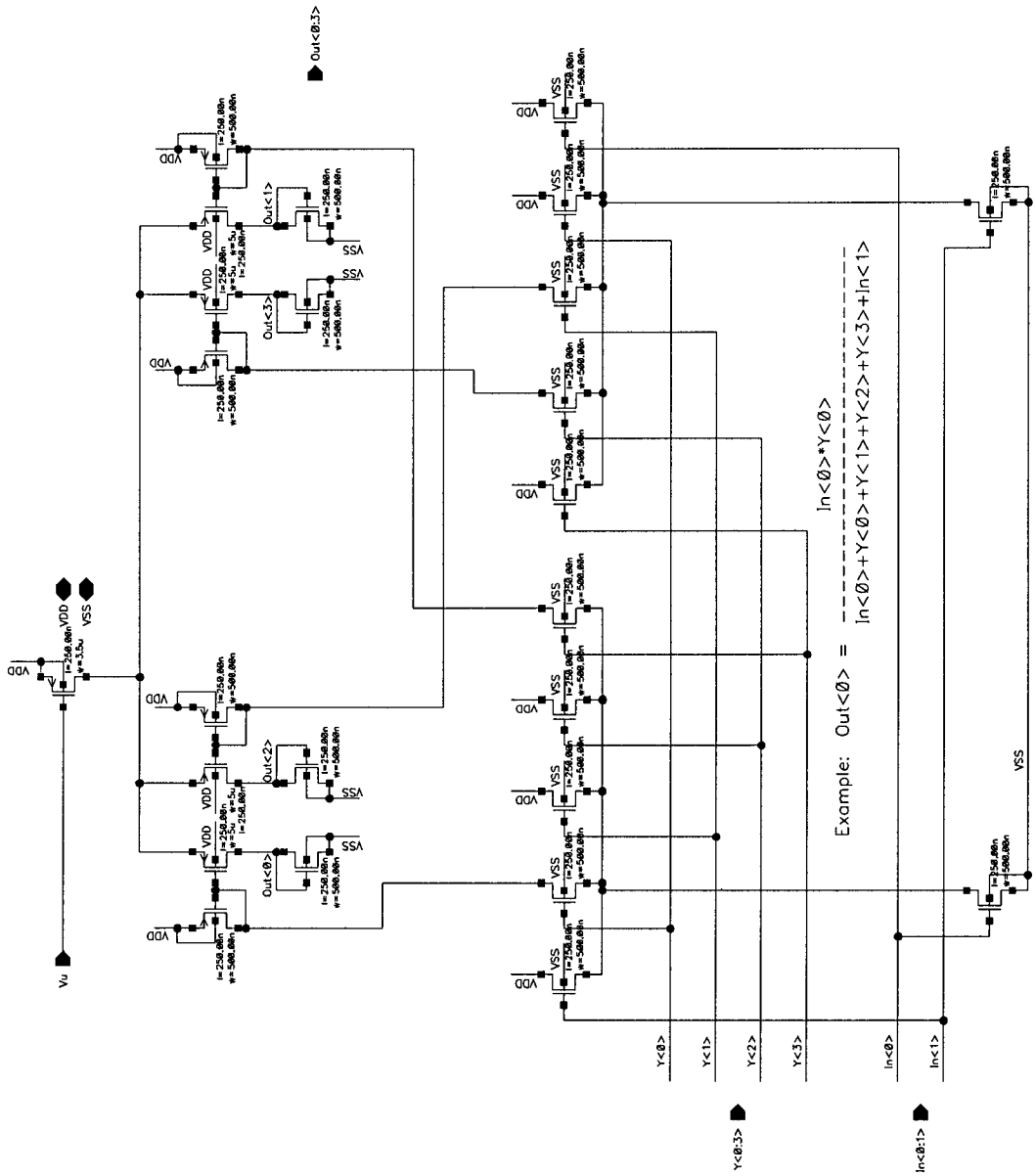



Figure 4.17: $\alpha 1$ : trellis section 1 forward metric

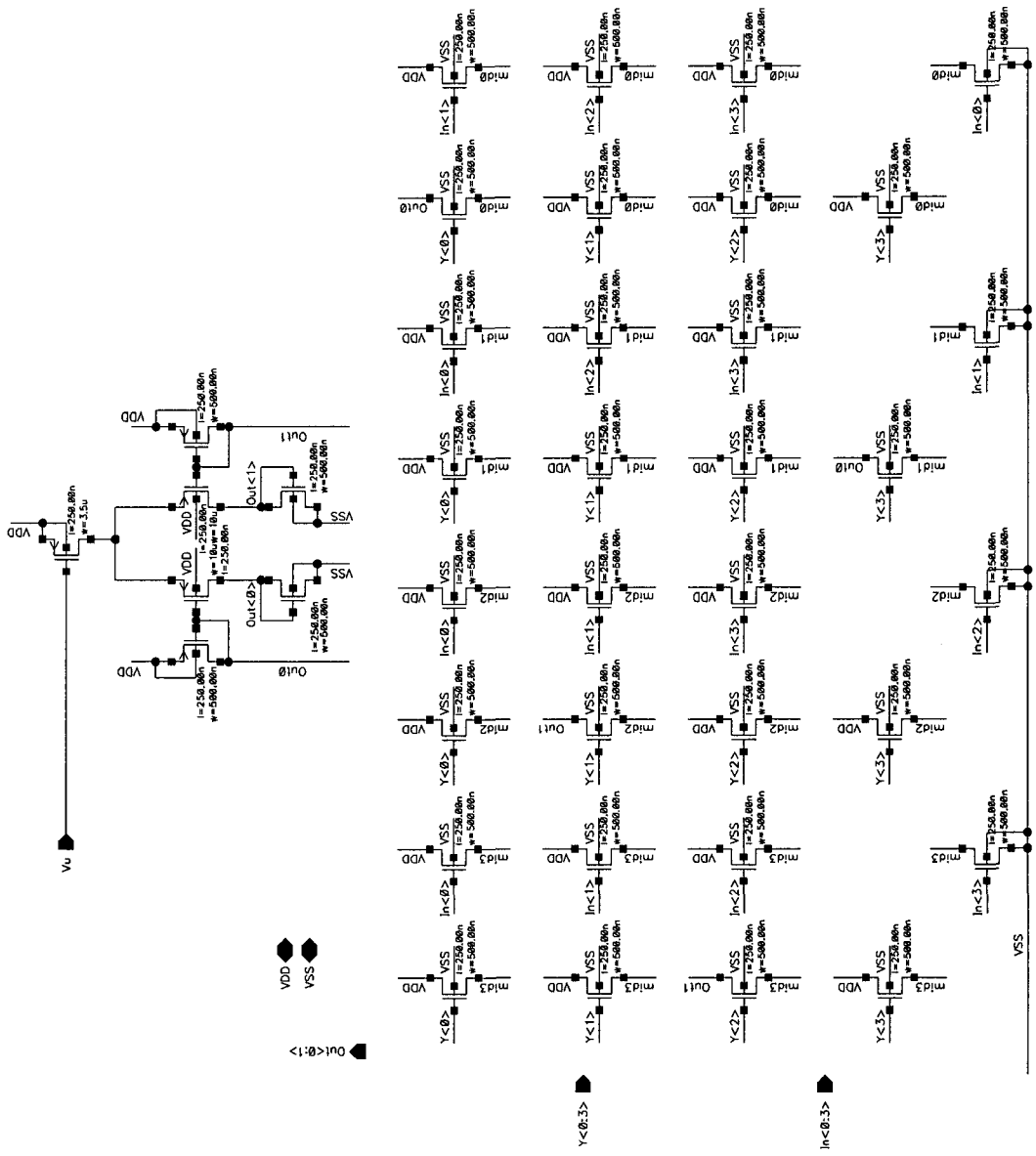



Figure 4.18: $\beta 1$ : trellis section 1 backward metric

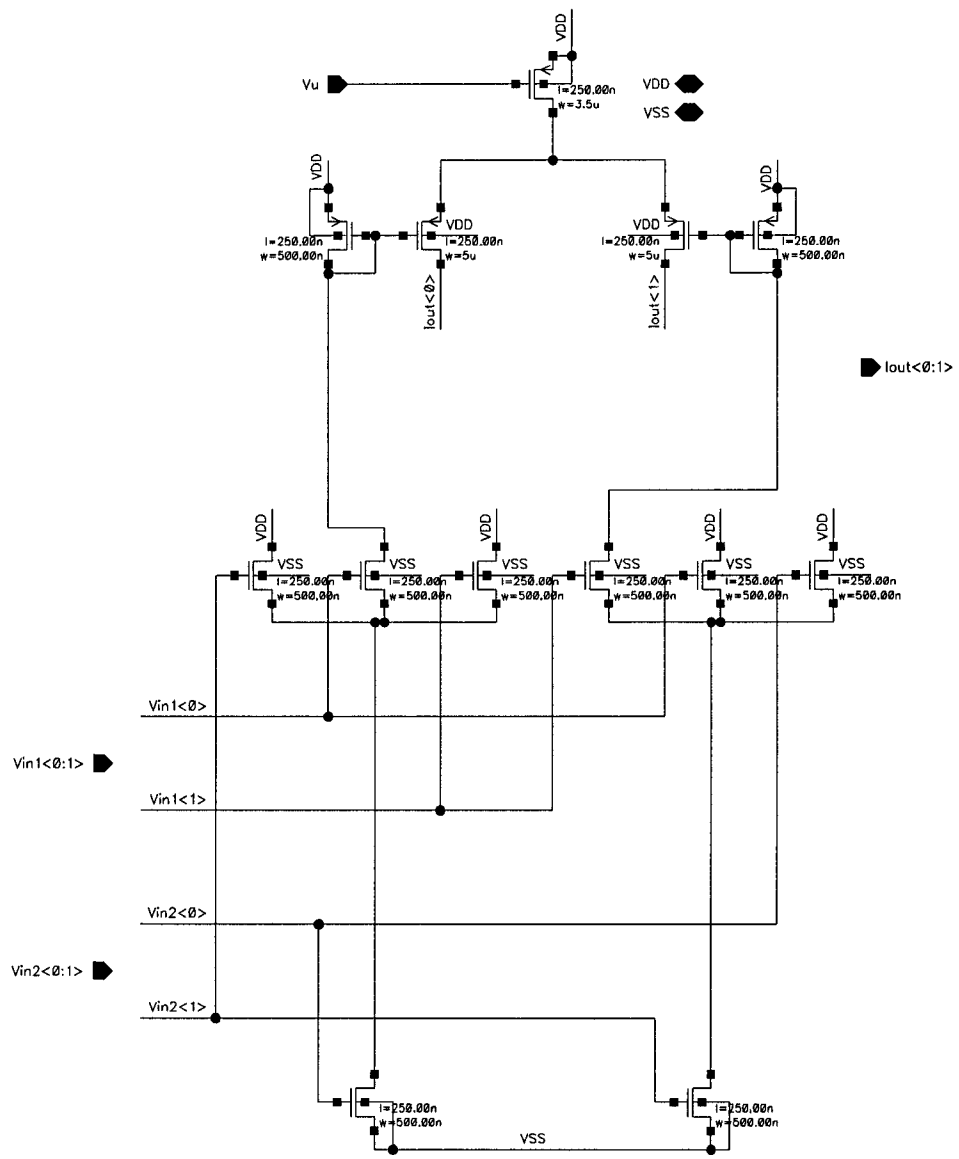



Figure 4.19: $\lambda 2$: APP calculation over 2 states

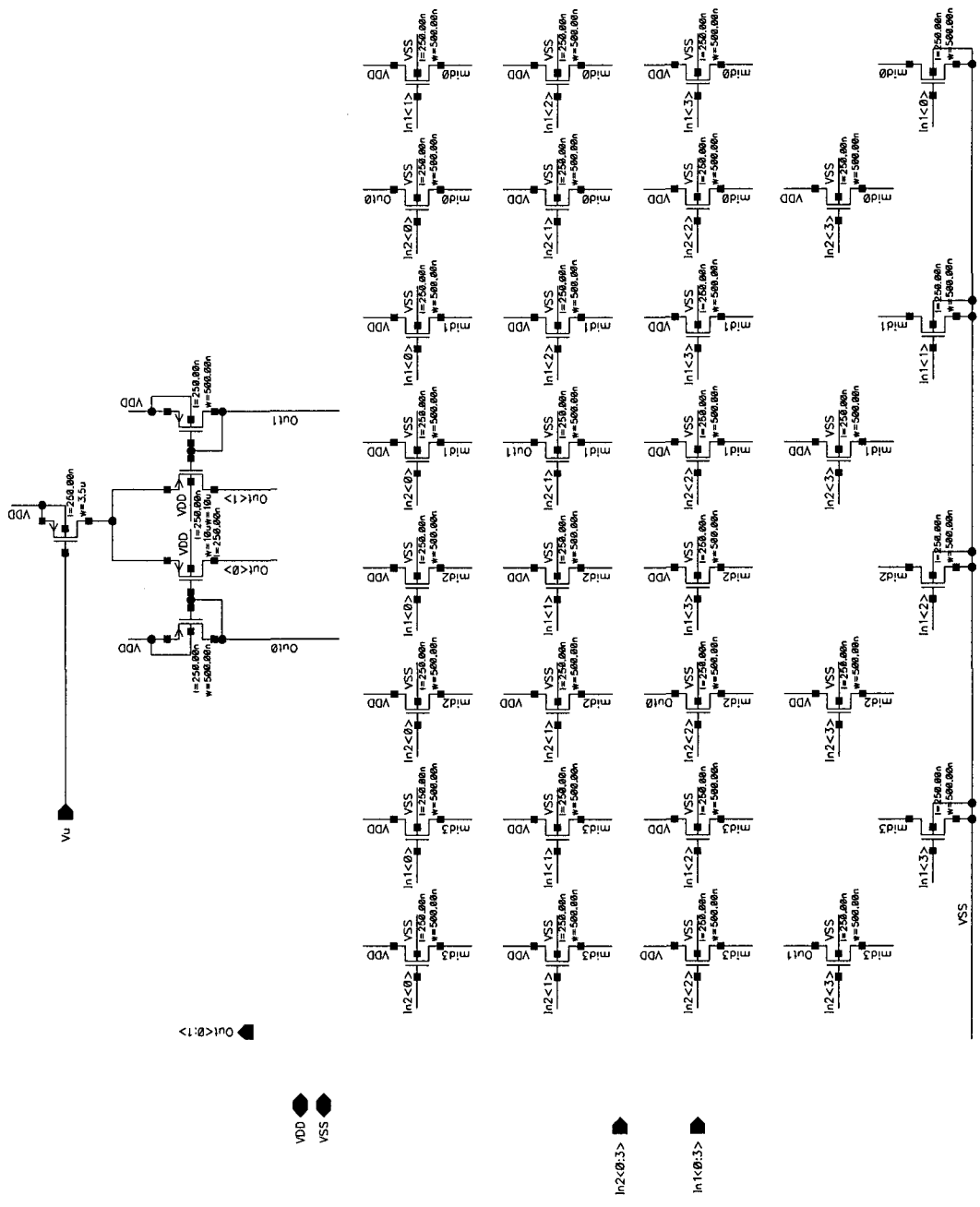



Figure 4.20: $\lambda 4$ : APP calculation over 4 states

## 4.3 Universal I/O interface

Both analog decoders described in the previous sections used I/O interfaces modified from [76, 75]. Shown in Fig. 4.21, the input is essentially an analog serial-to-parallel converter. LLR voltages are sampled one-by-one until the whole codeword is received. Then, these samples are transfered in parallel into a second capacitor for holding. The held LLR voltages are converted to probability currents and voltages for the decoding network to process. Outputs are compared, latched, and shifted serially. These interfaces were designed to be stackable and are used in other decoders designed by our group. To meet the requirements an (8,4) Hamming decoder, two input S/H chains of length 8, 4 output comparators, along with a length 4 shift register chain were needed.

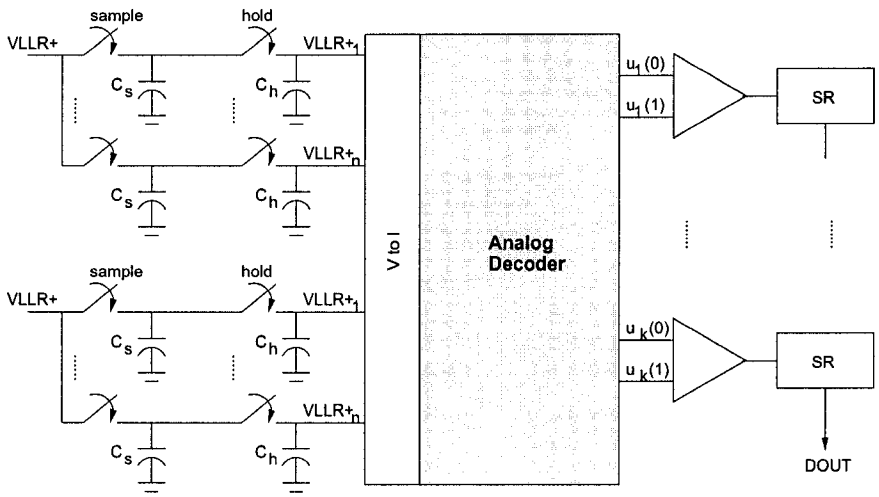



Figure 4.21: Universal I/O interface

The input S/H chain is shown in Fig. 4.22. Its timing is shown in Fig. 4.23. A high FRAME signal is needed only once at power up to reset the input interface. This signal has to overlap the high clock pulse completely. Once that is done, the input interface will enable SELi on every high clock pulse in order to sample $VLLRi+$ and $VLLRi-$, i = 1, .., 8. On SEL8, the holding capacitor is discharged to clear out old samples. Then PIPE is used to transfer in new samples. PIPE is also used as a decoder initialization/ reset signal if reset is required. This process then repeats for the next codeword.

The output serial interface consists of a comparator and a parallel load

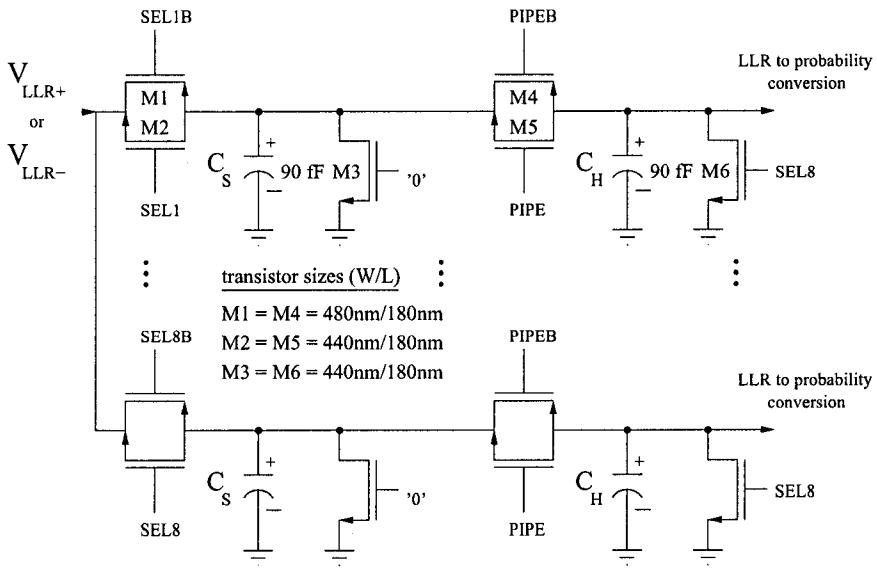



Figure 4.22: The input sample and hold (S/H) interface

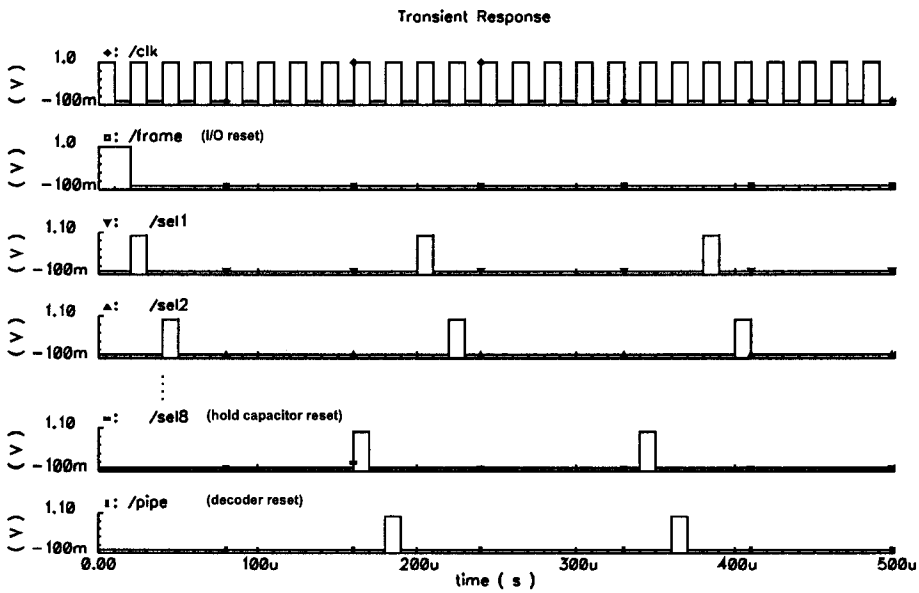



Figure 4.23: The input S/H interface timing showing sampling signals SELi and hold signal PIPE

shift register chain, as shown in Fig. 4.24. The internals of the comparator are shown in Fig. 4.25. The comparator latches two current inputs representing probability 1 and 0 into a bistable circuit. A decision in made and stored onto a set-reset (SR) latch. This decision is then transfered into a shift register when SEL7 is high. These decoded bits are then shifted out serially on the falling edges of CLK after SEL7. The timing of the output interface is shown in Fig. 4.26. This figure works in parallel with the input timing figure. The first outputs arrive $2n$ cycles after the initial FRAME signal. After that, a new set of decoded bits appear every $n+1$ bits due to pipelining and decoder reset.

Detailed schematics of the I/O interface are further shown in Appendix A.

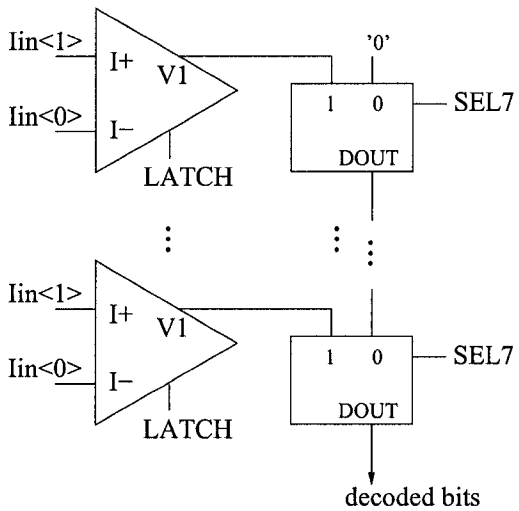



Figure 4.24: The output serial interface showing comparators and parallel load shift registers

## 4.4 Design Methodology

In the beginning, belief propagation simulations were run using the parity check matrix of the factor graph decoder to determine the approximate BER for comparison purposes. Row operations were done on the parity check matrix to produce an equal number of 1s in the rows and columns. This introduces regularity to make the design and layout process easier. Due to the time constraints of the project, no analog decoder modeling in C or MATLAB

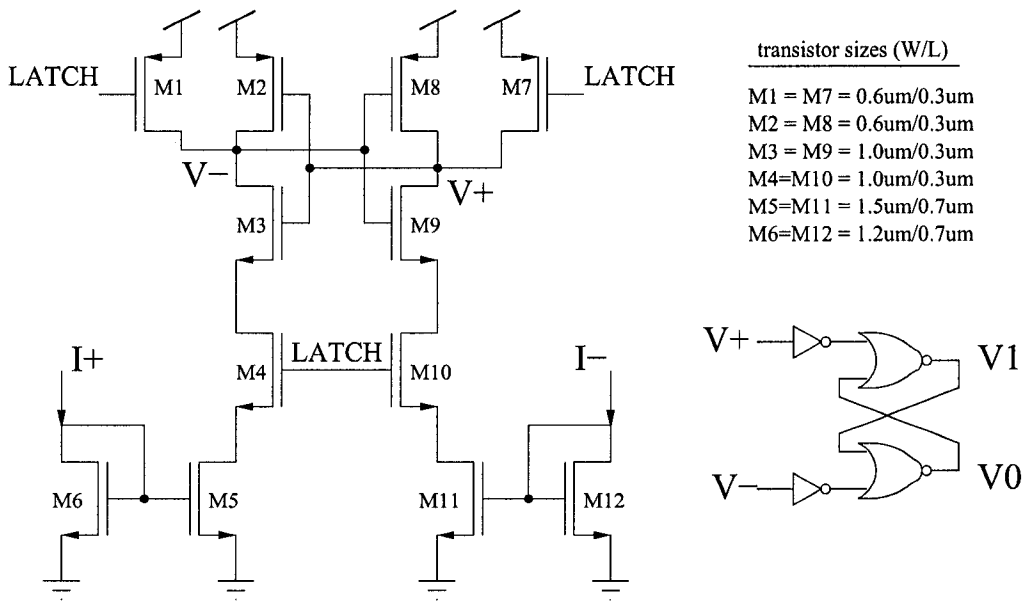



Figure 4.25: The internals of the output comparator with built in SR latch

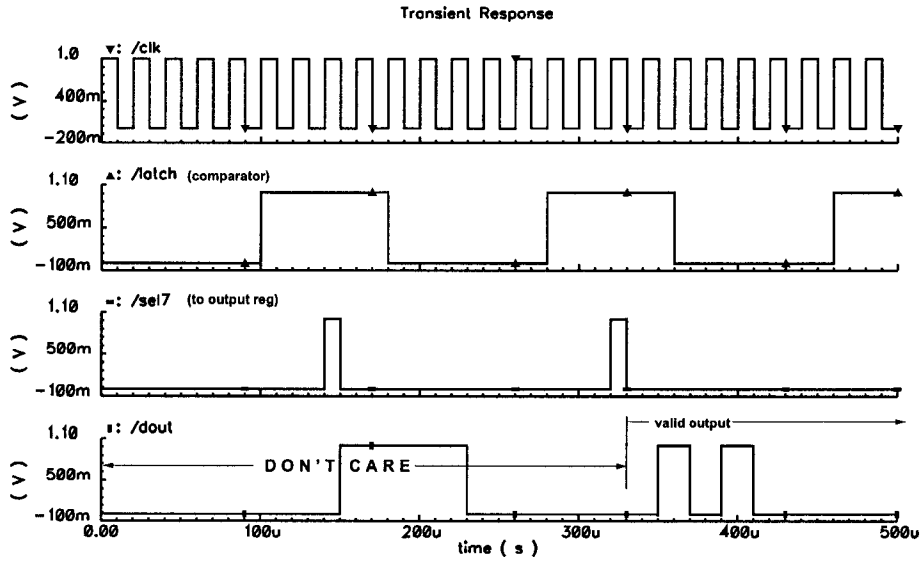



Figure 4.26: The output serial interface timing showing where comparator latching LATCH and output shift register loading SEL7 occur

was used. Similarly, an equivalent digital decoder was not designed. Instead, each block starting from a basic unidirectional node was drawn in Virtuoso and simulated mainly using Cadence Spectre. HSPICE was used to simulate factor graph nodes for $\Delta LLR$ plots because it facilitated easier data collection when used with Perl. These circuit simulators used the BSIM v3.1 model.

Once we were satisfied with simulation results, layout proceeded. There was a direct mapping of transistor location into layout location to help us identify transistors. The layout of each block was kept symmetric as possible and differential signals were kept close to each other. Substrate contacts were added wherever there was space. For example the layout of EQUALITY5 block in the factor graph decoder is shown in Fig. 4.27. The main goal at this design stage was to minimize area. The layout was then extracted with parasitic capacitance and were simulated and compared with schematic results. If the results agree, then we proceed with integrating the smaller block into a larger block. This process is repeated leading to the top-level decoder. Along every step of the way, LVS and DRC was performed to ensure that the final decoder matched the schematics. This methodology is suited to small designs such as the two decoders built in this thesis, but might be inefficient for larger length decoders because of the amount of manual routing required.

The simulation of individual blocks was done by looking at the impedance of the next stage. Transistor sizes in the product circuit were made similar for the ease of simulation. Normalizer widths were chosen to be multiples of product transistor widths to aid in layout. Parametric analysis was used in the beginning of the design cycle to sweep design variables. In theory, any allowable size ratio used in the product circuit will create a working multiplier and it is the sizing of the normalizer that matters most in this low voltage topology. However, in simulation certain transistor ratios give better current outputs and hence potentially faster speed. Once transistor sizes are chosen and the overall decoder is put together, *extracted* simulations for functionality, speed, and power can be done.

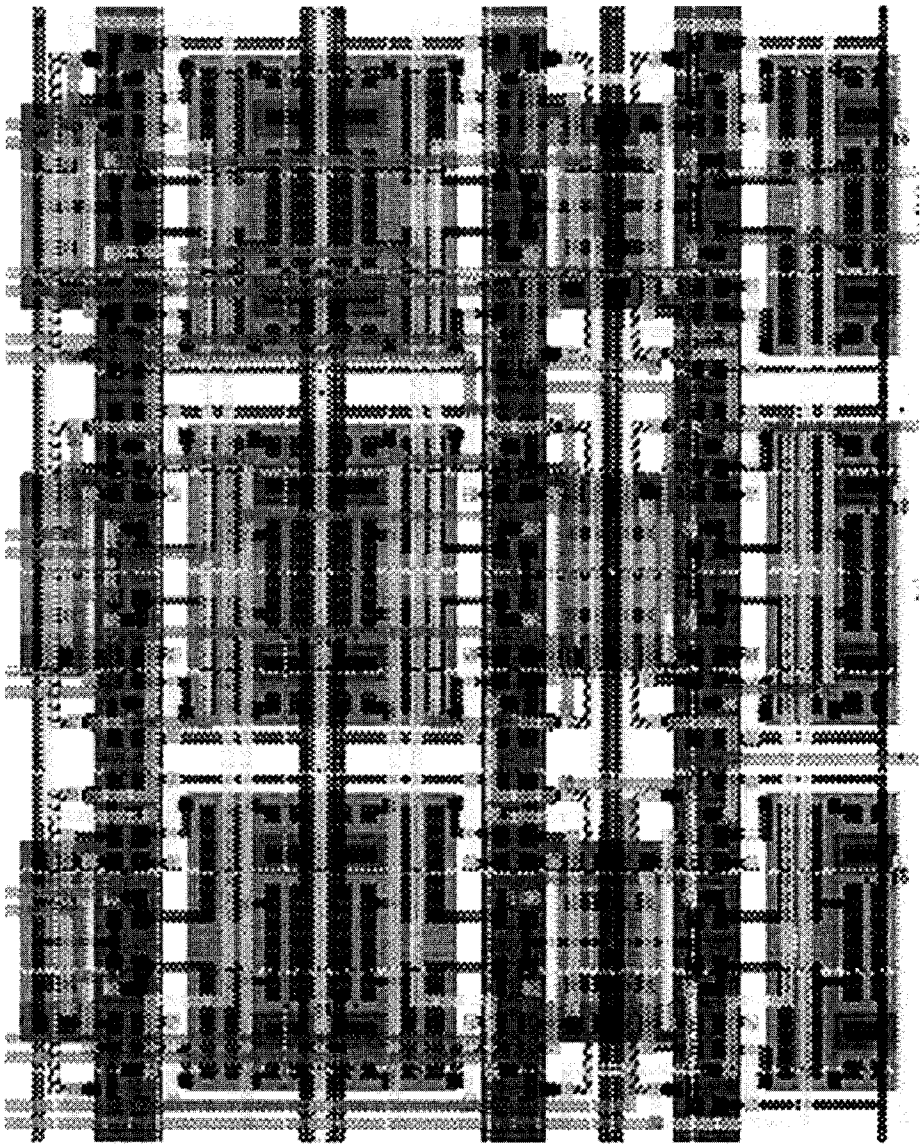


Figure 4.27: Layout of the EQUALITY5 block

## 4.5 SPICE Simulation

The analog portion of the factor graph decoder was simulated by injecting probability currents and monitoring its output. To test the robustness of the architecture, a supply of $V_{DD} = 0.5V$ and unit current bias of $I_u = 10nA$ were used. The probability values in Table 4.5 were injected into the input. These probabilities represent a codeword of 00110111 where the error bit is in **bold**. The output currents are displayed in Fig. 4.28. Even at such a low supply voltage, the moderate error on bit 3 is corrected from '1' to '0' within 8 $\mu$s. Other current values are also shown. The legend is indicated on the top portion of the graph. For example, O1<1> indicates output probability current of bit 1 being '1'. We should note that the settling time of the decoder will depend on many factors including the noise dependent input probabilities, error pattern, supply voltage, and unit current bias. Fig. 4.28 only gives us an idea of what an error correction will look like; it is not accurate enough to predict decoder operating speed.

Table 4.3: Probabilities used for the simulation of the analog portion of the factor graph decoder

| Bit # | p(0) | $I_0$ (nA) | $I_1$ (nA) |
|---|---|---|---|
| 1 | 0.7 | 7 | 3 |
| 2 | 0.6 | 6 | 4 |
| 3 | 0.3 | 3 | 7 |
| 4 | 0.2 | 2 | 8 |
| 5 | 0.8 | 8 | 2 |
| 6 | 0.3 | 3 | 7 |
| 7 | 0.2 | 2 | 8 |
| 8 | 0.1 | 1 | 9 |

The next step is to integrate the I/Os and simulate the overall factor graph decoder, as shown in Fig. 4.29. In this simulation, both I/Os and decoder are operating at a supply VDD = 0.8V. The input codewords are serially shifted in as $\Delta VLLR = V_{LLR+} - V_{LLR-}$ and this difference is represented through a differential pair ($V_{LLR+}$, $V_{LLR-}$). For convenience, we used a constant $V_{LLR-} =$

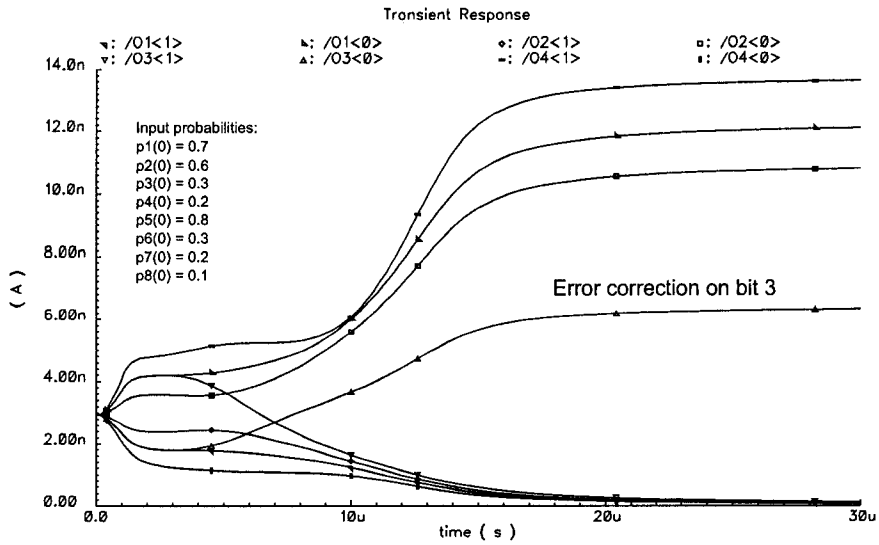



Figure 4.28: The current outputs of the analog portion of the factor graph decoder ($V_{DD} = 0.5V$, $I_u = 10nA$) showing an error correction on bit 3

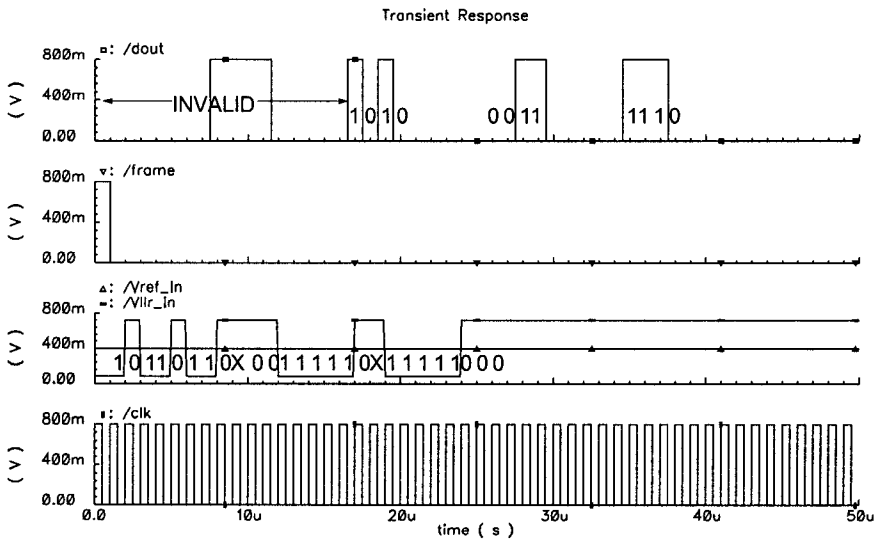



Figure 4.29: The simulated outputs of the full factor graph decoder ($V_{DD} = 0.8V$, $I_u = 1\mu A$) with 3 input codewords and their corresponding decoded information bits

Table 4.4: Factor graph decoder simulation summary

| | |
|---|---|
| Supply | VDD=0.8 V |
| Unit current | $I_u$=1$\mu$A |
| Power | 283$\mu$W (simulated) |
| I/O clk | 1 MHz (simulated) |
| Bit rate | 444 kbps (simulated) |
| Energy per decoded bit | 0.64 nJ/b (simulated) |

$0.4V$ while varying $V_{LLR-}$ between 0.08V and 0.72V giving $\Delta VLLR$ values of -0.32V and 0.32V, respectively. This difference translates into probabilities very close to 1 and 0 making input bits look like hard bits. When $V_{LLR-} > V_{LLR+}$, the input bit is '1' and '0' vice versa.

It takes 8 clock cycles to shift in input values and another 8 cycles for decoding. Outputs appear on the falling edge of the 16th cycle and are shifted out on the falling edges of CLK. Only 4 information bits are shifted out and the rest are set to '0'. Pipelining is achieved by shifting in a new codeword while the present codeword is decoding. After the first 16 clock cycles, information bits will appear every 9 cycles because 1 cycle is used for decoder reset. With a bias current of 1$\mu$A, a maximum clocking rate of 1 MHz was achieved giving a decoding rate of 444 kbps since only 4 information output bits are recorded every 9 clock cycles. Codewords of 10110110, 00111110, 11111000 were used. Error bits are shown in **bold**. The corrected output information bits were 1011, 0011, and 1110, as shown in Fig. 4.29. The simulated power consumption for this simulation setup was 275.2$\mu$W (analog) and 8.04$\mu$W (digital I/O) giving a total of 283.2$\mu$W. These results are summarized in Table 4.4.

Similar figures were created for the trellis graph decoder. Simulation was done on the analog portion of the trellis graph decoder to get Fig. 4.30. The same supply of $V_{DD} = 0.5V$ and unit bias current of $I_u = 10nA$ were used. The smooth curving current outputs resemble previous decoders using regular Gilbert multipliers at higher supply voltages.

Fig. 4.31 shows the simulation for an overall trellis graph decoder. This

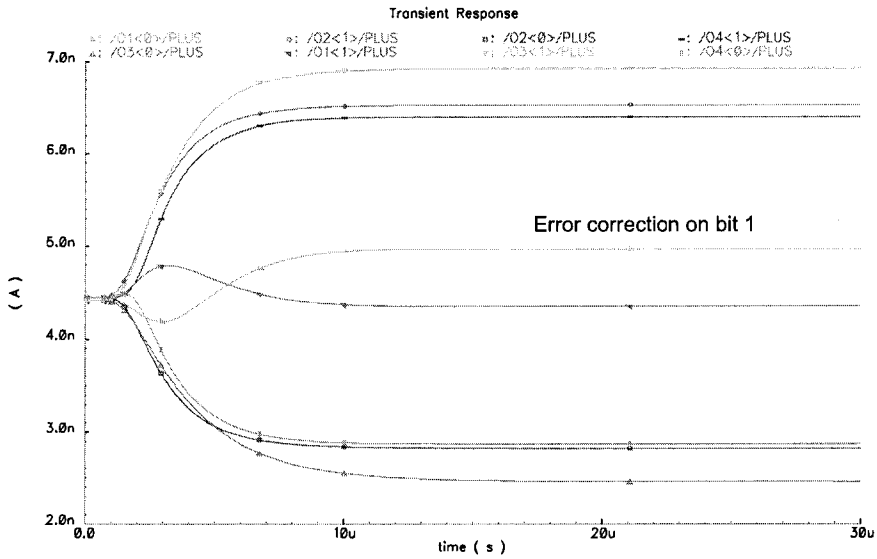



Figure 4.30: The current outputs of the analog portion of the trellis graph decoder ($V_{DD} = 0.5V$, $I_u = 10nA$) showing an error correction on bit 1

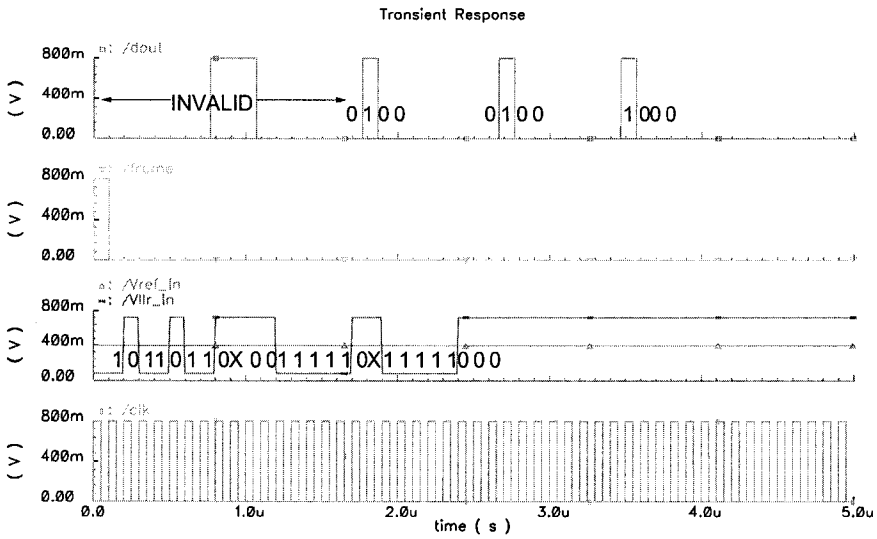



Figure 4.31: The simulated outputs of the full trellis graph decoder ($V_{DD} = 0.8V$, $I_u = 1\mu A$) with 3 input codewords and their corresponding decoded information bits

Table 4.5: Trellis Decoder Summary

| | |
|---|---|
| Supply | $V_{DD}$=0.8V |
| Unit current | $I_u$=1 $\mu$A |
| Power | 36$\mu$W |
| I/O clk | 10 MHz (simulated) |
| Bit rate | 4.444 Mbps (simulated) |
| Energy per decoded bit | 0.0082 nJ/b (simulated) |

decoder was biased at a supply of $V_{DD} = 0.8V$ and a unit current $I_u = 1\mu A$ using the exact same simulation setup and I/Os as the factor graph decoder, yet it is capable of faster clocking. The clock shown in the figure is running at 10 MHz giving a decoding rate of 4.44 Mbps. Codewords of 10110110, 00111110, and 11111000 were decoded to 0100, 0100, and 1000, respectively. The simulated power consumption for this setup was 24.14$\mu$W (analog) plus 12.26$\mu$W (digital I/O) giving a total of 36.4$\mu$W. These results are summarized in Table 4.5.

Simulations show that reset (or initialization) of the probability values is critical in making the factor graph decoder work but not so for the trellis decoder. A reason could be that the factor graph contains more processing circuits which adds to its inertia while the trellis decoder has a simpler structure. In the trellis decoder, there are short feedback $\alpha$ and $\beta$ loops which allow it to grasp onto new values easily.

Minimum-sized pass transistors were used to equalize probabilities between the connections of equality to parity check nodes, as discussed in Sec. 4.1. The factor graph decoder simulations with and without reset are shown in Figs. 4.32 and 4.33. Output probability currents with reset have full swing for both high and low currents whereas without reset, there is only good swing for the higher of the two currents.

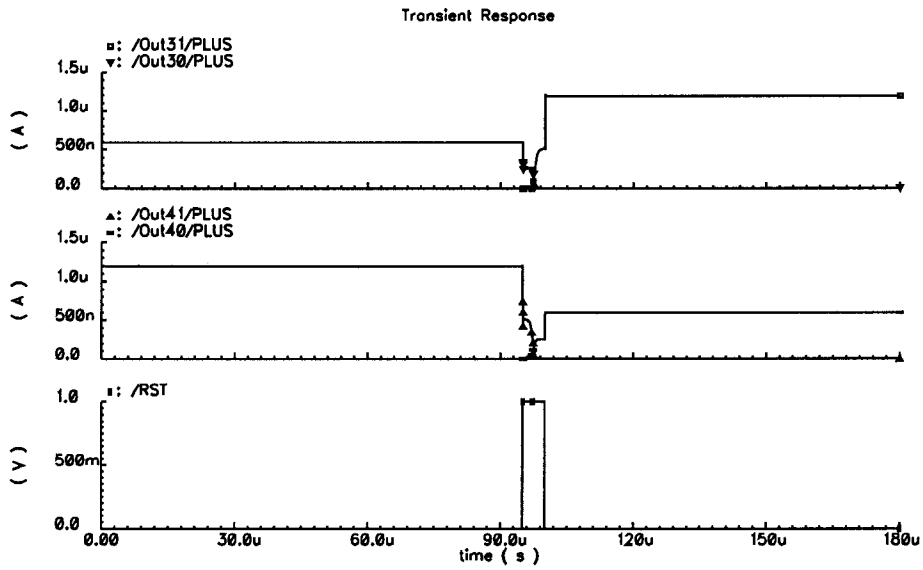



Figure 4.32: With reset in the factor graph decoder differential output currents have full swing

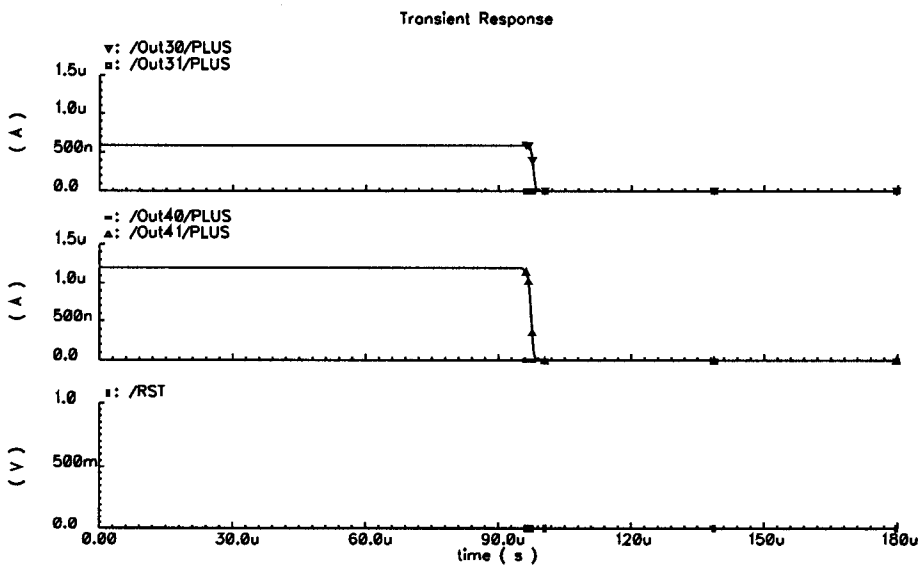



Figure 4.33: *Without* reset in factor graph decoder differential output currents have only good swing for the higher of the two currents

## 4.6 Chapter Summary

In this chapter, the low voltage multiplying architecture was applied to the construction of two analog decoders operating at sub-1V supplies. The design of an (8, 4) Hamming analog decoder based on the code's factor graph and the design of an (8, 4) Hamming analog decoder based on its tail biting trellis graph were described. The I/Os used for both decoders were also included. The design methodology and extracted SPICE simulation results were shown. We now proceed to describing the BER test setup and BER measurement results.

# Chapter 5

# Testing

This chapter describes the testing of low voltage analog decoders. The first section describes the test setup and equipment used in measuring BER. The second section focuses on testing methodology. This is followed by measurement results.

## 5.1 Test Setup

An analog decoder test setup is shown in Fig. 5.1. Describing clockwise from the left side, a PC (silver box on top rack) is used to send test samples and to record test results. A Keithley source/measure unit is used to generate a bias current on the order of nA to $\mu$A. A mixed-signal oscilloscope and multimeter are used for probing signals. The decoder is situated on a printed circuit board (PCB) and is connected to two others, as shown in Fig. 5.3. We can redraw the test setup as a block diagram (Fig. 5.2) to show the interaction between main blocks: test program, FPGA board, test support board, and device under test (DUT) board.

Channel observations, or samples, are sent from the PC via a USB interface to the FPGA controller. The FPGA collects 16 bits to represent each sample. This sample is applied to a DAC to form an analog channel observation. A block length (BL) of 8 samples is needed for the (8, 4) decoder. Once 8 samples have been received by the FPGA, they are clocked into the decoder chip situated on the DUT board. Time is given for decoding to take place. Once decoding is finished, the results are captured by the FPGA board and

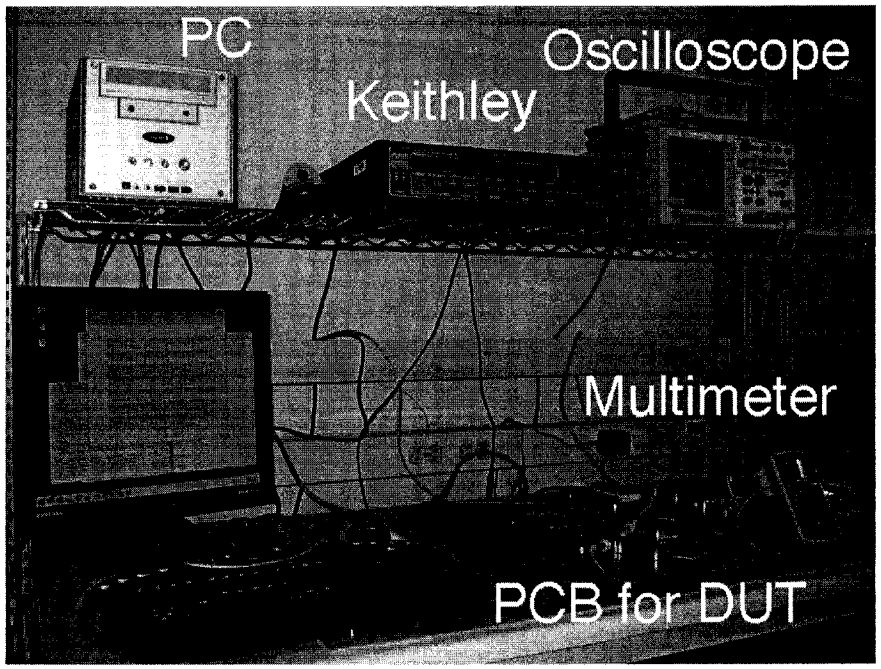



Figure 5.1: Picture of analog decoding test setup, clockwise from left side, PC, Keithley unit, oscilloscope, multimeter, and three PCBs (middle bottom)

sent back to the PC. Subsequently, new samples are applied. Test results collected by the FPGA board do not have to be sent back to the PC in real time. Instead, they can be stored until all information bits for one decoded word have been received. The number of bits sent from the PC to the FPGA will not be equal to the amount it receives back. In testing the (8, 4) decoder, a total of 8 samples / BL x 16 bits / sample = 128 bits will be sent from the PC to the FPGA. A total of 4 information bits will be sent from the FPGA back to the PC. It is up to the test program on the PC to calculate the final BER.

### 5.1.1 Test program

The test program is a modified BER simulation program. It utilizes similar libraries but it also has communication routines to and from the FPGA through the USB device. The program sends codewords corrupted by noise and receives decoded bits. BER results are updated on a waterfall curve. The

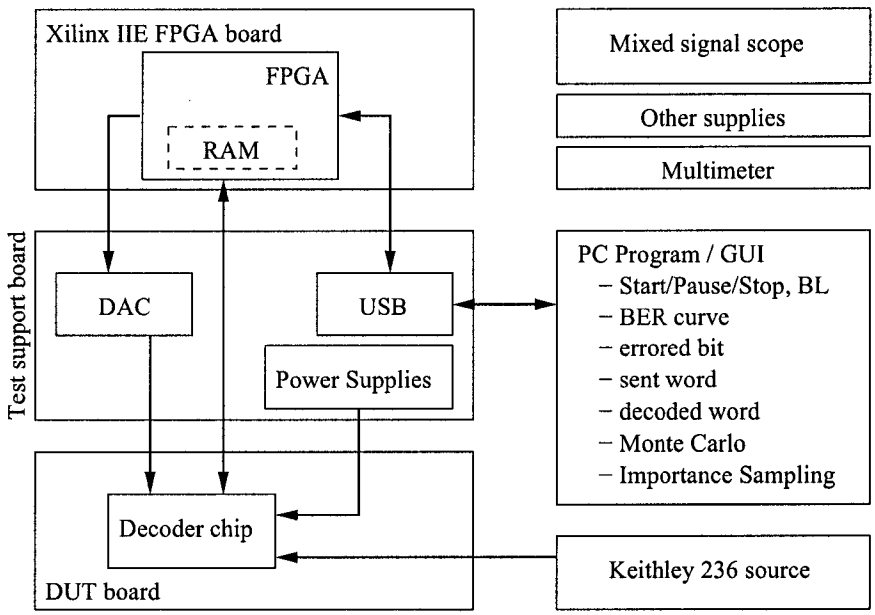



Figure 5.2: An analog decoding test setup block diagram showing interaction between blocks

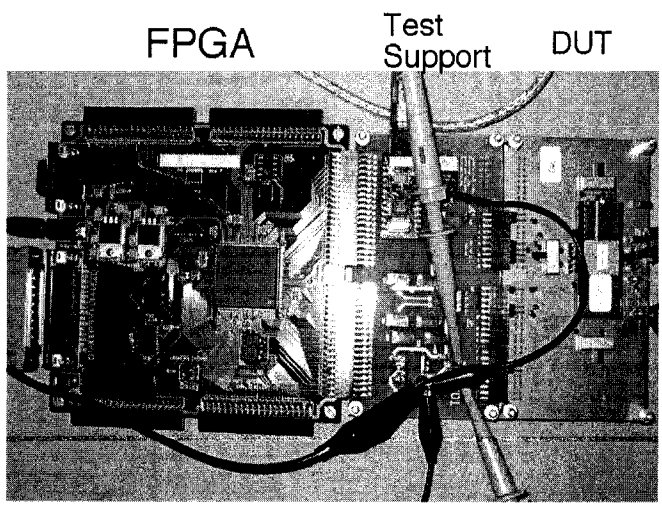



Figure 5.3: Picture of FPGA, Test Support, and DUT boards

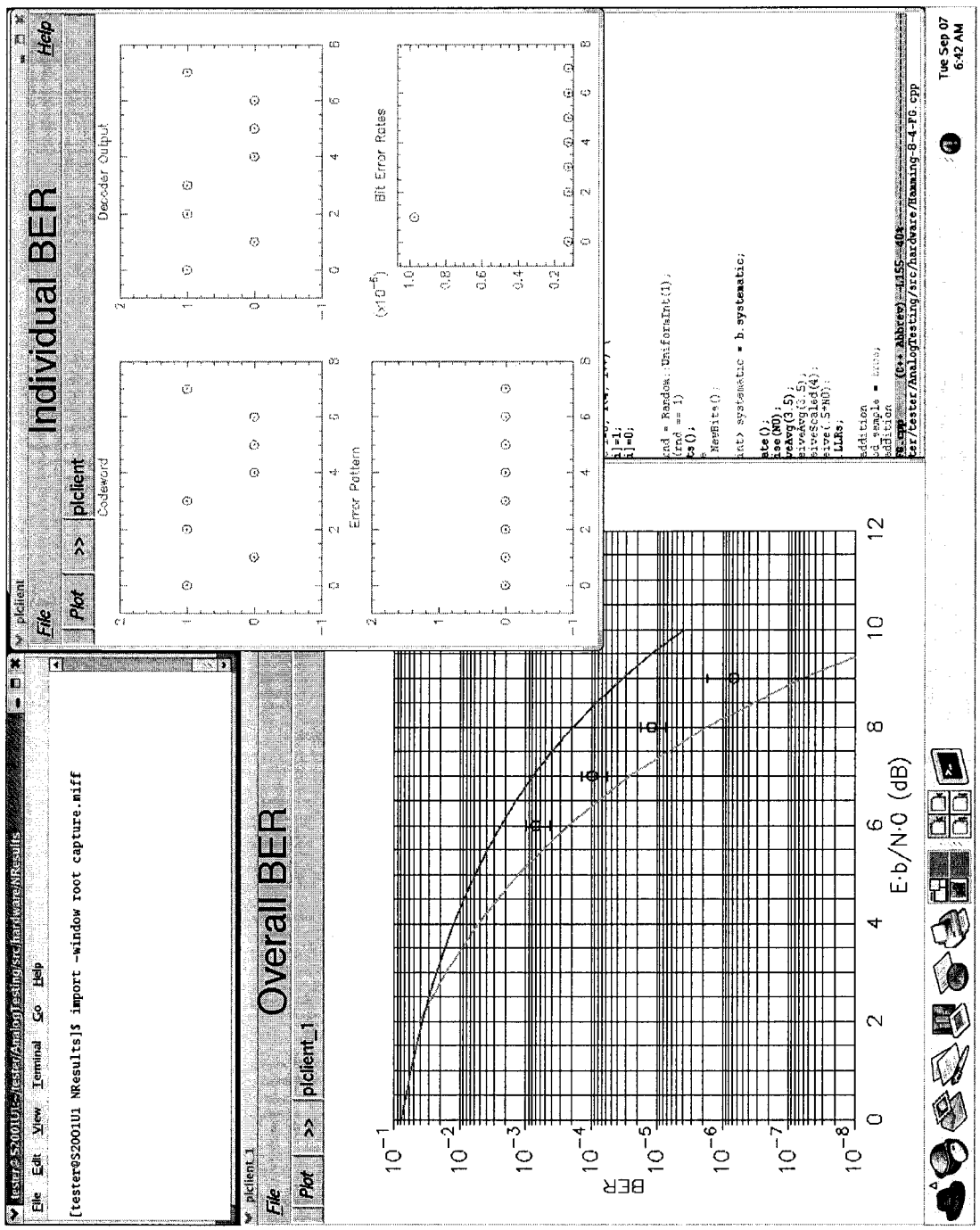



Figure 5.4: Screen capture of test program

error probability of each bit is displayed on a graph. Just like in a regular BER simulation, normal Monte Carlo analysis and importance sampling can be used. A screen capture of the test program is shown in Fig. 5.4.

The test program goes through these steps:

1. Display ideal uncoded BPSK and Dmin asymptote bits
2. Send control data indicating start of test, testing speed, and block length
3. Randomly or deterministically generate a source word
4. Encode the source word to a code word using a generator matrix
5. Modulate code word to BPSK symbols {'0'→+1, '1'→-1}
6. Add AWGN to modulated symbols according to SNR
7. Convert AWGN modulated bits to LLR values (3 scaleable options)
   - Avg($x$) : scale LLRs so they average out to be $x$
   - Receive($x$) : scales LLRs to $LLR = 4E_s/N_o$, $N_o = x$
   - ReceiveScaled($x$) : scales LLRs so that the maximum LLR is $x$
8. Convert LLR values to DACCODE values (scaleable)
   - Scale DACCODE using $x$ where $DACCODE = LLR \cdot x + 8192$
   - The range for DACCODE is from 0 to 16383
9. Send DACCODE values to FPGA test controller and wait for specified amount of time
10. Receive decoded bits from FPGA
11. Compare bit positions to determine bit errors
12. Update BER and individual bit error probability graphs
13. Repeat steps 3 to 12 until enough errors occur for each SNR

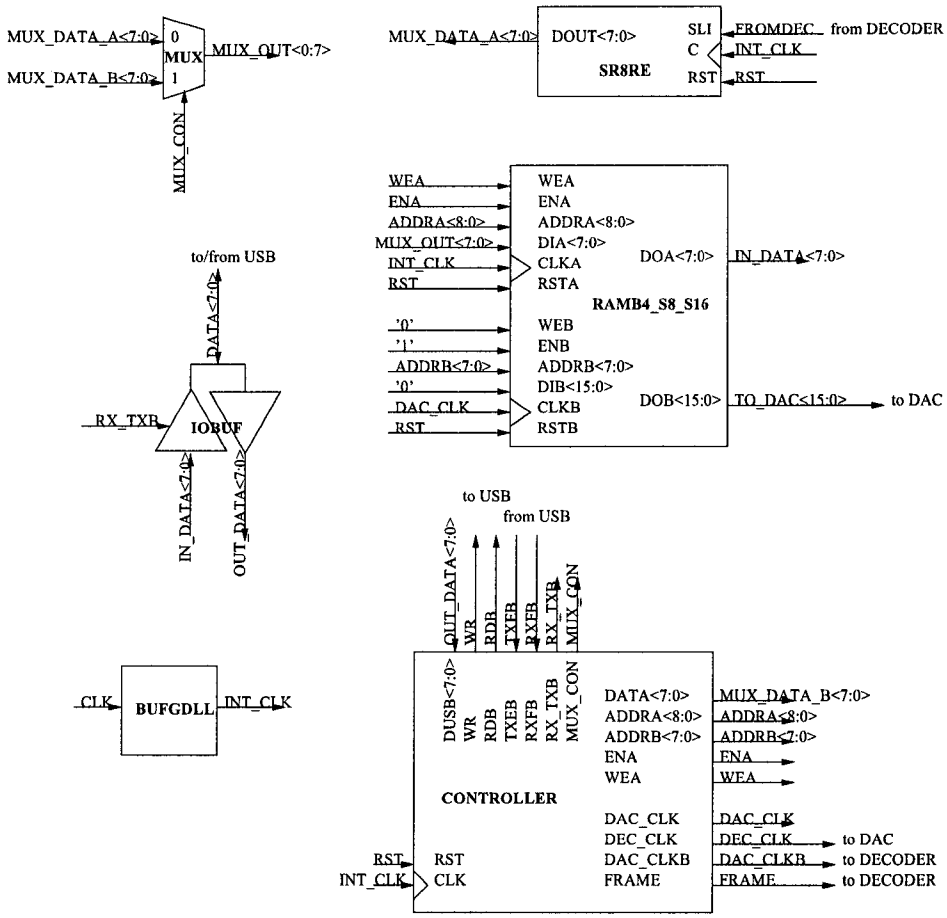



Figure 5.5: The FPGA controller consists of an FSM, RAM, and other support circuitry

### 5.1.2 FPGA controller board

The Digilent D2E board contains a Xilinx Spartan IIE FPGA chip with 200K gates. The test controller, which is specified in VHDL, is made up of an FSM and support blocks shown in Fig. 5.5.

We begin with a description of the support blocks. The SR8RE shift register loads decoded bits serially and transfers its contents to RAM in parallel. MUX is used to switch between samples applied to the DAC and decoded bits from the decoder. IOBUF provides bus interface to the USB device. BUFGDLL stabilizes the incoming clock for internal components. A dual port memory component RAMB4_S8_S16 initially stores samples and later stores decoded bits. The memory size of 4096 bits is adequate for storing up $256 \times 16$

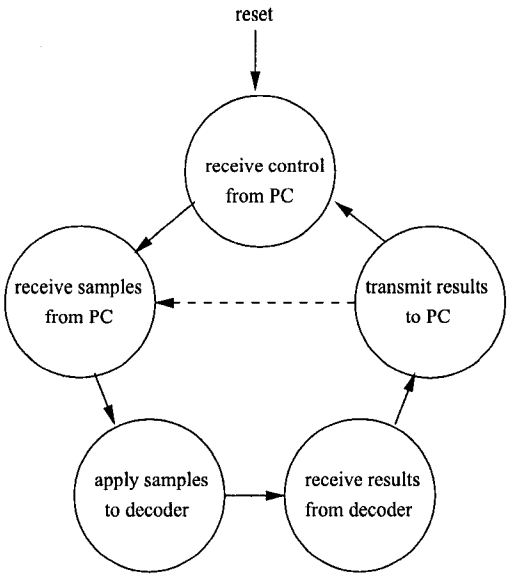



Figure 5.6: The FPGA controller FSM (simplified) showing 5 major states

bit DAC samples. We write to RAMB4_S8_S16 using port A only. Port B is used to read out test samples during transfers to DAC.

The FPGA controller is an FSM as shown in Fig. 5.6. Its operation can be grouped into five functions: receiving control values from PC, receiving test samples from PC, applying test samples to decoder, receive test results from decoder, and transmit test results back to PC. The operation of the FPGA board (with the controller) is shown in Fig. 5.7. Its VHDL code is shown in Appendix B.

The USB data path is limited to one byte which facilitates the need for describing communication using one byte packets. Initially, the controller looks for a start sequence that consists of 2 bytes as illustrated below,

| 11CC CCCC | BBBB BBBB |
|---|---|

where CC CCCC is a 6 bit number used to represent the clock divider (CLK_DIV), BBBB BBBB is an 8 bit number which represents the block length (BL) in *bytes*. CLK_DIV and BL are saved in hardware for later use. The above is followed by $2n + 1$ bytes of which $2n$ bytes are saved into RAM to represent $n$ code samples,

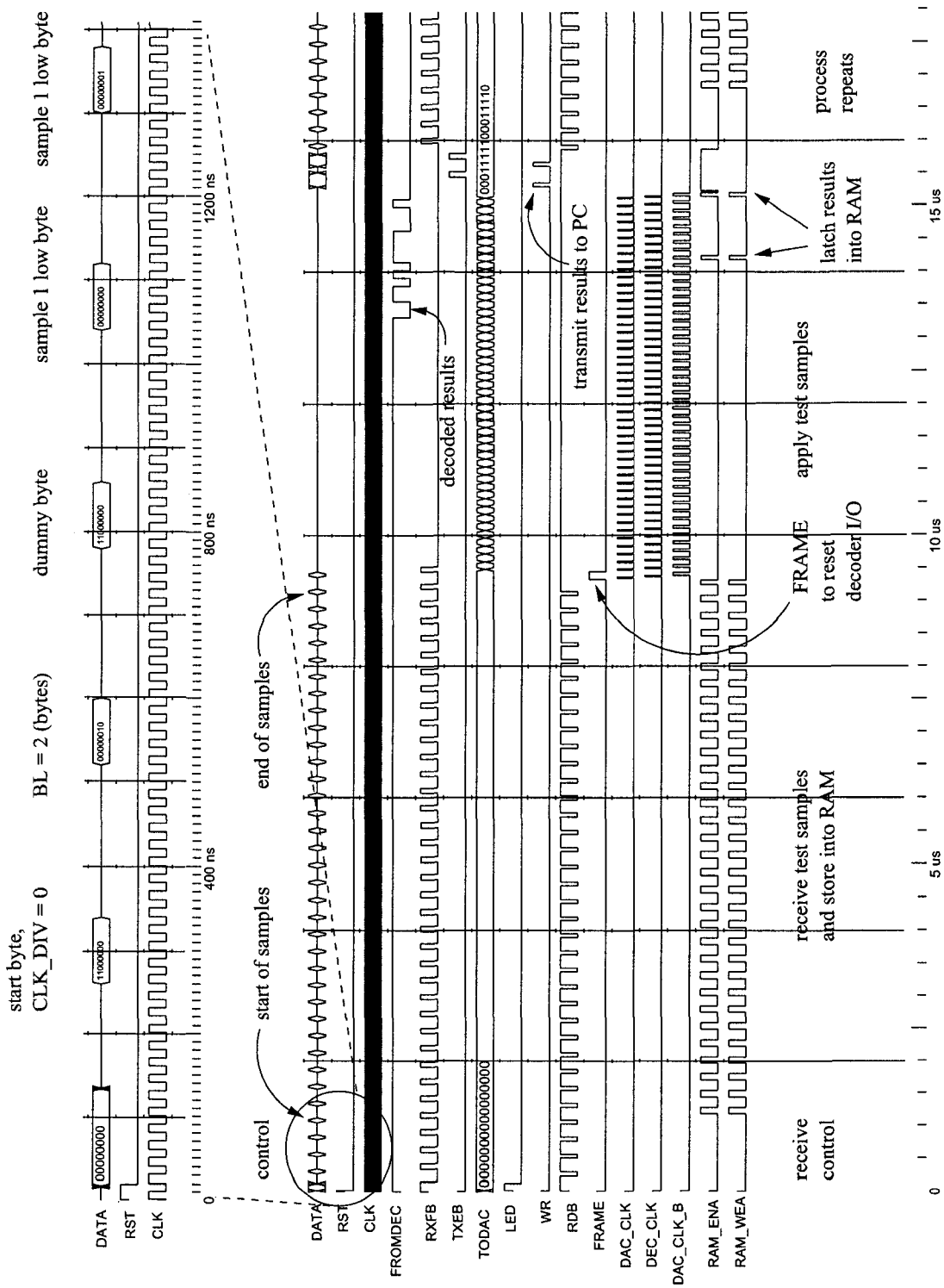



Figure 5.7: FPGA controller timing diagram showing the transition of 5 major states

| HHHH HHHH | SSSS $SSSS_1$ | XXSS $SSSS_2$ | ... | XXSS $SSSS_{2n}$ |
|---|---|---|---|---|

where HHHH HHHH is a dummy byte which can be used to stop the test when it is 0000 0000 otherwise testing proceeds normally. Two bytes are needed to represent a 14-bit DAC sample. For instance, XXSS $SSSS_2$ and SSSS $SSSS_1$ are used to represent the first analog sample. X's are not used.

In non-pipelined testing, these samples are applied twice to the decoder. The second application is only used for clocking purposes allowing the decoder enough time to settle. Once $2n$ cycles have been clocked, decoded values are shifted out and saved into RAM. These results are then sent back to the PC. The process then repeats with the start sequence. This is the simplest, least error prone approach and it is used in our test setup.

To speed up testing, a pipelining approach can be taken. In this case, the start sequence can be sent once and then only samples are sent. Each $n$ samples are still represented by $2n + 1$ bytes and HHHH HHHH is used as a stopping mechanism. When one sample is being decoded another is shifted in speeding up the testing process by almost a factor of 2. The test program on the PC has to be modified accordingly adding more complexity.

There are four clocking mechanisms at work: INT_CLK, DAC_CLK, DEC_CLK, and DEC_CLKB. The naming of these signals might be a source of confusion since their roles have been changed in the course of debugging. An on board 50 MHz clock enters the FPGA and passes through BUFGDLL which consists of a DLL and buffer for clock skew management. The output signal INT_CLK is a 50% duty cycle clock which is used by the controller, RAMB4_S8_S16, RAM and SR8RE. DAC_CLK is used to read out new samples from the RAM to the DAC. A slightly out of phase clock DEC_CLK is used to latch samples into the DAC. Another slightly out of phase clock DEC_CLKB is used to clock the analog decoder. The timing between three out of phase clocks will vary based on CLK_DIV. For instance if CLK_DIV is 0, then DEC_CLK will be one INT_CLK cycle out of phase with DAC_CLK. DEC_CLKB will be one INT_CLK cycle out of phase with DEC_CLK. This is to ensure that the DAC and the decoder has enough set up time to latch new values. The

decoding clock is dependent on how fast INT_CLK can operate. Using the standard 50 MHz input on the FPGA board for INT_CLK, the decoding clock DEC_CLKB is capable of testing from approximately 155 kSps (CLK_DIV = '11 1111') to 8.333 MSps (CLK_DIV = '00 0000') at a 40% duty cycle. The testing speed can be bumped up to 10 MSps if CLK_DIV operates on a 20% cycle. DAC_CLK and DEC_CLK have 20% duty cycles. This is sufficient since the RAM block is capable of operating at the FPGA speed of 50 MHz and the DAC is capable of operating at 125 MSps [16]. DAC_CLK, DEC_CLK, and DEC_CLKB only operate during decoding operation. The rest of the time these signals are '0'.

### 5.1.3 Test Support Board

The test support (TS) board is an intermediate board that sits between the FPGA controller and the device under test (DUT) board as shown in Fig. 5.3. A schematic of TS board is shown Fig. 5.8. The purpose of this board is to (1) provide digital to analog conversion (DAC) of samples, (2) provide voltage supplies for the DUT, and (3) provide USB communications to the PC.

The AD9764 is a 14-bit Analog Devices DAC [16] which generates differential output currents which can be converted into voltage with resistive loads. An output differential voltage is produced dependent on

$$V_{DIFF} = \frac{2 \cdot DACCODE - 16383}{16384} \cdot \frac{32 \cdot R_{LOAD}}{R_{SET}} \cdot V_{REFIO} \qquad (5.1)$$

where $DACCODE$ is a 14 bit input having the range of 0..16383, $V_{REFIO}$ is an internal $1.2V$ source, $R_{LOAD}$ is the resistive load, and $R_{SET}$ is a 20 kΩ trimmer potentiometer used for setting the reference current. This voltage is fed into a differential driver having a gain of [17]

$$\left| \frac{V_{OUT,dm}}{V_{IN,dm}} \right| = \frac{R_F}{R_G} \qquad (5.2)$$

where $R_F$ and $R_G$ are feedback and input resistors respectively. The common mode voltage $V_{OCM}$ of this differential buffer can be controlled via another 20kΩ trimmer potentiometer.

Hence, the swing of voltage samples are fully adjustable through $DACCODE$ and $R_{SET}$. For example, by choosing $R_{LOAD} = 10\Omega$, $R_G = 10\Omega$, $R_F = 20\Omega$, $1\text{k}\Omega \leq R_{SET} \leq 20\text{k}\Omega$, we get

$$38.4mV \leq \left|V_{OUT,dm(max)}\right| \leq 768mV \tag{5.3}$$

The upper limit of $V_{OUT,dm(max)}$ can be extended by adjusting $R_{SET}$ lower. The maximum output differential swing was measured to be about 2V.

The board features variable voltage supply sources generated by an AD8544 part connected in voltage follower configuration. These sources can generate voltages from 0.02 to 0.98 of $V_{DD}$ with current levels depending on the output voltage [18]. For example, with $V_{DD}$ of 3.3V, the output voltage can range approximately from 0.066V to 3.234V with output current levels of up to 18 mA when the output voltage is 2.3V. To support larger decoders or devices consuming higher currents, an LM117 voltage regulator is also available. This well known limiter produces voltages down to 1.2V with 1.5A of current.

These supplies will meet the needs of the low voltage decoder chip. The chip needs analog supply voltages of approximately 1V or less for its analog portion. Its I/Os need a digital supply of 1.8V to be recognizable to the chip's standard library pads. The chip's pads in turn need to be powered at 3.3V to interface with the DAC and FPGA board. Hence, we need AVDD < 1V, DVDD = 1.8V, and PVDD = 3.3V for the decoder chip to function.

To ensure fast rise times for digital signals, digital buffers are used for DAC_CLK, DEC_CLK, FRAME, and decoder output signals. Depending on the testing speed desired, careful attention must be paid to phase delay times $t_{pd}$. This time will vary according to the supply voltage and capacitive load. For example at $V_{DD}$ of 3.3V and $C_L = 50pF$, the SN74LV125A [38] buffer will delay the input phase by 13ns.

A USB module by DLP Design [15] is used by the FPGA controller for communication to and from the PC. This device acts like a serial port when connected to the PC. Data can be written to and read from the port using a library available from the DLP's website or other sources available on the Internet. The FPGA controller uses four handshaking signals to communicate

with the USB device. The module features a 384 byte FIFO transmit buffer and a 128 byte FIFO receive buffer. The module is capable of transferring up to 8 Mbps (1 MBps) *if data is transfered using these buffers.* If individual bytes are transfered, the speed of transfer is severely limited to 1ms per byte. The module is bus powered meaning that it uses power from the PC. Its I/O interface to the FPGA is separately maintained at 3.3V. The current test setup will transfer and receive one byte at a time limiting measurement speed to about 50 kSps since 2 bytes are needed to represent one test sample. Note that this does not affect decoder testing speed since samples are applied to the analog decoder according to CLK_DIV.

### 5.1.4 Device Under Test Board

Generally, chips will have different packaging and pin outs thereby necessitating a need for separate DUT boards. The DUT board holds the decoder chip and contains switches if needed to test a multi decoder chip. Power supply lines are routed to the test support board. Posts are used to attach a bias unit current. Test points and probes can be added to make test tracking and voltage probing easier.

## 5.2 Test Methodology

A number of variables needs to be tweaked in order to find the best operating point for the decoder. These include supply voltage AVDD, bias current $I_u$, and $V_{DIFF}$ on the test support board. It also includes LLR scaling, DAC scaling factor DAC_SCALE, and decoder testing speed CLK_DIV in the PC test program. A good starting point is to:

1. Adjust all supply voltages AVDD, DVDD, and PVDD making sure there is voltage going into the chip and proper grounding.

2. Adjust $V_{OCM}$ on the differential buffer equal to AVDD since this common voltage will be divided equally between the input sample and hold capacitors.

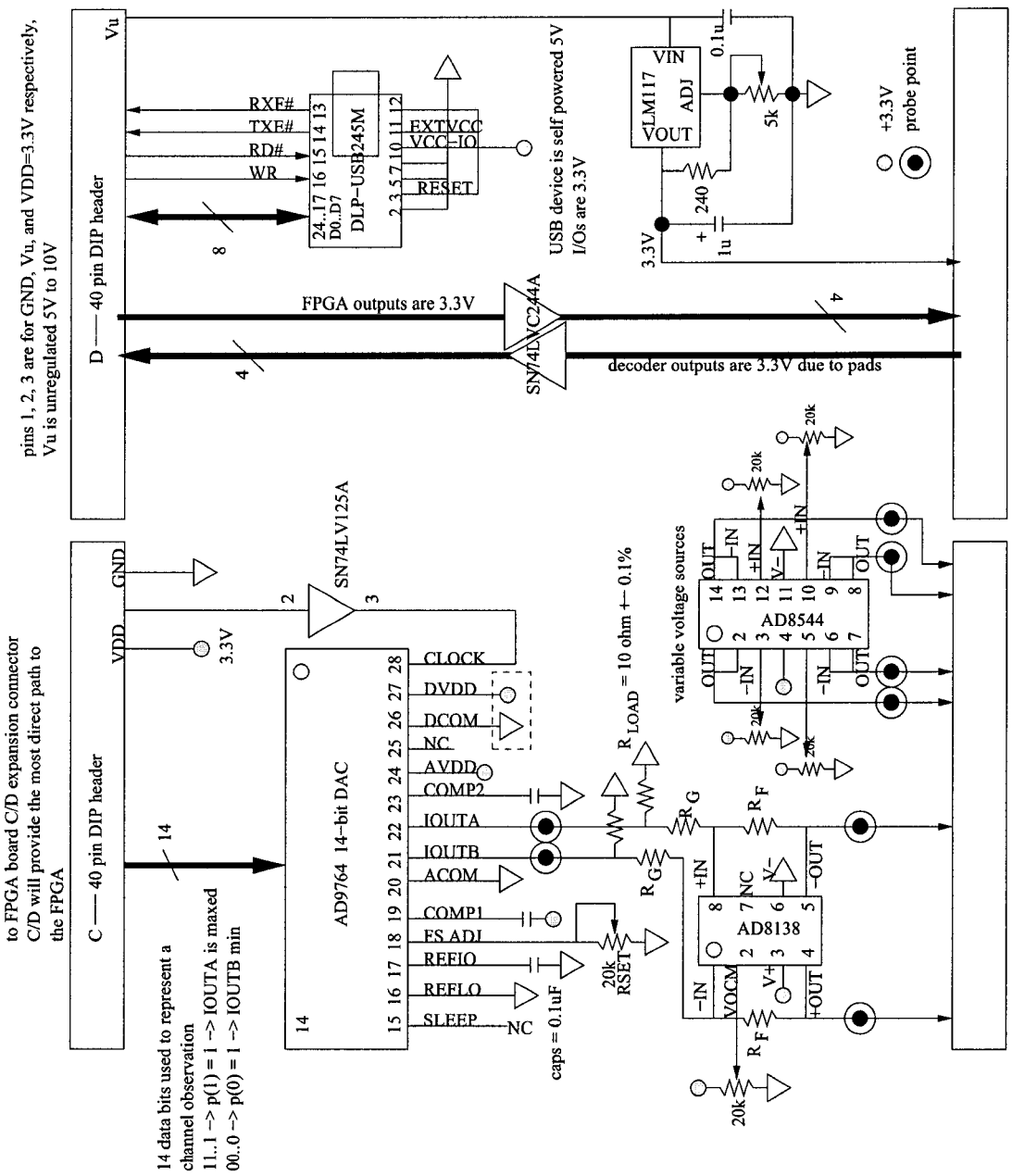



Figure 5.8: Schematic of test support board

3. Adjust $R_{SET}$ to its minimum to give the input LLR $V_{DIFF}$ maximum swing

4. Set DAC_SCALE to 4096 in software (a higher value will increase the resolution for LLR values close to 0)

5. Set maximum CLK_DIV = 63 in software to get the slowest decoder testing speed of 155 kSps. See Table 5.1 for other testing speeds.

6. Allow all decoded bit probabilities to be counted towards BER

Table 5.1: CLK_DIV setting and test speed

| 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 |

It is hard to describe a *systematic test strategy* since there are so many variables at work to finding the *sweet spot*. Hence we can start from the above and check its BER curve. Due to the poor performance of I/O interfaces used in this design (as will be explained in the next section), there might only be a few good performing bits. BER can still be calculated using these bits since the statistics of one bit are independent from the next. The best performing bit can be used as a best performance indicator. By adjusting $V_{OCM}$ and $R_{SET}$ we can control the common mode and differential input swing, respectively. We can also limit the swing of the DAC through controlling its upper and lower bound in software while maintaining resolution through DAC_SCALE. When these adjustments improve the BER, testing speed can be increased by lowering CLK_DIV. To obtain good BER at faster speed, $I_u$ and AVDD can also be increased.

## 5.3 Measurement Results

Two chips were designed and manufactured in a 1P6M TSMC 0.18$\mu$m process. The first chip, ICFAANN1, contains a factor graph decoder and an I/O test loop, as shown in Fig. 5.9. The factor graph decoder was found to be defective, but the I/O test loop was functional. The second chip, ICFAANN2, contains a corrected factor graph decoder and a trellis graph decoder, as shown in Fig. 5.10. These two decoders share the same die but their I/Os and pads are completely separated from each other. We describe the measurements for the I/O test loop in ICFAANN1 and both decoders in ICFAANN2.

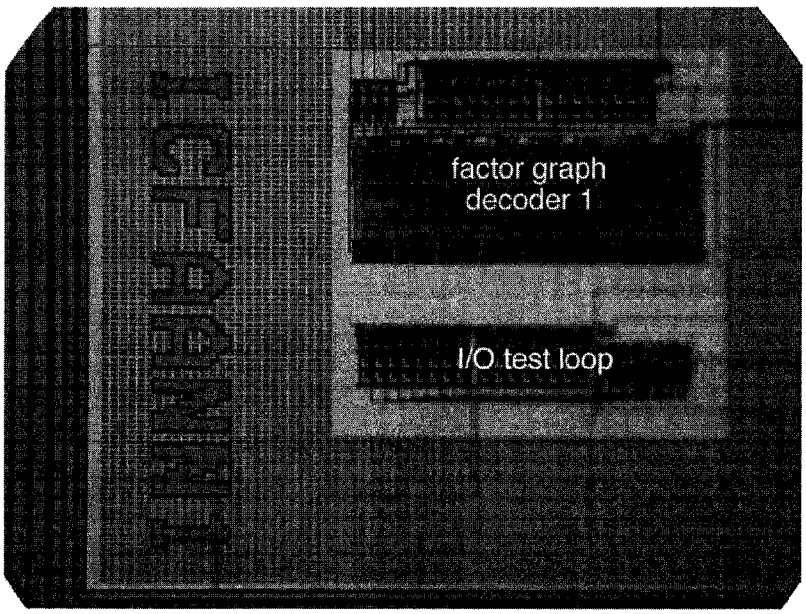



Figure 5.9: Die photo of chip ICFAANN1 showing (1) factor graph decoder and (2) test loop

### 5.3.1 I/O Test Loop

The test loop was constructed by feeding two sets of eight S/H chains into eight comparators. The loop is tested by injecting $\Delta V_{LLR}$ voltages and reading the comparator output. Ideally as discussed previously, if $V_{LLR+} > V_{LLR-}$, then a '0' will be output by the comparator and otherwise, a '1'. However, problems

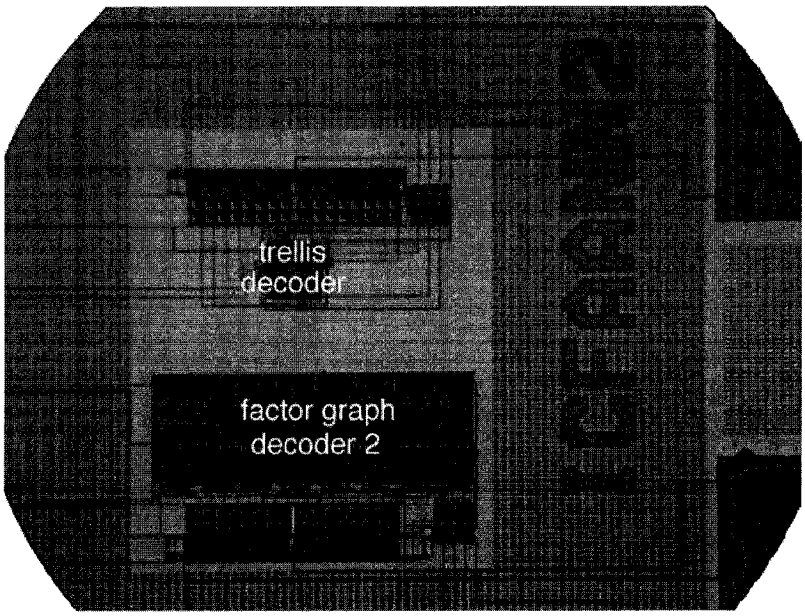



Figure 5.10: Die photo of chip ICFAANN2 showing (1) trellis graph decoder and (2) factor graph decoder

such as charge leakage on S/H capacitors and comparator DC offset decrease accuracy.

Capacitor charge leakage occurs mainly because an extra pass transistor is added in parallel for discharging before holding a new voltage. Even when the pass transistor is off, there will be leakage since the transistor is not a perfect switch, as shown in Fig. 5.11. If this leakage is equal for both $V_{LLR+}$ and $V_{LLR-}$, then it should not introduce overall errors. However, when too much time is taken between sampling and transfer to hold, a significant amount of leakage can occur reducing signal strength. This is more of a problem in larger S/H chains. To lessen the effects of leakage, a larger capacitor can be used as well as a smaller discharge transistor, leading to a slower chain.

Comparator DC offsets occur because of mismatch and can be defined as an input voltage bias needed to bring the output differential voltage to 0 [54]. As shown in Fig. 5.12, an offset on one input introduces errors when this input is compared to the other. For example, if $Vin(0) = 0.45V$ and $Vin(1) = 0.55V$, the output should be '1' since $Vin(0) < Vin(1)$. If there is an additional DC

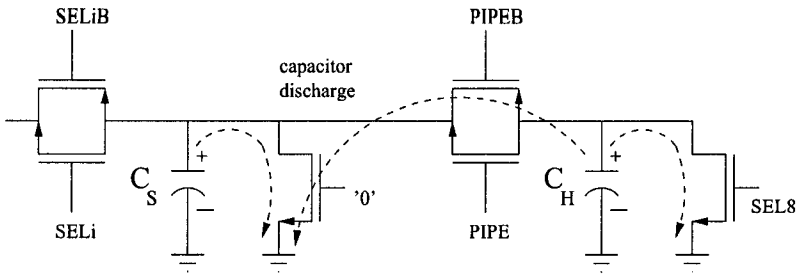



Figure 5.11: Capacitor discharge through imperfect switches

offset of 0.2 V on $Vin(0)$ then $Vin(0) > Vin(1)$ and the output is '0'.

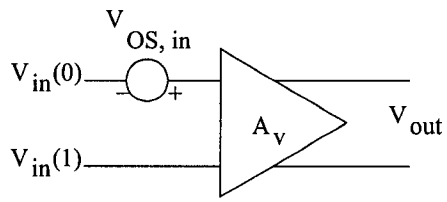



Figure 5.12: DC offset on a comparator

Capacitor discharge could potentially degrade input SNR for long block lengths while comparator offset could give wrong bit decisions. Therefore comparator offset is considered to be more serious. In fact, when the test loop is characterized, only a few bits out of 8 perform according to uncoded BPSK as shown in Fig. 5.13. In that figure two bits perform as expected and fall almost on the uncoded BPSK curve while the others had BER rates which were almost flat averaging around 0.5.

The implemented decoders using these interfaces will suffer similar problems at the input and output. In the factor graph decoder chip, 2 out of 4 output information bits were used to measure the BER. In the trellis decoder chip, 1 out of 4 outputs were used. The unused bits appear to suffer from comparator offset errors. They raised BER significantly when included in the calculation.

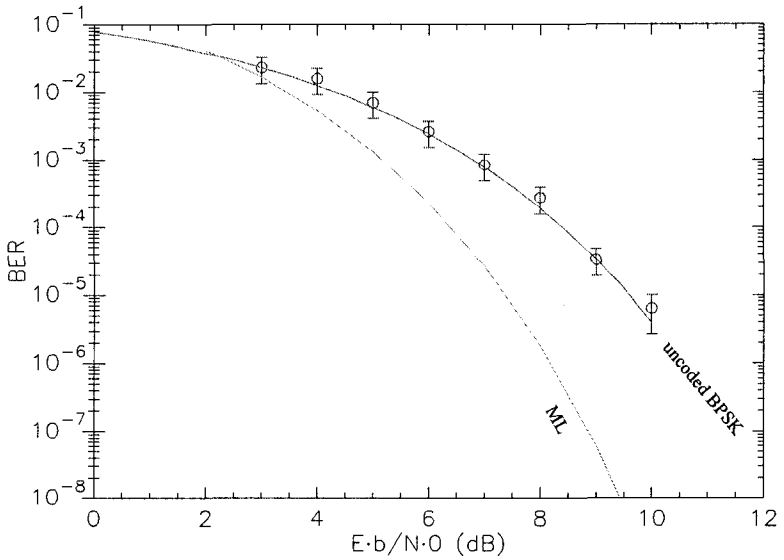



Figure 5.13: Measurement of test loop using error probabilities on 2 out of 8 bits

### 5.3.2 Factor Graph Decoder

We first characterize BER results according to varying test speed (or I/O CLK) and unit currents $I_u$ while holding other variables constant. The measured curves for varying test speed with $V_{DD} = 0.8V$ and $I_u = 10\mu A$ are shown in Fig. 5.14. A slower I/O clock speed allows the curve to move closer to ML. The reason is that it allows the decoder more time to settle and arrive at a correct result. Now, if $I_u$ is varied while keeping $V_{DD} = 0.8V$ and test speed=155 kSps, then we get the set of curves shown in Fig. 5.15. The results seem to indicate that a larger $I_u$ value is better for BER. This is because a larger $I_u$ value contributes to faster settling time. Recall that the maximum allowable $I_u$ value depends on the supply and has an approximate upper bound dictated by (3.38) in Section 3.2.2. The relationship between $V_{DD}$, $I_u$, and test speed can be summarized by Fig. 5.16. To get the same BER curve at different $V_{DD}$, both $I_u$ and test speed need adjusting. A decoder with $V_{DD} = 0.8V$ and current bias $I_u = 10\mu A$ can operate at almost an order of magnitude higher speed than a decoder with $V_{DD} = 0.5V$ and $I_u = 0.5\mu A$.

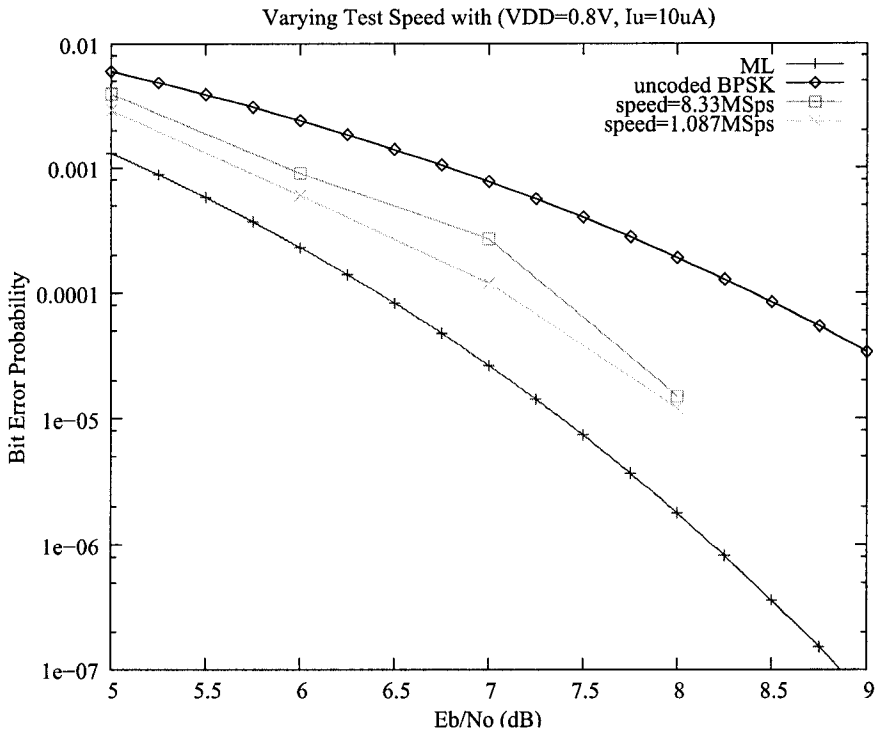



Figure 5.14: Factor graph measurement results with varying test speed

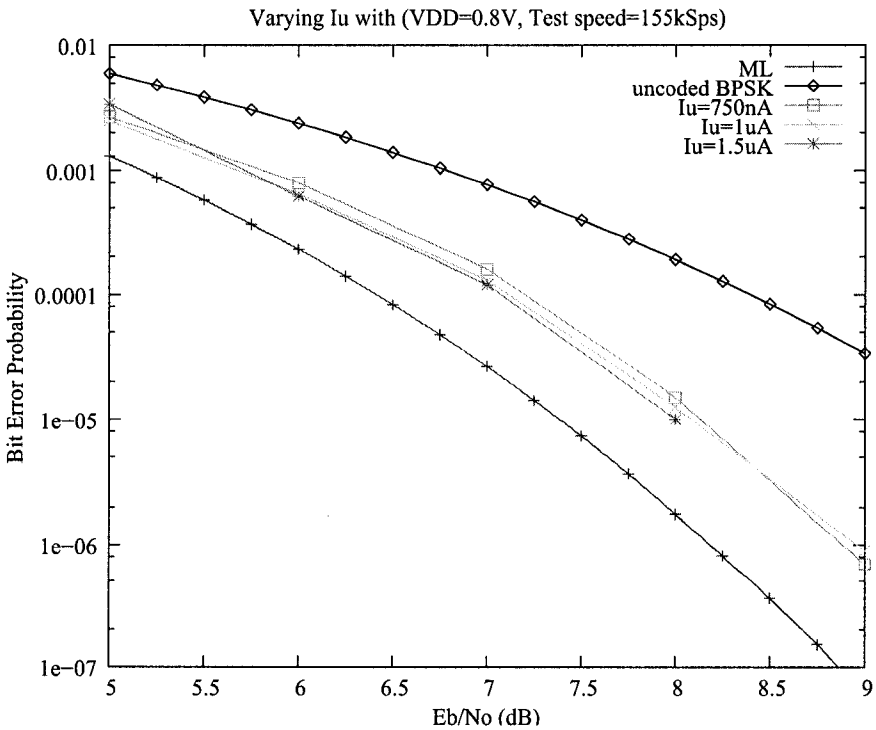



Figure 5.15: Factor graph measurement results with varying $I_u$

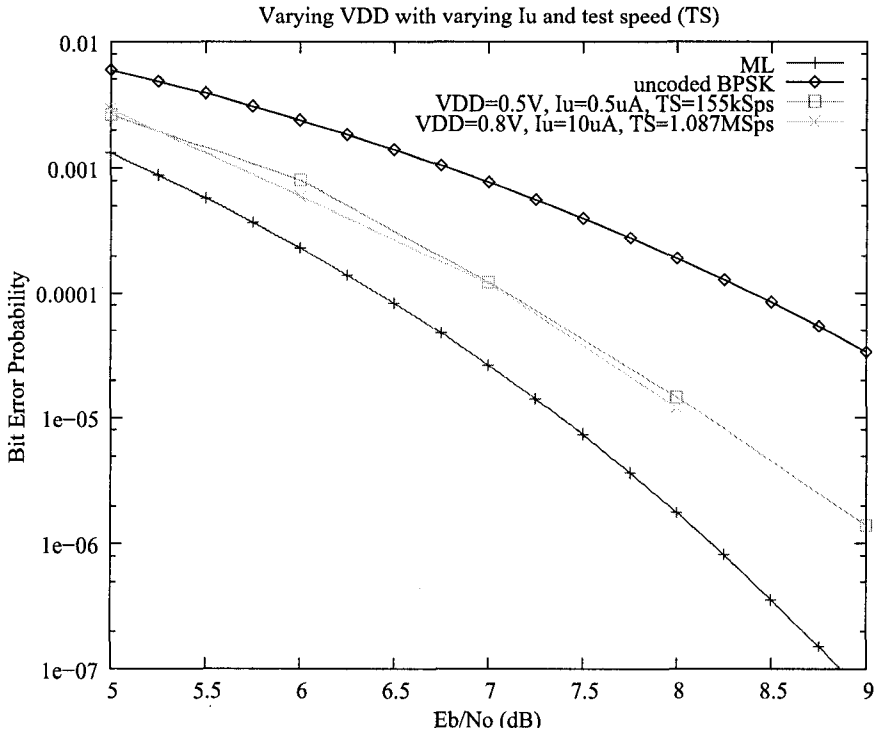



Figure 5.16: Factor graph measurement results with varying $V_{DD}$

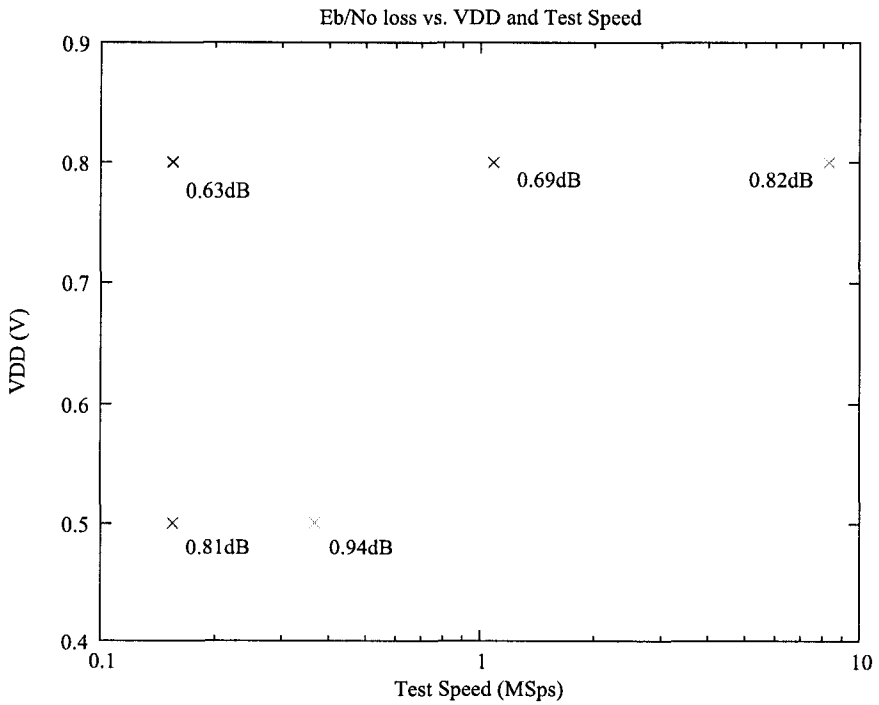



Figure 5.17: Factor graph measurements showing $V_{DD}$ vs. test speed and SNR loss (measured at the highest available SNR)

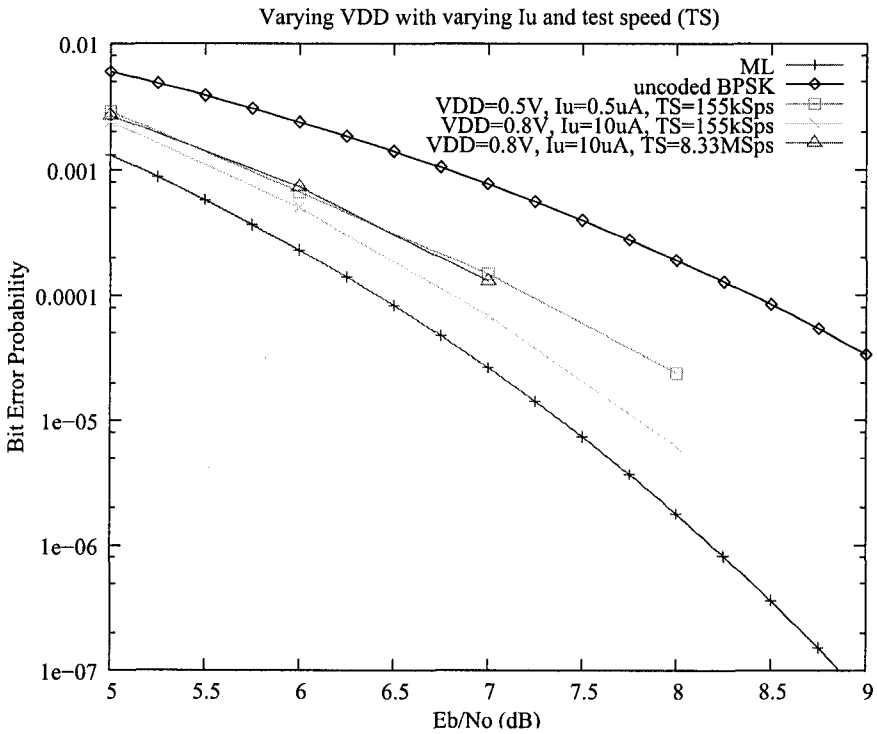



Figure 5.18: Trellis graph measurement results with varying $V_{DD}$

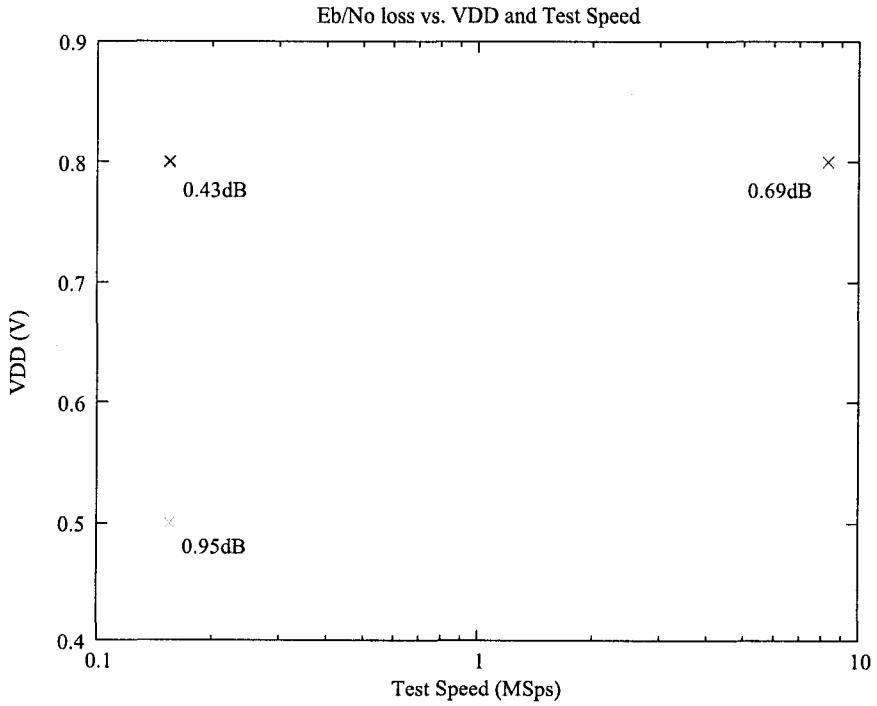



Figure 5.19: Trellis graph measurement showing $V_{DD}$ vs. test speed and SNR loss (measured at the highest available SNR)

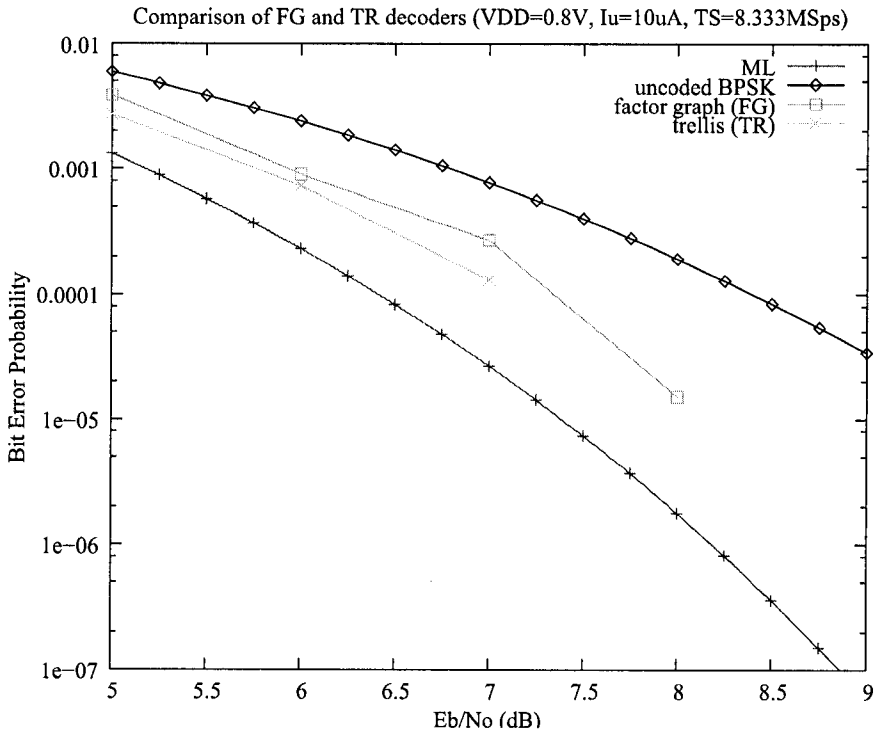



Figure 5.20: Trellis vs. factor graph decoder measurements at $V_{DD} = 0.8V$

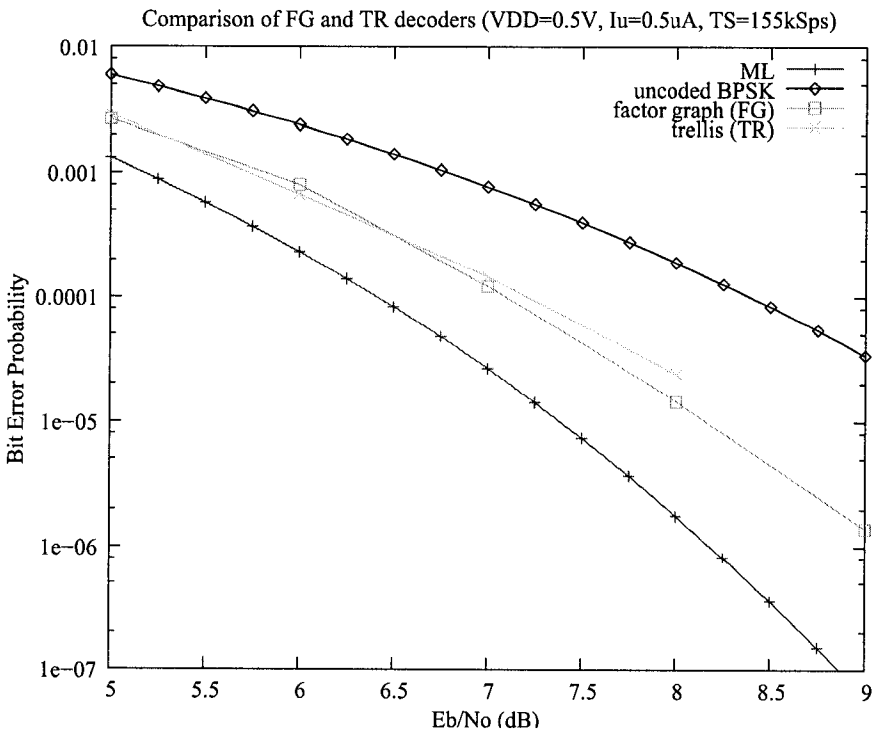



Figure 5.21: Trellis vs. factor graph decoder measurements at $V_{DD} = 0.5V$

The measured BER curves are 0.6dB to 1dB off of ML decoding. The losses are larger at lower $V_{DD}$ and higher test speed as shown in Fig. 5.17. The x-axis used to indicate test speed is in log scale. At low $V_{DD}$, the allowable test speeds are significantly lower. For example, if the speed is increased beyond 500 kSps with $V_{DD} = 0.5V$, the BER starts to approach uncoded BPSK. From this crude observation, it can be said that valid measurements can only be recorded in the top half triangle of the graph.

### 5.3.3 Trellis Graph Decoder

The trellis graph decoder can be characterized in the same way as the factor graph decoder. A slower testing speed and higher $I_u$ will improve BER since they will allow the decoder more time to settle. The interplay between these variables are shown in Fig. 5.18. A small drop in $V_{DD}$ needs a larger drop in $I_u$ and test speed to get the same BER curve.

Available measurement results are shown on Fig. 5.19. We can compare these results to the factor graph decoder's results in Fig. 5.17. At $V_{DD} = 0.8V$ and test speed = 8.333 MSps, the trellis decoder has a loss of 0.69dB while the factor graph decoder incurs 0.82dB. This is better shown when we plot both their measured results on one graph as shown in Fig. 5.20. The preliminary results suggest better performance for the trellis decoder at higher supply voltages (accompanied with higher test speeds). This reinforces the SPICE simulation results mentioned in the previous chapter where the trellis decoder was simulated using a clock speed of up to 10 MHz. A comparison can also be done at $V_{DD} = 0.5V$ and test speed = 155 kSps shown in Fig. 5.21. In this case both curves look comparable up to SNR = 7dB. The factor graph curve performs better at SNR = 8dB. However, these comparisons might not be reliable due to the small number of measurements that we have. More results are needed to solidify these observations.

A summary of both decoders is given in Table 5.2. The power consumption of the analog portion, digital I/Os, and pads cannot be measured with the equipment we currently have. The Keithley source/current measure unit is already used to bias the unit current. Another Keithley is needed to measure

currents less than 1mA accurately. When a regular power supply is used, the current recorded is less than '0.000A' indicating that it less than 1mA. The analog portion, digital I/Os, and pads measured '0.000A' each. From Spectre simulation results, the factor graph analog portion and digital I/Os consumed less than 300$\mu$W. The trellis graph analog portion and digital I/Os consumed less than 40$\mu$W. Using these simulated results and the fact that measured current consumption of each component was less than 1mA, we conclude that power consumption is less than 1mW for each decoder (with I/Os and pads included).

Table 5.2: Decoder Implementations Summary

| | Factor graph | Trellis graph |
|---|---|---|
| Code | (8,4,4) Hamming | (8,4,4) Hamming |
| Application areas | LDPC codes | Turbo codes |
| Technology | TSMC 0.18$\mu$m 1P6M | TSMC 0.18$\mu$m 1P6M |
| Analog Area | 100 x 275 $\mu m^2$ (1336 nfets, 640 pfets) | 50 x 50 $\mu m^2$ (328 nfets, 184 pfets) |
| I/O Area | 58 x 275 $\mu m^2$ (468 nfets, 443 pfets, 32 mimcaps) | 58 x 275 $\mu m^2$ (468 nfets, 443 pfets, 32 mimcaps) |
| VDD | 0.5V to 0.8V (tested) | 0.5V to 0.8V (tested) |
| Power | < 1mW | < 1mW |
| Clock speed | up to 8.3 MSps | up to 8.3 MSps |
| Information rate | up to 3.7 Mbps | up to 3.7 Mbps |

## 5.4 Chapter Summary

In this chapter, we described the low voltage analog decoding test setup by breaking it up into individual submodules and explaining its operation. We described the test methodology used to make BER measurements. Then BER measurement results were presented for two implemented decoders operating at below 0.8V. We proceed now with the conclusion of this work.

# Chapter 6

# Conclusion

## 6.1 Summary of Completed Work

We have presented analysis and IC test results for a low voltage sum product circuit with extra transistors added to produce a constant denominator. The minimum allowable supply voltage using this circuit is dependent on the bias unit current and specific process parameters. The number of extra transistors added depends on the number of symbols used (they add negligible area when the number of symbols is small). These sum product circuits eliminate the need for reference voltages. The bias of the decoder comes down to just simply the supply and unit bias current.

We used the low voltage sum product circuit to construct two (8,4,4) Hamming analog decoders capable of operating below 0.8V. These decoders operate on small codes and are used as proof of concept. The structures implemented can be applied to larger sized decoders operating on LDPC and Turbo style codes. We have shown that these decoders can operate down to $V_{DD} = 0.5V$. The reduction in voltage needs a larger reduction in unit bias current and much reduced speed to get similar performance. More measurement results are needed to back up this claim. When operated at $V_{DD} = 0.8V$, however, both decoders are capable of decoding up to 3.703 Mbps.

The power consumption cannot be measured with the equipment we have. This is because the current drawn on all power supplies (pads, I/O, and analog decoder) were all less than 1mA. Hence we estimated the power consumption to be less than 1mW but it could actually be less than 0.5mW.

We have also designed an implemented an inexpensive low voltage analog decoding test bench using off the shelf components. This test bench is capable of testing up to 8.333 MSps (coded). Tests are controlled by a computer program which displays BER measurements in real time.

We have further proven the robustness of analog decoding against analog effects. The reason is threefold. First, we are operating in CMOS using a sub-1V supply where analog effects run amuck. Second, unlike many other previously designed decoders, we used transistor sizes only slightly larger than minimum. These smaller sized transistors are more prone to mismatch. Third, the emphasis in layout was to get the smallest possible die area by packing transistors as close as possible. The result is that adjacent signals couple more easily to each other adding further interference. Despite this however, the decoder was still able to perform with only 0.5dB to 1dB SNR loss when compared to ML decoding.

## 6.2 Future Work

More needs to be done to better understand low voltage sum product circuits and analog decoding in general. This section looks at some problems starting with the ones we encountered and working outward to a big picture perspective.

First, there were problems with the I/O interface which gave incorrect output bits. More design and characterization needs to be done to this end to find reliable I/O circuits. It is critical to get comparators working properly since they can give incorrect decisions. More effort can also be spent on the input S/H to mitigate effects such as charge leakage on longer length chains.

We used a supply of 1.8V for the I/Os to interface with 3.3V pads. Perhaps lower supply I/Os operating at below 1V could be designed to work on the same voltage as the analog decoder. Now imagine if this decoder exists in a system-on-chip environment. Is it possible for digital decoders to operate at below 1V as well? Can they use the same voltage?

Providing a supply voltage might be easy, but what about the bias current?

How can we get an accurate, standalone current on the order of $\mu A$? The current is even more crucial when the supply voltage is lowered since only a limited amount is allowed before the sum product module ceases to operate.

Now let us look at the low voltage sum product circuit. A question that was raised is what is causing the negative symmetry in the equality node. What happens when so many current terms are discarded? It might be helpful to model this in a simulation to see what effect it has on the BER. It was also observed that in the middle regions, where probabilities are 0.5, there is almost no difference between the implemented and ideal LLR. Perhaps a simple simulation study can be done to find out where bit flipping occur.

A similar probability-based mismatch study can be done on decoders built with low voltage sum product circuits to find out how mismatch affects the BER. The mismatch can be modeled as a difference in mirrored current as mentioned in Sec. 3.5 (modeling methods which looks at actual SPICE parameters might be too detailed and cumbersome for simulation). It is also possible to add a few extra transistors outside of the decoder during manufacturing runs as was done by [21]. Measurements can be made on these transistors to get an idea of mismatch levels specific to that process. The transistor behaviour can also be used to further fine tune SPICE models.

This all ties into finding the reason for SNR losses when compared to ML. Studies can be done to quantify how much loss is attributed to each aspect of analog decoding. They include transistor sizing, I/O interface, test setup, implementation (algorithm), and floating point losses (analog metrics do not have floating point precision).

Having demonstrated that low voltage analog decoders are feasible, their attractiveness to mobile devices becomes apparent. The characterization of these decoders must use practical channel models to get an idea of real world conditions.

One of the overlooked areas is the testing of analog decoders. To this end, a few researchers have been looking into how to test such decoders with efficiency and accuracy. Currently, efforts are made to add extra circuitry to test regular analog decoders that use higher supply voltages. These extra circuits should

not add significant overhead since that will undermine the advantages of analog decoding, namely, the principle that simple transistor arrays are used for signal processing and these arrays take up minimal area. If there is enough interest in regular voltage analog decoders then similar steps should be taken to test low voltage analog decoders.

# Bibliography


[1] A. S. Acampora. United States Patent 4,087,787, 1978.

[2] A. S. Acampora and R. P. Gilmore. Analog Viterbi decoding for high speed digital satellite channels. *IEEE Trans. on Communications*, 26:1463–1470, Oct. 1978.

[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] 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.

[5] S. Benedetto and E. Biglieri. *Principles of Digital Transmission: With Wireless Applications.* Plenum/Kluwer, New York, NY, 1999.

[6] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara. A soft-input soft-output APP module for iterative decoding of concatenated codes. *lEEE Communications Letters*, 1:22–24, Jan. 1997.

[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. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R.-H. Yan. A unified Turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-$\mu$ CMOS. *IEEE J. of Solid State Circuits*, 37(11):1555–1564, Nov. 2002.

[9] A. J. Blanksby and C. J. Howland. A 690 mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code decoder. *IEEE J. of Solid State Circuits*, 37(3):404–412, Mar. 2002.

[10] A. R. Calderbank, G. D. Forney Jr., and A. Vardy. Minimal tail-biting trellises: the Golay code and more. *IEEE Trans. on Information Theory*, 45(5):1435–1455, Jul. 1999.

[11] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. *IEEE J. of Solid State Circuits*, 27(4):473–484, Apr. 1992.

[12] S.-Y Chung, G. D. Forney Jr., T. J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 5:58–60, Feb. 2001.

[13] N. S. Correal, J. Heck, and M. C. Valenti. An analog Turbo decoder for an (8,4) product code. In *IEEE Midwest Symp. on Circuits and Sys.*, Tulsa, OK, Aug. 2002.

[14] J. Dai. *Design methodology for analog VLSI implementations of error control decoders.* PhD thesis, University of Utah, Salt Lake City, Dec. 2002.

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

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

[17] Analog Devices. AD8138 Data Sheet: Low Distortion Differential ADC Driver, 2003.

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

[19] P. Elias. Coding for noisy channels. In *IRE Conv. Rec.*, volume 4, pages 37–46, Mar. 1955.

[20] M. Frey, H.-A. Loeliger, F. Lustenberger, P. Merkli, and P. Strebel. Analog decoder experiments with subthreshold CMOS soft-gates. In *IEEE Int. Symp. on Circuits and Systems*, Bangkok, Thailand, May 2003.

[21] M. Frey, H.-A. Loeliger, F. Lustenberger, P. Merkli, and P. Strebel. Measurements on an analog (8,4,4) Hamming code decoder chip. Technical Report 200402, Signal and Information Processing Laboratory, ETH Zurich, Jan. 2004.

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

[23] V. Gaudet, R. Gaudet, and G. Gulak. Programmable interleaver design for analog iterative decoders. *IEEE Trans. on Circuits and Systems II*, 49(7):457–464, July 2002.

[24] V. Gaudet and G. Gulak. A 13.3Mbps 0.35um CMOS analog turbo decoder IC with a configurable interleaver. In *IEEE Int. Solid State Circuits Conference*, pages 148–149, 484, Feb. 2003.

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

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

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

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

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

[30] J. Hagenauer, M. Moerz, and E. Offer. Analog turbo networks in VLSI: The next step in turbo decoding and equalization. In *Proc. Int. Symp. on Turbo Codes*, pages 209–218, Bretangne, France, Sept. 2000.

[31] J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *IEEE Trans. on Information Theory*, 42:429–445, Mar. 1996.

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

[33] M. Helfenstein, H.-A. Loeliger, F. Lustenberger, and F. Tarkoy. United States Patent 6,282,559, 2001.

[34] M. Helfenstein, H.-A. Loeliger, F. Lustenberger, and F. Tarkoy. United States Patent 6,584,486, 2003.

[35] S. Hemati and A. H. Banihashemi. Iterative decoding in analog CMOS. In *Great Lakes Symposium on VLSI*, pages 15–20, Apr. 2003.

[36] Z.-Q. Hu, W. H. Mow, and W.-H. Ki. Analog integrated circuit design of a hypertrellis decoder. In *Fourth Int. Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT'2003)*, pages 552–556, Aug. 2003.

[37] W. Huang, V. Igure, G. Rose, Y. Zhang, and M. Stan. Analog Turbo decoder implemented in SiGe BiCMOS technology. In *40th Design Automation Conference Student Contest*, June 2003.

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

[39] G. D. Forney Jr. *Concatenated codes.* PhD thesis, Massachusetts of Technology (MIT), Cambridge, MA, 1965.

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

[41] 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.

[42] 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.

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

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

[45] F. Lustenberger, M. Helfenstein, H.-A. Loeliger, F. Tarkoy, and G. S. Moschytz. An analog VLSI decoding technique for digital codes. In *IEEE Int. Symp. on Circuits and Systems*, pages 428–431, Orlando, FL, June 1999.

[46] F. Lustenberger and H.-A. Loeliger. On mismatch errors in analog-VLSI error correcting decoders. In *IEEE Int. Symp. on Circuits and Systems*, pages 198–201, Sydney, Australia, May 2001.

[47] 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.

[48] T. W. Matthews and R. R. Spencer. An integrated analog CMOS Viterbi detector for digital magnetic recording. *IEEE J. of Solid State Circuits*, 28(12):1294–1302, Dec. 1993.

[49] C. A. Mead. *Analog VLSI and neural systems.* Addison Wesley Computation and Neural Systems Series, Addison Wesley, Reading, MA, 1989.

[50] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. An analog 0.25um BiCMOS tailbiting MAP decoder. In *IEEE Int. Solid State Circuits Conference*, pages 356–357, San Francisco, CA, Feb. 2000.

[51] A. F. Mondragon-Torres, E. Sanchez-Sinecio, and K. R. Narayanan. Floating-gate analog implementation of the additive soft-input soft-output decoding algorithm. *IEEE Trans. on Circuits and Systems I*, 50(10):1256–1269, Oct. 2003.

[52] R. H. Morelos-Zaragoza. *The Art of Error Correcting Coding.* John Wiley & Sons, 2002.

[53] 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 I, pages 1116–1119, May 2004.

[54] B. Razavi. *Design of analog CMOS integrated circuits.* McGraw-Hill, New York, NY, 2001.

[55] P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. In *IEEE Int. Conf. on Communications*, volume 2, pages 1009–1013, Seattle, WA, 1995.

[56] E. Seevinck, E. A. Vittoz, M. du Plessis, T-H. Joubert, and W. Beetge. CMOS translinear circuits for minimum supply voltage. *IEEE Trans. on Circuits and Systems II*, 47(12):1560–1564, Dec. 2000.

[57] T. Serrano-Gotarredano, B. Linares-Barranco, and A. G. Andreou. A general translinear principle for subthreshold MOS transistors. *IEEE Trans. on Circuits and Systems I*, 46(5):607–616, May 1999.

[58] M. H. Shakiba, D. A. Johns, and K.W. Martin. BiCMOS circuits for analog Viterbi decoders. *IEEE Trans. on Circuits and Systems II*, 45(12):1527–1537, Dec. 1998.

[59] M. H. Shakiba, D. A. Johns, and K.W. Martin. An integrated 200 MHz 3.3V BiCMOS class-IV partial response analog Viterbi decoder. *IEEE J. of Solid State Circuits*, 33(1):61–75, Jan. 1998.

[60] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423 (part I), 623–656 (part II), July 1948.

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

[62] R. Tanner. A recursive approach to low complexity codes. *IEEE Trans. on Information Theory*, IT-27:533–547, Sept. 1981.

[63] B. Tomatsopoulos and A. Demosthenous. A low-power, hard-decision analogue convolutional decoder using the modified feedback decoding algorithm. In *IEEE Int. Symp. on Circuits and Systems*, volume IV, pages 181–184, May 2004.

[64] Various. 2nd analog decoding workshop. In *www.isi.ee.ethz.ch/adw*, Zurich, Switzerland, 2003.

[65] Various. 3rd analog decoding workshop. In *www.analogdecoding.org*, Banff, Canada, 2004.

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

[67] N. Wiberg, H.-A. Loeliger, and R. Kotter. Codes and iterative decoding on general graphs. *Eur. Trans. Telecomm.*, 6:513–525, Sep./Oct. 1995.

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

[69] 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.

[70] 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.

[71] 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.

[72] C. Winstead, N. Nguyen, V. Gaudet, and C. Schlegel. Low-voltage CMOS circuits for analog iterative decoders. submitted to IEEE Trans. on Circuits and Systems II, May 2004.

[73] 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.

[74] A. Xotta, D. Vogrig, A. Gerosa, A. Neviani, A. Graell-Amat, G. Montorsi, M. Bruccoleri, and G. Betti. An all-analog CMOS implementation of a turbo decoder for hard-disk drive read channels. In *IEEE Int. Symp. on Circuits and Systems*, pages 69–72, 2002.

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

[76] 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.

# Appendix A

# I/O Interface schematics

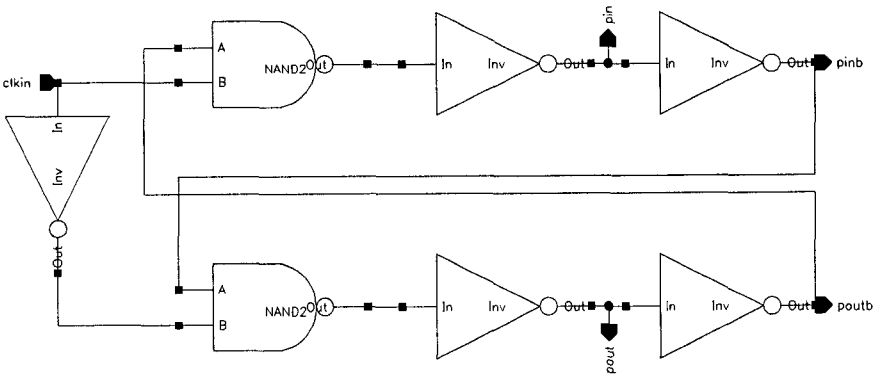
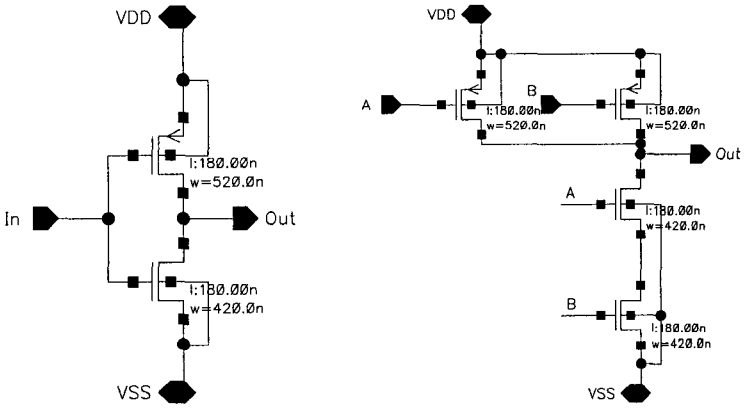



Figure A.1: Clock Generator




(a) Inverter (b) NAND2

Figure A.2: Digital Gates

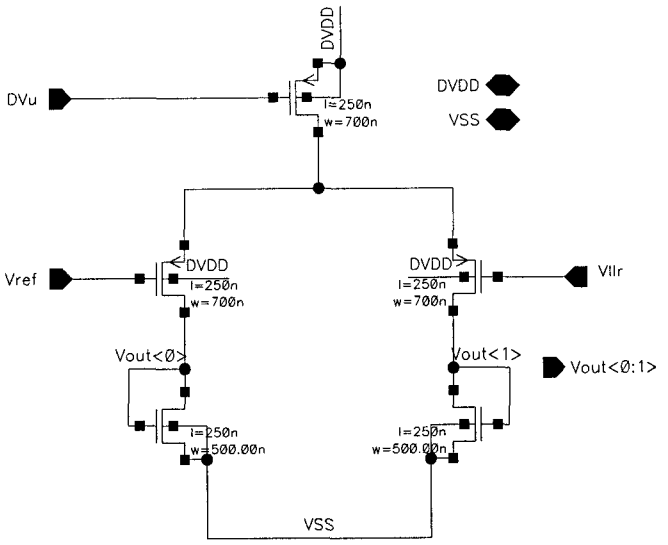



Figure A.3: LtoP : input LLR to probability converter

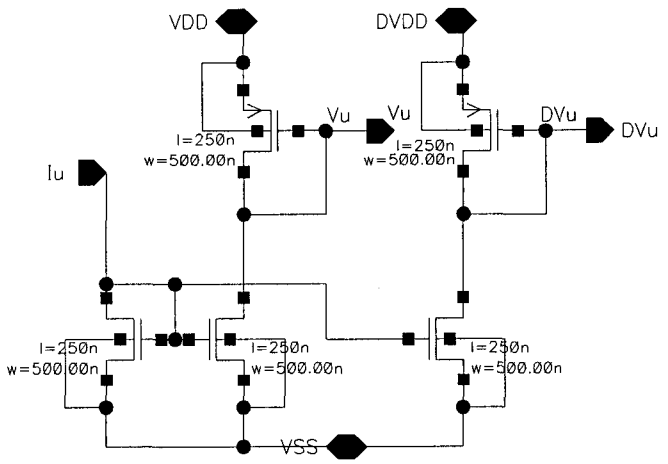



Figure A.4: input mirror : unit current mirror for I/O and analog decoder

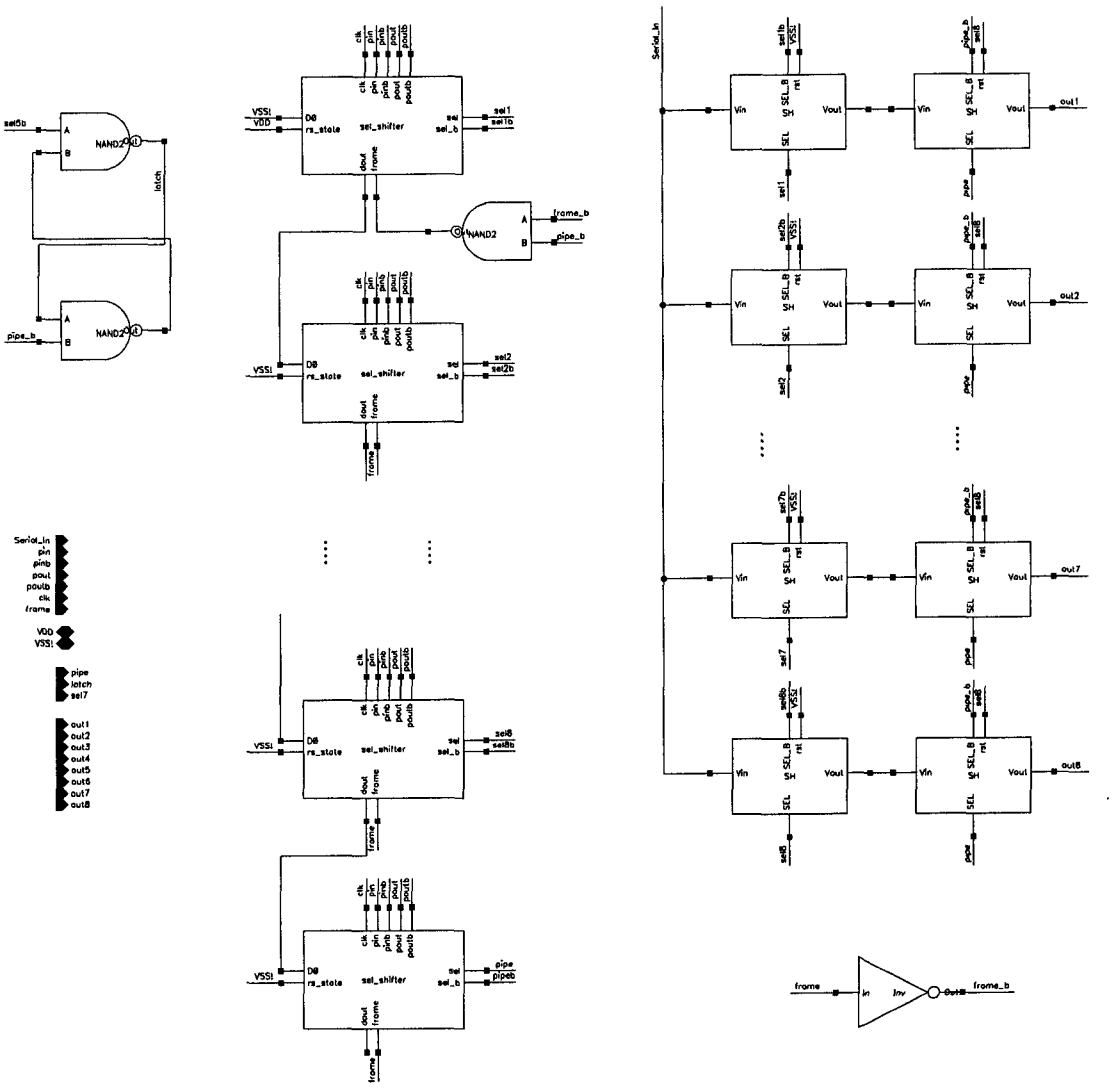


Figure A.5: Shift Register SH : input shift register and S/H chain

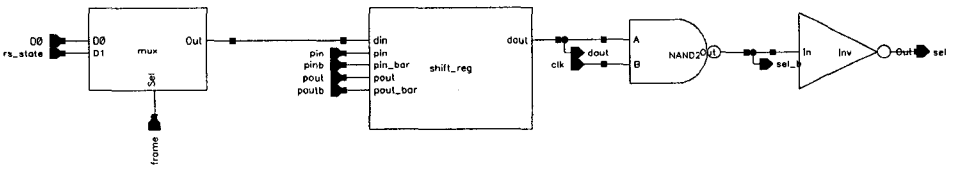


Figure A.6: selshifter : selectable shifter

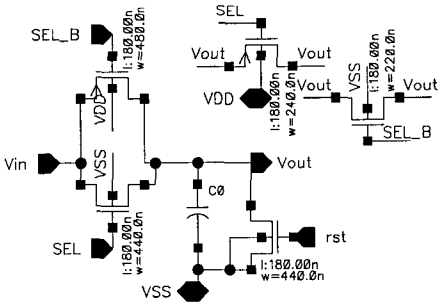



Figure A.7: SH : sample and hold capacitors

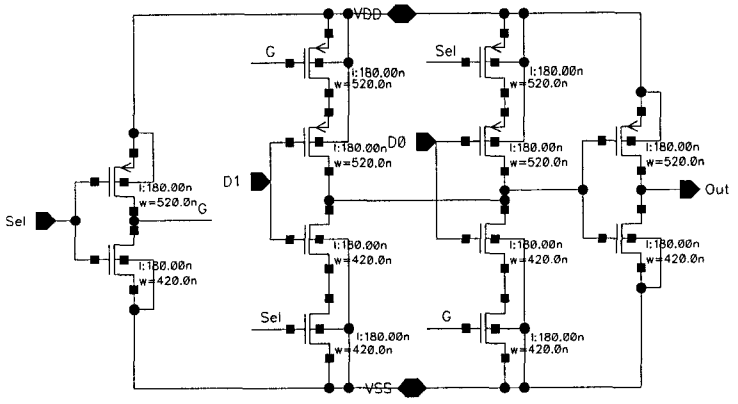



Figure A.8: mux : multiplexer

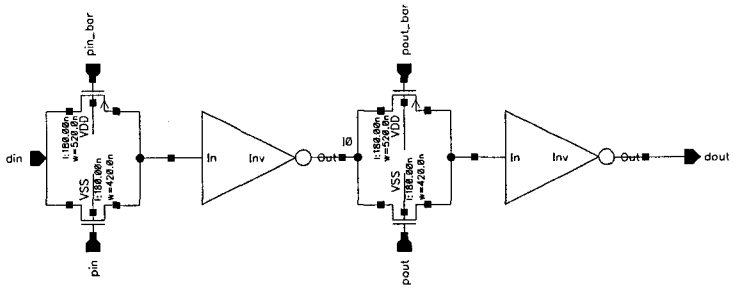



Figure A.9: shiftreg : level sensitive shift regsiter

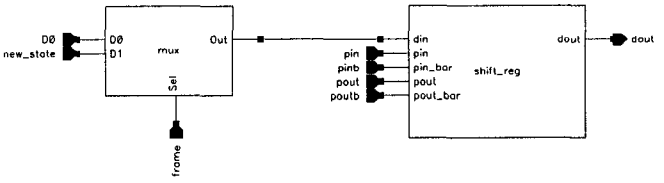



Figure A.10: shifter : output parallel load shifter

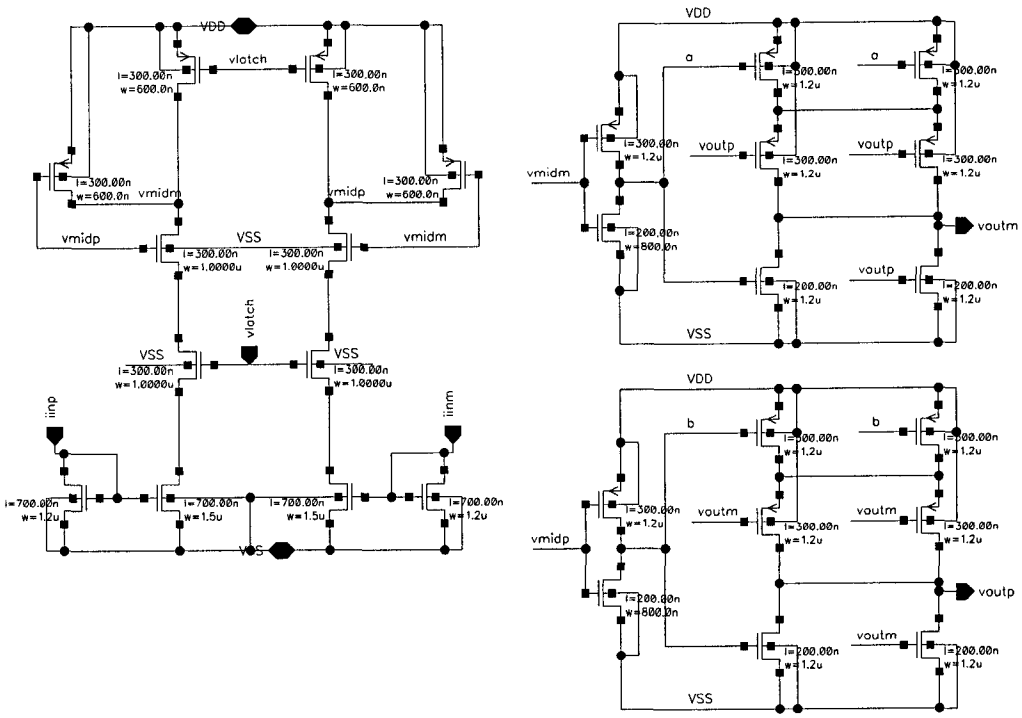



Figure A.11: Comparator : output comparator with SR latch

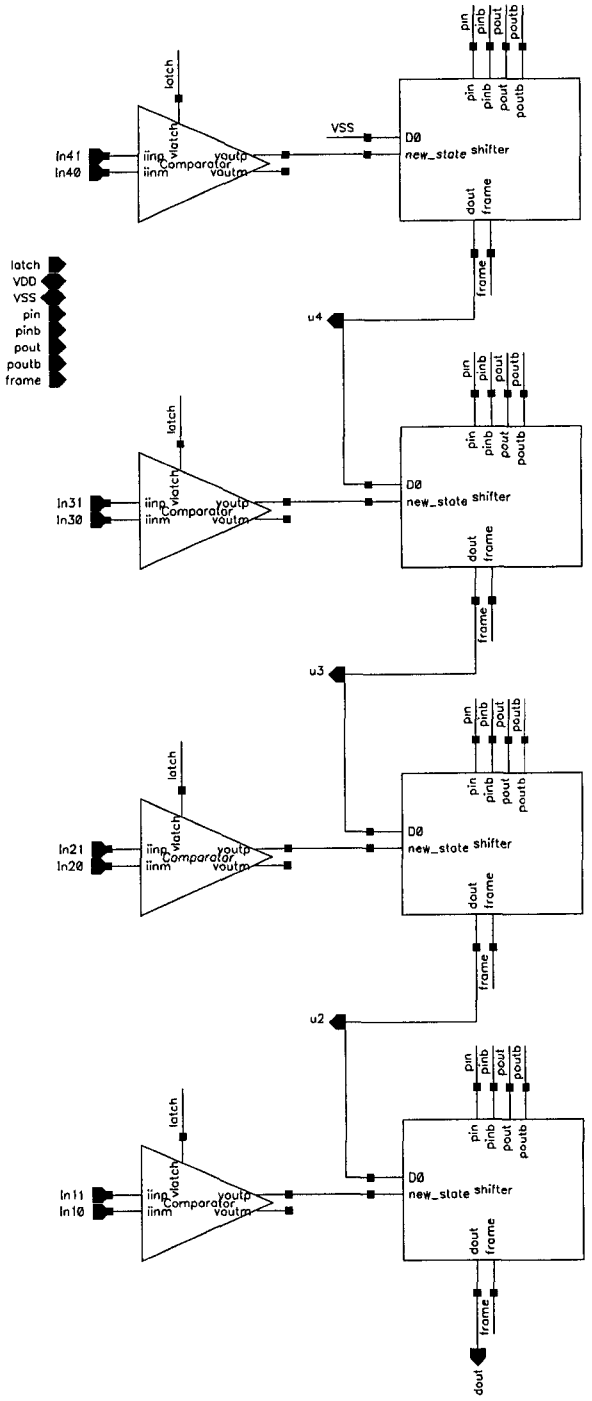



Figure A.12: Output Shifter : ouput shift register chain

# Appendix B

# Test Related

## B.1 Top level VHDL code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- instantiate Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity testmodule1611 is
    Port ( data : inout std_logic_vector(7 downto 0);
           rst : in std_logic;
           clk : in std_logic;
           fromdec : in std_logic;
     rxfb : in std_logic;
     txeb : in std_logic;
           todac : out std_logic_vector(15 downto 0);
     led : out std_logic;
     wr : out std_logic;
     rdb : out std_logic;
     frame : out std_logic;
     dac_clk : out std_logic;
     dec_clk : out std_logic;
     dac_clk_b : out std_logic
        );
end testmodule1611;

architecture mixed of testmodule1611 is

  signal int_clk, int_dac_clk, int_reset, int_dac_clk_b : std_logic;
  signal out_data : std_logic_vector(7 downto 0);
  signal in_data : std_logic_vector(7 downto 0);

  signal rx_txb : std_logic;
  signal Q_INT, mux_data_a, mux_data_b, mux_out : std_logic_vector(7 downto 0);
  signal addra : std_logic_vector(8 downto 0);
  signal addrb : std_logic_vector(7 downto 0);
  signal ena, wea : std_logic;
  signal muxcon : std_logic;

  signal gnd_vector : std_logic_vector(15 downto 0);
  signal logic0, logic1 : std_logic;

  component BUFGDLL
    port (O : out STD_ULOGIC; I : in STD_ULOGIC);
```

```
end component;
component BUFG
  port (O : out STD_ULOGIC; I : in STD_ULOGIC);
end component;
component IOBUF
  port (O : out STD_ULOGIC; IO : inout STD_ULOGIC;
    I : in STD_ULOGIC; T : in STD_ULOGIC);
end component;

COMPONENT controller1611
PORT(
  clk : IN std_logic;
  reset : IN std_logic;
  dusb : IN std_logic_vector(7 downto 0);
  txeb : IN std_logic;
  rxfb : IN std_logic;
  wr : OUT std_logic;
  rdb : OUT std_logic;
  rx_txb : OUT std_logic;
  frame : OUT std_logic;
  dac_clk : OUT std_logic;
  dec_clk : OUT std_logic;
  dac_clk_b : OUT std_logic;
  addra : OUT std_logic_vector(8 downto 0);
  dataa : OUT std_logic_vector(7 downto 0);
  addrb : OUT std_logic_vector(7 downto 0);
  wea : OUT std_logic;
  ena : OUT std_logic;
  muxcon : OUT std_logic;
  led : OUT std_logic
  );
END COMPONENT;

-- Component Declaration for RAMB4_Sm_Sn
-- Should be placed after architecture statement but before begin keyword
component RAMB4_S8_S16
port (DOA : out STD_LOGIC_VECTOR (7 downto 0);
  DOB : out STD_LOGIC_VECTOR (15 downto 0);
  ADDRA : in STD_LOGIC_VECTOR (8 downto 0);
  ADDRB : in STD_LOGIC_VECTOR (7 downto 0);
  CLKA : in STD_ULOGIC;
  CLKB : in STD_ULOGIC;
  DIA : in STD_LOGIC_VECTOR (7 downto 0);
  DIB : in STD_LOGIC_VECTOR (15 downto 0);
  ENA : in STD_ULOGIC;
  ENB : in STD_ULOGIC;
  RSTA : in STD_ULOGIC;
  RSTB : in STD_ULOGIC;
  WEA : in STD_ULOGIC;
  WEB : in STD_ULOGIC);
end component;
-- Component Attribute Specification for RAMB4_Sm_Sn
-- Should be placed after architecture declaration but before the begin keyword
-- Put attributes, if necessary

begin

logic0 <= '0';
logic1 <= '1';
gnd_vector <= (others => '0');
dac_clk <= int_dac_clk;
dac_clk_b <= int_dac_clk_b;

BUFGDLL_INSTANCE : BUFGDLL port map(O => int_clk, I => clk);
BUFG_INSTANCE : BUFG port map(O => int_reset, I => rst);

iobuf41: for i in 7 downto 0 generate
iobuf_instance : IOBUF port map (O => out_data(i), IO => data(i),
```

```
            I => in_data(i), T => rx_txb);
end generate;

 Inst_controller: controller1611 PORT MAP(
  clk => int_clk,
  reset => int_reset,
  dusb => out_data,
  txeb => txeb,
  rxfb => rxfb,
  wr => wr,
  rdb => rdb,
  rx_txb => rx_txb,
  frame => frame,
  dac_clk => int_dac_clk,
  dec_clk => dec_clk,
  dac_clk_b => int_dac_clk_b,
  addra => addra,
  dataa => mux_data_b,
  addrb => addrb,
  wea => wea,
  ena => ena,
  muxcon => muxcon,
  led => led
  );

 RAMB4_S8_S16_INSTANCE_NAME : RAMB4_S8_S16
  port map (
  DOA => in_data,
  DOB => todac,
  ADDRA => addra,
  ADDRB => addrb,
  CLKA => int_clk,
  CLKB => int_dac_clk,
  DIA => mux_out,
  DIB => gnd_vector,          -- unused
  ENA => ena,
  ENB => logic1,              -- always enable
  RSTA => int_reset,
  RSTB => int_reset,
  WEA => wea,
  WEB => logic0);             -- can never write using B


  -- a shift register macro modified to exclude CE
  process(int_reset, int_dac_clk_b)
  begin
    if (int_reset='1') then
      Q_INT <= (others => '0');
    elsif (int_dac_clk_b'event) and (int_dac_clk_b='1') then
      Q_INT <= Q_INT(6 downto 0) & fromdec;
    else
      Q_INT <= Q_INT;
    end if;
  end process;
  mux_data_a <= Q_INT;
  ------------------------------------------------------


  -- a mux macro ---------------------------------------
  process (mux_data_a, mux_data_b, muxcon) -- mux_data_b is from controller
  begin
    case muxcon is
      when '0' => mux_out <= mux_data_a;
      when '1' => mux_out <= mux_data_b;
      when others => NULL;
    end case;
  end process;
  ------------------------------------------------------
```

```
end mixed;
```

## B.2 Controller VHDL code

```
--*************************************************************************
--controller1611.vhd
--April-June, 2004
--This test bench controller operates in 6 phases:
--1. Detection of start command, saving of clock divider (6 bits)
--   and block length (1 byte which will be multiplied by 2)
--2. Storing of maximum 512 8 bit samples into Port A of dual port RAM
--3. Application of maximum 256 - 16 bit samples using Port B in
--   dual port RAM
--4. Clocking BL times for transient decoding
--5. Clocking BL times for shifing out results (in increments of 8
--   to latch bits into RAM)
--6. Send BL decoded bits from RAM via USB
--*************************************************************************

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity controller1611 is
  port (
    clk : in std_logic;
    reset : in std_logic;
    -- USB signals
    dusb  : in std_logic_vector(7 downto 0);  -- USB data lines
    txeb  : in std_logic;          -- USB tx handshaking
    rxfb  : in std_logic;        -- USB rx handshaking
    wr  : out std_logic;       -- USB write
    rdb : out std_logic;       -- USB read
    rx_txb  : out std_logic;       -- directional control for data lines
    -- DAC and decoder signals
    frame : out std_logic;       -- decoder reset
    dac_clk : out std_logic;       -- DAC and decoder I/O clock
    dec_clk : out std_logic;
    dac_clk_b : out std_logic;
    -- dual port RAM signals
    addra : out std_logic_vector(8 downto 0); -- RAM address  A
    dataa : out std_logic_vector(7 downto 0); -- RAM data A
    addrb : out std_logic_vector(7 downto 0); -- RAM address B
    wea     : out std_logic;       -- write enable
    ena   : out std_logic;       -- (read) enable A
    muxcon  : out std_logic;       -- DIA mux control
    -- LED output
    led   : out std_logic        -- visual feedback
  );
end controller1611;

architecture RTL of controller1611 is
--signals, variables, etc.

  type controller_state_type is (

  READ_PREPARE, TOGGLE_RD_ON, READ_BYTE, TOGGLE_RD_OFF,
  ANALYZE_BYTE, APPLY_SAMPLES_WAIT, APPLY_SAMPLES_PREPARE,
  DAC_CLK_L1, DAC_CLK_H1, DAC_CLK_H15, DAC_CLK_H2, DAC_CLK_L2, DAC_CLK_CHECK,
  WRITE_PREPARE, TOGGLE_WR_ON, TOGGLE_WR_OFF,
  WRITE_INCREMENT

  );
  signal controller_state : controller_state_type;
```

```
  type byte_state_type is (

  BYTE_0, BYTE_1, BYTE_2, BYTE_3

  );
  signal byte_state : byte_state_type;

begin

state_machine: process (clk)

  variable clock_divider : std_logic_vector(5 downto 0);
  variable block_length : std_logic_vector(11 downto 0);
  variable addressa : std_logic_vector(8 downto 0);
  variable addressb : std_logic_vector(7 downto 0);

  variable block_length_count : integer range 0 to 4095;
  variable clock_divider_count : integer range 0 to 63;

  variable block_length_mod8 : std_logic_vector(2 downto 0);

  variable delay_rd : integer range 0 to 2;
  variable delay : integer range 0 to 2;
  variable delay_wr : integer range 0 to 2;

  variable temp : std_logic_vector(7 downto 0); -- temp buffer

begin

  addra <= addressa;
  addrb <= addressb;

  if (clk'event and clk = '1') then
  if (reset = '1') then -- initialize system

    -- initialize outputs
    wr <= '0';     -- disable USB write
    rdb <= '1';    -- disable USB read
    rx_txb <= '0';     -- transmit for checking

    frame <= '0'; -- pull FRAME 0
    dac_clk <= '0'; -- pull DAC CLK 0
    dec_clk <= '0';
    dac_clk_b <= '0';

    dataa <= (others => '0');
    wea <= '0';    -- disable RAM port A write
    ena <= '0';    -- disable RAM port A

    muxcon <= '1';  -- set DIA to receive controller

    led <= '1';

    -- initialize variables

    addressa := (others => '1');  -- will be incremented right away
    addressb := (others => '0');

    clock_divider := (others => '0');
    block_length := (others => '0');

    temp := (others => '0');

    block_length_count := 0;
    clock_divider_count := 0;
    block_length_mod8 := (others => '1');

    delay_rd := 0;
```

```
    delay := 0;
    delay_wr := 0;

    byte_state <= BYTE_0;
    controller_state <= READ_PREPARE;

  else

      case controller_state is
-------------------------------------------------------------------------------
-- USB read sequence DO NOT MODIFY ------------------------------------------
    when READ_PREPARE =>
      rdb <= '1';    -- disable USB read
      wr <= '0';     -- disable USB write
      led <= '0';    -- turn LED off
      rx_txb <= '1';  -- receive mode

      if (delay_rd = 2) then
        delay_rd := 0;
        controller_state <= TOGGLE_RD_ON;
      elsif (rxfb = '0') then
        delay_rd := delay_rd + 1;
        controller_state <= READ_PREPARE;
      else
        controller_state <= READ_PREPARE;
      end if;

    when TOGGLE_RD_ON =>
      rdb <= '0';    -- TOGGLE RD ON
      wr <= '0';
      led <= '0';
      rx_txb <= '1';

    -- added this part to turn off RAM
    -- while making sure that a write occurs
      wea <= '0';
      ena <= '0';
    ------------------------------------

      if (delay_rd = 1) then  -- wait for 40 ns
        delay_rd := 0;
        controller_state <= READ_BYTE;
      else
        delay_rd := delay_rd + 1;
        controller_state <= TOGGLE_RD_ON;
      end if;

    when READ_BYTE =>
      rdb <= '0'; -- wait for another 20 ns
      wr <= '0';
      led <= '0';
      rx_txb <= '1';

      temp := dusb; -- READ BYTE
      controller_state <= TOGGLE_RD_OFF;

    when TOGGLE_RD_OFF =>
      rdb <= '1'; -- TOGGLE RD OFF
      wr <= '0';  -- disable USB write
      led <= '0'; -- turn LED off
      rx_txb <= '1';

      if (delay_rd = 1) then
        delay_rd := 0;
        controller_state <= ANALYZE_BYTE;
      elsif (rxfb = '1') then
        delay_rd := delay_rd + 1;
        controller_state <= TOGGLE_RD_OFF;
```

```
    else
      controller_state <= TOGGLE_RD_OFF;
    end if;
-- USB read sequence DO NOT MODIFY ----------------------------------------
-----------------------------------------------------------------------------

    when ANALYZE_BYTE =>
    case byte_state is

      when BYTE_0 =>
        if (temp(7 downto 6) = "11") then
          byte_state <= BYTE_1;
          clock_divider := temp(5 downto 0);
          controller_state <= READ_PREPARE;
        else
          controller_state <= READ_PREPARE;
        end if;

      when BYTE_1 =>
        byte_state <= BYTE_2;
        block_length(11 downto 4) := temp(7 downto 0);
        block_length_count := 1;   -- have to put it here because XST will reset to 0
        controller_state <= READ_PREPARE;

      when BYTE_2 =>
        if (temp(7 downto 6) = "00") then
          byte_state <= BYTE_0;
          controller_state <= READ_PREPARE;
        else
          byte_state <= BYTE_3;
          controller_state <= READ_PREPARE;
        end if;

      when BYTE_3 =>

        ena <= '1'; -- enable RAM port A
        wea <= '1'; -- write RAM port A
        dataa <= temp;
        addressa := addressa + 1;

        if (block_length_count = block_length) then

-- modified this for Chris' test program to go back to beginning
          byte_state <= BYTE_0; -- look for a stop byte next

          block_length_count := 0;  -- to account for FRAME
          controller_state <= APPLY_SAMPLES_WAIT;
        else
          block_length_count := block_length_count + 1;
          controller_state <= READ_PREPARE;
        end if;

      when others => NULL;
    -- if using READ_PREPARE, then won't have extra FFDs

    end case;

  when APPLY_SAMPLES_WAIT =>

    if (delay = 2) then
      delay := 0;
      controller_state <= APPLY_SAMPLES_PREPARE;
    else
      delay := delay + 1;
      controller_state <= APPLY_SAMPLES_WAIT;
    end if;

  when APPLY_SAMPLES_PREPARE =>
```

```
    ena <= '0';    -- disable RAM port A
    wea <= '0';    -- disable RAM port A write
    frame <= '1';        -- can possibly make this '1' to start with
    addressa := (others => '1');
    addressb := (others => '1');
    block_length := '0' & block_length(11 downto 1);  -- real block length
    muxcon <= '0';  -- get DIA ready to receive SR8 outputs
    controller_state <= DAC_CLK_L1;

  when DAC_CLK_L1 =>
    dac_clk <= '1';
    dec_clk <= '0';
    dac_clk_b <= '0';

    if (clock_divider_count = clock_divider) then
      controller_state <= DAC_CLK_H1;
      clock_divider_count := 0;
    else
      controller_state <= DAC_CLK_L1;
      clock_divider_count := clock_divider_count + 1;
    end if;

  when DAC_CLK_H1 =>
    dac_clk <= '0';
    dec_clk <= '1';
    dac_clk_b <= '0';

    if (clock_divider_count = clock_divider) then
      controller_state <= DAC_CLK_H15;
      clock_divider_count := 0;
    else
      controller_state <= DAC_CLK_H1;
      clock_divider_count := clock_divider_count + 1;
    end if;

  when DAC_CLK_H15 =>
    dac_clk <= '0';
    dec_clk <= '0';
    dac_clk_b <= '1';

    if (clock_divider_count = clock_divider) then

      if (delay = 2 and block_length_mod8 = 7) then
        controller_state <= DAC_CLK_H2;
        addressa := addressa + 1;
        wea <= '1';
        ena <= '1';
      else
        controller_state <= DAC_CLK_H2;
        clock_divider_count := 0;
      end if;

    else
      controller_state <= DAC_CLK_H15;
      clock_divider_count := clock_divider_count + 1;
    end if;

  when DAC_CLK_H2 =>
    dac_clk <= '0';
    dec_clk <= '0';
    dac_clk_b <= '1';

    if (clock_divider_count = clock_divider) then
      controller_state <= DAC_CLK_L2;
      clock_divider_count := 0;
    else
      controller_state <= DAC_CLK_H2;
      clock_divider_count := clock_divider_count + 1;
```

```
    end if;

  when DAC_CLK_L2 =>
    dac_clk <= '0';
    dec_clk <= '0';
    dac_clk_b <= '0';

    if (clock_divider_count = clock_divider) then
      controller_state <= DAC_CLK_CHECK;
      clock_divider_count := 0;
    else
      controller_state <= DAC_CLK_L2;
      clock_divider_count := clock_divider_count + 1;
    end if;

  when DAC_CLK_CHECK =>
    frame <= '0'; -- pull frame especially important for first cycle
    dac_clk <= '0';
    dec_clk <= '0';
    dac_clk_b <= '0';

    wea <= '0';
    ena <= '0';
    block_length_mod8 := block_length_mod8 + 1;    -- assume mod8 counting

    if (block_length_count = block_length) then
      block_length_count := 1;
      addressb := (others => '0');

      if (delay = 2) then
        controller_state <= WRITE_PREPARE;
        delay := 0;
        addressa := (others => '0');
        block_length := "0000" & block_length(10 downto 3);
      else
        controller_state <= DAC_CLK_L1;
        delay := delay + 1;
      end if;

    else
      controller_state <= DAC_CLK_L1;
      block_length_count := block_length_count + 1;
      addressb := addressb + 1;
    end if;

-------------------------------------------------------------------------------
-- USB write sequence DO NOT MODIFY ---------------------------------------------
  when WRITE_PREPARE =>
    rdb <= '1'; -- disable USB read
    wr <= '0';  -- disable USB write
    led <= '0'; -- turn LED on
    rx_txb <= '1';

  -- added this part to turn off RAM write
    ena <= '1';
    wea <= '0';
  ---------------------------------

    if (delay_wr = 2) then
      delay_wr := 0;
      controller_state <= TOGGLE_WR_ON;
    elsif (txeb = '0') then
      delay_wr := delay_wr + 1;
      controller_state <= WRITE_PREPARE;
    else
      controller_state <= WRITE_PREPARE;
    end if;
```

```
    when TOGGLE_WR_ON =>
      rdb <= '1'; -- disable USB read
      wr <= '1';  -- TOGGLE WR ON
      led <= '0';
      rx_txb <= '0';

      --in_data <= temp;

      if (delay_wr = 2) then
        delay_wr := 0;
        controller_state <= TOGGLE_WR_OFF;
      else
        delay_wr := delay_wr + 1;
        controller_state <= TOGGLE_WR_ON;
      end if;

    when TOGGLE_WR_OFF =>
      rdb <= '1'; -- disable USB read
      wr <= '0';  -- TOGGLE WR OFF
      led <= '0';
      rx_txb <= '0';

      if (delay_wr = 1) then
        delay_wr := 0;
        controller_state <= WRITE_INCREMENT;
      elsif (txeb = '1') then
        delay_wr := delay_wr + 1;
        controller_state <= TOGGLE_WR_OFF;
      else
        controller_state <= TOGGLE_WR_OFF;
      end if;
-- USB write sequence DO NOT MODIFY ----------------------------------------
-------------------------------------------------------------------------------

      when WRITE_INCREMENT =>
      rdb <= '1';
      wr <= '0';
      led <= '0';
      rx_txb <= '1';

      if (block_length_count = block_length) then -- block length here is in bytes
        controller_state <= READ_PREPARE;
        addressa := (others => '1');
        block_length := (others => '0');
        block_length_count := 1;
        block_length_mod8 := (others => '1');
        muxcon <= '1';
      else
        controller_state <= WRITE_PREPARE;
        addressa := addressa + 1;
        block_length_count := block_length_count + 1;
      end if;

    when others => NULL;

    end case;
  end if; -- reset
  end if; -- clk'event
end process state_machine;

end RTL;
```