

# Analog Subthreshold CMOS Probability Gates

Matthew Aldrich

Yale University Department of Electrical Engineering

## Abstract

Analog CMOS circuits operating in the subthreshold region can represent soft logic gates. These soft logic gates compute discrete probabilities using current. This paper discusses the fundamental aspects of these probability networks, presents circuits that describe these functions, and finally gives performance estimates of the basic blocks used in an analog decoder.

## 1 Introduction

“The nervous system of even a very simple animal contains computing paradigms that are orders of magnitude more effective than are those found in systems made by humans...I have become confident that the powerful organizing principles found in the nervous system can be realized in our most commonly available technology - silicon integrated circuits,”

Carver Mead

Analog VLSI and Neural Systems (1989)

This paper discusses the ability of analog CMOS transistors to mimic a common biological function, the ability to make a decision. Often times these decisions are influenced by other sources of information. How much or how little these sources effect our decision can be represented in terms of probability. This human like quality to weigh a decision based all aspects of a situation is the basis for this discussion. What follows is a paper that bridges the gap between human cognition and silicon circuit.

This paper draws upon the ideas of analog decoding pioneers Hans Loeliger and Felix Lustenberger and makes use of Carver Mead’s previous work in analog VLSI.

## 2 Probability Propagation Networks

This section provides an introduction to the theory that serves as the motivation for the circuits presented in this paper. The theory is centered around one simple

principle: the way a group of functions influences the behavior of a single function and in turn, how that single function's response can be observed. It will be shown that one way to monitor a variable's reaction to a global influence is to measure how responsive or unresponsive the function is to that influence - that is discuss its state by using probabilities.

Only the most important concepts necessary for device motivation are included. For a more formal treatment on this subject, consult [1],[2].

## 2.1 Factor Graphs

A factor graph is a bipartite graph that expresses how a "global" function of many variables factors into a product of "local" functions. A factor graph visually depicts the variables in a system and the functions that operate on these variables. Variables are represented as a circle and local functions as a small box. Single variables or groups of variables are edge connected to these local functions. A variable, or variables, that is/are connected to local function implies that the local function performs some arbitrary computation on that/those variable(s). The output of the local function can then be passed along the graph. Decision making problems, such as decoding, can be represented with these graphs.

Factor graphs can also quickly and easily determine the effect of a group of variables on a single variable. The sum product algorithm gives us an orderly way to compute the influence on that particular variable. Computing this influence is known a *function summary* or *marginal*. A function summary is a mathematical way of stating a group of variables' effect on a single variable.

You might reason that an easy way of computing this influence is by the calculation of all previous variables and local functions that lead up to this variable - this assumption is correct. It may seem obvious, but in computing the marginal, the effect of the node in question is not computed. To illustrate this point consider a simple example. The example makes use of figure ???. The circles represent variables and the black boxes represent local functions. We see that the global function can be written as follows:

$$g(x_1, x_2, x_3) = f_a(x_1)f_b(x_2)f_c(x_1, x_2, x_3), \tag{1}$$

where the function summary is given by:

$$g_3(x_3) = \sum_{x_1} \sum_{x_2} g(x_1, x_2, x_3), \tag{2}$$

and in terms of its local functions:

$$g_3(x_3) = \sum_{x_1} f_a(x_1) \sum_{x_2} f_b(x_2). \tag{3}$$

In general, any marginal can be written by computing

$$g_i(x_i) = \sum_{\sim\{x_i\}} g(x_1, \dots, x_n). \quad (4)$$

Equation 4 states mathematically what we qualitatively reasoned through, in order to observe the effect on a single variable the influence of the "environment" on that variable is calculated. The  $\sim\{x_i\}$  in (4) reminds us that we are summing all information connected to  $x_i$ , but not  $x_i$ .

If this factor graph represented a decision, then the observed variable  $x_3$  would represent the effects of input  $x_1$  and  $x_2$ . In fact, what the graph represents is arbitrary, however a very interesting application results when the variables are representatives of probability. Consider for a moment if variables  $x_1$  and  $x_2$  represent how certain we are of a given process, then  $x_3$  represents how strongly we believe in this process, that is: in the product of  $x_1$  and  $x_2$ .

## 2.2 Probability Calculus Using the Sum Product Algorithm

The previous section provided an introduction to factor graphs and using the sum product algorithm to compute local function multiplies and marginals. In [1] it is heavily discussed how messages passed from node to node can represent probabilities or probability density functions. In this analysis, this is assumed to be truth. Let us now introduce a module that allows for probability propagation following the rules outlined in the previous section. Figure ?? represents a general configuration. Define  $p_x$  and  $p_y$  to be input probabilities and let  $p_z$  be the output probability defined as:

$$p_z(z) = \gamma \sum_X \sum_Y p_x(x)p_y(y)f(x, y, z). \quad (5)$$

In equation 5  $\gamma$  represents a scaling factor to ensure that  $p_z = 1$ . Here the local function  $f(x, y, z)$ , chosen by the user and is the subject of the next section.

## 3 Probability Modules as Boolean Logic Gates

In the previous section, the concept of factor graphs and the sum product algorithm was introduced. A method of applying the sum product rule to groups of probabilities was introduced. In this section we explain the local function "black box" of figure ?? in greater detail. By defining when a function is valid or not, we can control the operations that the function performs. This operation can be defined in a way such that these functions mimic the operation of classic logic gates. However, because we are operating with probabilities, the gates are "soft." In this case, only an indication

of how the data approximates a given logic function is yielded. We begin with a discussion of trellis diagrams and their usefulness.

### 3.1 Trellis Diagrams

Trellis diagrams are a visual way to represent all possible states of a function. For example if we define  $z$  to be a function of  $x$  and  $y$ . Define  $z(x, y) = 1$  when  $x = 1$  and  $y = 0$  and when  $x = 0$  and  $y = 1$ . In addition, define  $z(x, y) = 0$  when  $x = 0$  and  $y = 0$  and when  $x = 1$  and  $y = 1$ . In boolean logic, this function  $z(x, y)$  is known as an exclusive OR. Its trellis diagram is the middle trellis given in figure ??.

From the figure, all possible inputs  $x$  and  $y$  are given. This diagram represents all the possible states of  $z$  given all possible  $x$  and  $y$  inputs. Here it is relevant to note that any possible sequence of integers can be represented as a trellis diagram. For the purpose of this section we will only consider one stage examples.

### 3.2 Soft Logic Gates Using the Sum Product Algorithm

Recall from equation 5 that  $f(x, y, z)$  specified how inputs were processed. In addition, the equation states that we can represent any probability  $p_z$  with any combination of adds and multiplies. The local function  $f(x, y, z)$  can limit what types of adds and multiplies are permitted, the trellis is defined for  $f(x, y, z) = 1$  only. Several configurations are discussed below. The outputs are defined in terms of  $p_z(0)$  and  $p_z(1)$  these outputs represent the marginal (summary function).

#### 3.2.1 Equal Gate

Define the local function  $f(x, y, z) = 1$  iff  $x = y = z$ . The resulting trellis is shown in figure ?. Furthermore, by applying this function to equation 5 for both  $p_z(0)$  and  $p_z(1)$  we obtain the following expression:

$$\begin{bmatrix} p_z(0) \\ p_z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_x(0)p_y(0) \\ p_x(1)p_y(1) \end{bmatrix} \quad (6)$$

The equal gate is simply a local function node that multiplies two probabilities together.

#### 3.2.2 Soft-XOR Gate

An XOR gate defines  $f(x, y, z) = 1$  iff  $z = x \oplus y$ . The trellis is shown in figure ?. From inspection of the trellis, one may infer that a function summary of this trellis must include some form of addition. As given in [1] the resulting probability formulation is given by:

$$\begin{bmatrix} p_z(0) \\ p_z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_x(0)p_y(0) + p_x(1)p_y(1) \\ p_x(0)p_y(1) + p_x(1)p_y(0) \end{bmatrix} \quad (7)$$

Here, the soft XOR both multiplies and adds probability distributions.

### 3.2.3 Backwards Reasoning AND Gate

The backwards reasoning AND takes advantage of the bidirectionality of the factor graph nodes however it is unlikely to occur in decoding [1]. Define  $f(x, y, z) = 1$  iff  $x = y + z$ . The backwards reasoning applied to the trellis diagram is given in figure 1. By substituting  $f$  into 5 we find:

$$\begin{bmatrix} p_z(0) \\ p_z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_x(0)p_y(0) + p_x(0)p_y(1) \\ p_x(0)p_y(0) + p_x(1)p_y(1) \end{bmatrix} \quad (8)$$

### 3.3 Remarks

By using the probability propagation network and defining specific local functions we have successfully created a library of soft boolean gates. These gates allow the addition and multiplication of probabilities using the sum product algorithm.

## 4 Implementation in Analog CMOS Circuits

This section is concerned with the hardware implementation of the soft logic gates discussed in the previous section. For local functions  $f$  it was shown that the sum product algorithm allowed probability distributions which behaved similar to boolean logic gates. This section documents the building blocks and circuits that accomplish these probabilistic adds and multiplies.

Is it not obvious how these circuits will operate. Will they be voltage or current mode implementations? How can adds and multiplies be constructed? Adds are simple in analog circuitry. Essentially they are available for free. Recall that Kirchoff's Current Law states the sum of all currents into a node is zero. By connecting one wire to another (shorting a wire), the addition of current is realized.

Multiplication is not straightforward in analog CMOS. In 1975 Gilbert's current multiplier used both the exponential and logarithmic characteristics of BJTs to multiply current [3]. Performing multiplication using analog CMOS requires the manipulation of the operation range of the MOS transistor. This operating range is known as the subthreshold region, and it is synonymous with Carver Mead. Mead's work focused on exploiting the non linearities of MOS transistors operating in the subthreshold region. The goal was create circuits that used little power and behaved like biological functions. By operating in the subthreshold region, Mead found that

CMOS technology behaved like bipolar junction transistors, that is, the CMOS implementations modeled the characteristics of BJTs necessary have current multiplication [4].

The following sections discuss current multiplication in the subthreshold region and give operating parameters based upon PSPICE simulations in a MOSIS 1.5 $\mu$  process. Finally, circuits which implement the logic functions described earlier are given.

## 4.1 Subthreshold MOS Characteristics and Basic Circuits

### 4.1.1 Defining the Saturation Current in NMOS and PMOS

The classic derivation of subthreshold saturation current in NMOS and PMOS transistors is well documented and accepted. For a complete treatment of this subject beginning with the MOS diffusion equation consult [4],[5]. In the subthreshold region, the transistor operates slightly below the voltage threshold,  $V_T$ . The charge on the gate of the MOS device is almost balanced by the negatively-charged depletion region underneath the gate.

In the subthreshold region of a n channel MOS device (NMOS), we can write the saturation current  $I_s$  as:

$$I = I_0 e^{V_g/U_T} (e^{-V_s/U_T} - e^{-V_g/U_T}). \quad (9)$$

Furthermore as we increase the drain source voltage  $V_{ds}$  beyond  $4U_T$ , we move from the linear region of operation to the saturation region of operation and equation 9 no longer depends on  $V_{ds}$ . This occurs typically around  $V_{ds} = 100\text{mV}$  [MEAD].

$I_s$  for an NMOS transistor in subthreshold operation in the saturation region may then be written as:

$$I = I_0 e^{(V_g - V_s)/U_T}, \quad (10)$$

where  $I_0$  is the pre-exponential.  $I_s$  for a saturated PMOS transistor in subthreshold operation may be written:

$$I = I_0 e^{(-V_g + V_s)/U_T}, \quad (11)$$

where  $U_T$  is the thermal voltage,  $kT/q$ . Figure 1 shows the results of increasing  $V_{gs}$  and the resulting  $I_{sat}$  in a NMOS transistor. As expected, the current response is exponential function of  $V_{gs}$ . The deviation from the exponential behavior in figure 1 occurs when the flow of  $I_{sat}$  becomes restricted.

Figure 1 is important, we have just defined a critical operating parameter in our probability circuits. We define:

$$P_1(z = 1) = I_{sat} = 10nA \text{ and } P_1 < 10nA \text{ when } P_1(z \neq 1), \quad (12)$$

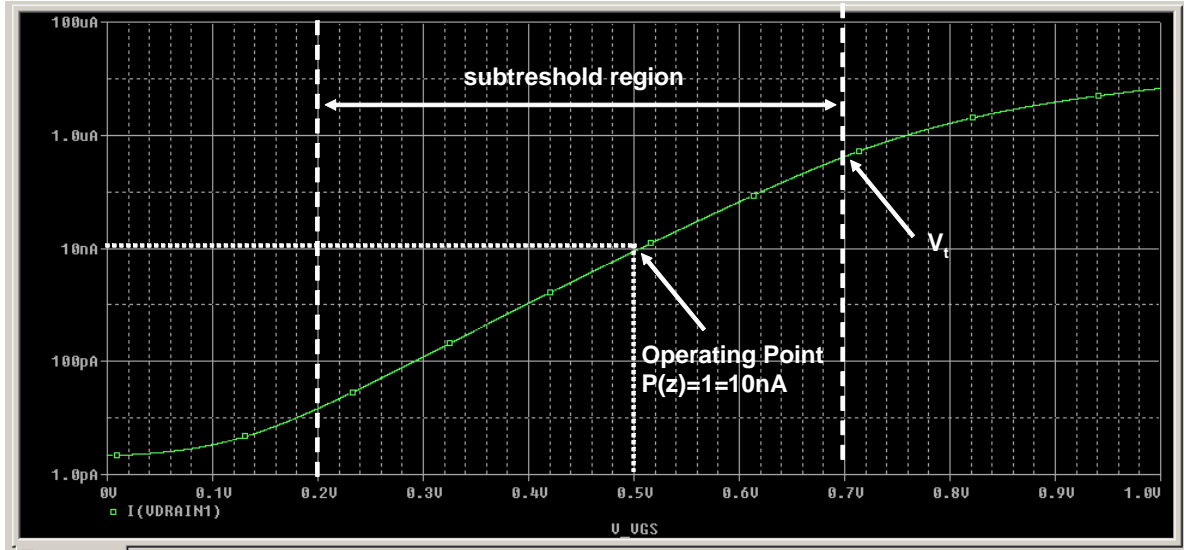


Figure 1: Saturation current of a NMOS transistor as a function of gate voltage. The threshold is approximately .7V. Therefore a current level less than this threshold must be used to ensure subthreshold operation. This simulation is for a minimum sized NMOS transistor  $W/L = 3.2\mu/1.6\mu$  in a MOSIS  $1.5\mu$  process.

and

$$P_0(z = 1) = I_{sat} = 10nA \text{ and } P_0 < 10nA \text{ when } P_0(z \neq 1), \quad (13)$$

under the following constraint that:

$$P_1 + P_0 = 1, \quad (14)$$

so that the currents are complimentary.

## 4.2 A Subthreshold CMOS Multiplier

In order to mimic the functions described by the soft logic gates, a circuit must be created to allow for the multiplication. The multiplier circuit is nothing more than a collection of several current mirrors, and a differential pair - a simple, but extremely elegant solution! We begin by discussing several critical operation parameters of a diode connected transistor, show how it creates a current mirror, define the transfer characteristics of a differential pair, and then show how these circuits create a current multiplier.

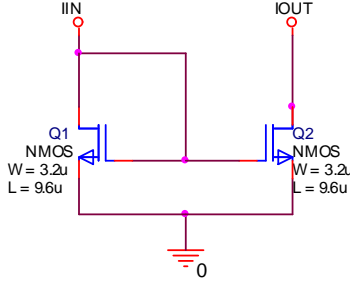


Figure 2: This current mirror shares the same source potentials and sizes. The long channel lengths help diminish the Early effect.

#### 4.2.1 Diode Connected MOS Transistors and CMOS Current Mirrors

In an NMOS device, the drain current  $I_{sat}$  is an exponential function of  $V_s$  and  $V_d$  (eq. 10). If this equation is solved in terms of  $V_g$  the result is a logarithmic current to voltage converter where:

$$V_g = V_s + U_T \log \left( \frac{I}{I_0} \right). \quad (15)$$

In a current mirror, a diode connected transistor shares its gate node with another transistor of the same type. If the current sources are fixed and the devices are in saturation, they act as current sources. Similarly if both devices are the same geometries and share the same source potential then they source the same current. This is explained in figure 2: Current  $I_{IN}$  sets gate voltage  $V_g$  for both  $Q_1$  and  $Q_2$ . Consequently  $V_g$  now sets current  $I_{OUT}$ . In this example,  $V_{s,Q1} = V_{s,Q2}$  and  $V_{d,Q1} = V_{g,Q1} = V_{g,Q2}$ . Using equation 15 and plugging  $V_g$  into equation 10, it is easily verified that  $I_{OUT} = I_{IN}$  because of the logarithmic-exponential characteristics.

As pointed out in [other analog], current mirrors allow the current output to be scaled by either using different source voltage potentials in which:

$$I_{OUT} = e^{(V_{s1}-V_{s2})/U_T} I_{IN}, \quad (16)$$

however we can also scale the current output by varying the ratio of transistor widths given as:

$$I_{OUT} = \frac{W_2 L_2}{W_1 L_1} I_{IN}. \quad (17)$$

The implementation of efficient and noise tolerant current scaling is important when a decoding application is considered. Due to the multiplicative nature of the sum product algorithm and the fact that the multiplied probabilities will not always be unity, the current level will tend to 0 if it is not scaled from time to time.

### 4.2.2 The Differential Pair

As the final step towards building the Analog CMOS multiplier, the differential pair is introduced. The derivation of current outputs is well documented in [4], therefore just the outputs will be presented and its usefulness explained. Its circuit schematic is presented in figure 3. This circuit is quite powerful; by careful consideration of what voltage is applied at  $V_1$  and  $V_2$ , the current at the sources of the differential pair can be steered to appear at either output. This property is demonstrated in figure 4. This property can be summarized as follows:

$$I_1 = I_b \frac{e^{V_1}}{e^{V_1} + e^{V_2}} \quad \text{and} \quad I_2 = I_b \frac{e^{V_2}}{e^{V_1} + e^{V_2}} \quad (18)$$

### 4.2.3 A CMOS Multiplier

There is not much work left to create this multiplier. In fact, all the necessary components have been discussed. Similar to the reasoning in [6], all that remains is to add a diode connected transistor to each of the inputs in figure 3. Recall that in equation 18 the current  $I_b$  was steered by the dominating voltage. By adding diode connected transistors to inputs  $V_1$  and  $V_2$ , creating current mirrors at the inputs of the differential pair,  $I_b$  is now multiplied by currents instead of voltages. Formally that is:

$$I_1 = I_b \frac{I_{in1}}{I_{in1} + I_{in2}} \quad \text{and} \quad I_2 = I_b \frac{I_{in2}}{I_{in1} + I_{in2}}. \quad (19)$$

Equations 12-14 allow us to set  $I_{in1} + I_{in2} = 1$ . Recall that these currents represent the probability of an arbitrary state. One current must be the compliment of the other. This circuit is illustrated in figure 5.

## 4.3 Implementing Soft Logic Gates

Using the soft logic gates presented earlier and the circuit theory provided by the previous section, fully functional circuits that implement the sum product algorithm can be implemented. The circuits mimic the trellis diagrams, that is they take inputs  $x$  and  $y$  and produce output  $z$  when  $f(x, y, z) = 1$ . This section presents the schematics for all three logic gates presented earlier. The PMOS current mirrors tied to VDD source current rather than sink current. These mirrors allow multiple blocks to be connected. Following the introduction of these circuits, some insight on the issues of supply voltage, proper biasing, and sizing of the transistors is given.

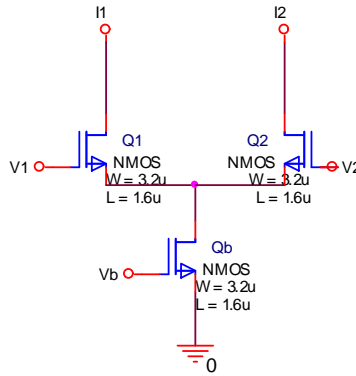


Figure 3: Schematic of the differential pair. The source of transistor  $Q_b$  is by the voltage applied at the gate of  $Q_b$ . The amount of  $I_b$  seen at  $I_1$  and  $I_2$  is proportional to the difference in  $V_1$  and  $V_2$ .

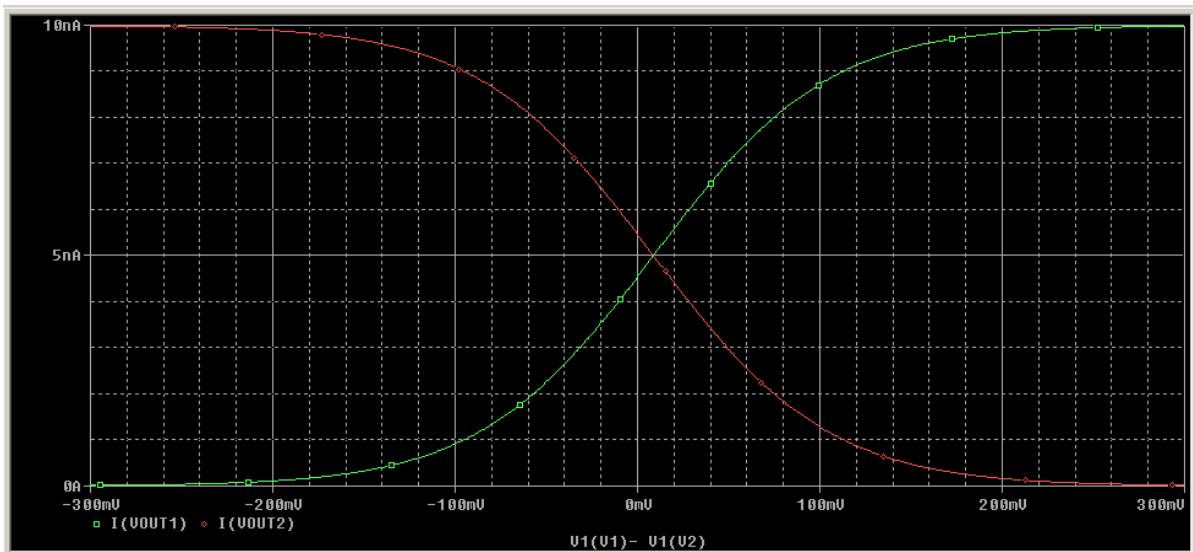


Figure 4: Output currents of the differential pair as a function of differential input voltage. When  $V_2 > V_1$  current propagates from  $I_2$  and when  $V_1 > V_2$  current propagates from  $I_1$ . The lines do not cross at zero due to a simulated transistor mismatch

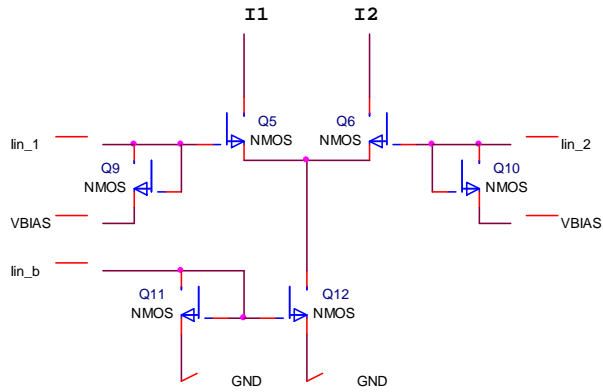


Figure 5: Fully developed current multiplier. Operation is similar to the differential pair discussed earlier.

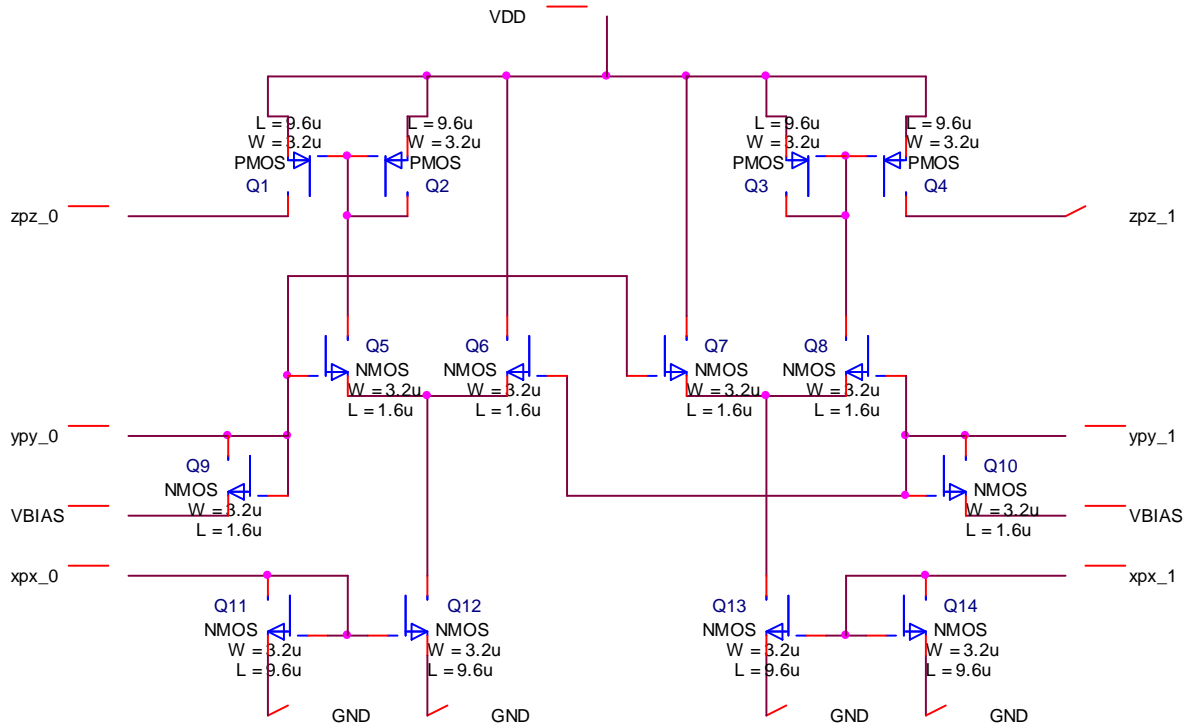


Figure 6: A CMOS soft equal gate implementing the logic in eq. 6

### 4.3.1 The Equal Gate

The equal gate is the simplest of the functions presented earlier. Refer to figure ?? and equation 6 for the trellis and function summary. The circuit schematic is presented in figure 6. Notice that transistors  $Q_6$  and  $Q_7$  are not shorted to  $Q_5$  and  $Q_8$  this is due to the fact that the equal gate does not perform addition. Therefore, a quick way to block current from  $Q_6$  and  $Q_7$  is to tie the sources to  $V_{DD}$ . The operation is straightforward: current at  $x_{px\_0}$  and  $x_{px\_1}$  bias  $V_g$  of  $Q_{11}$  and  $Q_{12}$  and  $Q_{13}$  and  $Q_{14}$  respectively.  $Q_{12}$  and  $Q_{13}$  mirror the current. Transistors  $Q_9$  and  $Q_{10}$  pass their current to  $Q_5$  and  $Q_8$ . In turn, the current sourced by  $Q_1$  is the product of  $x_{px\_0}$  and  $y_{py\_0}$  and the current sourced by  $Q_4$  is a product of  $x_{px\_1}$  and  $y_{py\_1}$ . We observe that the multiple of  $Q_{12}$  and  $Q_6$  and  $Q_{13}$  and  $Q_7$  is impossible - thus this circuit is in accordance with the trellis.

### 4.3.2 The Soft XOR Gate

Refer to figure ?? and equation 7 for the trellis diagram and output characteristics. Figure 7 is the circuit representation. From the trellis diagram, we expect this circuit perform two adds. As expected, the source currents of  $Q_5$  and  $Q_7$  express  $p_z(0) = p_x(0)p_y(0) + p_x(1)p_y(1)$  and  $Q_6$  and  $Q_8$  express  $p_z(1) = p_x(1)p_y(0) + p_x(0)p_y(1)$ . With the exception of the addition, this circuit behaves exactly like the equal gate.

### 4.3.3 Backwards Reasoning AND Gate

Refer to ?? and equation 8 for the trellis diagram and output characteristics. Figure 7 is the circuit representation. This circuit is different from the others because  $Q_{15}$  replicates the multiple of  $x_{px\_0}$  and  $y_{py\_1}$ . It does this because if there is a high probability of  $x$  being 0, then probability of of input  $y$  must yield a 0 for  $z$ . When reasoning through this gate, it is helpful to recall that the AND boolean operation yields only a 1 when both inputs are 1.

## 5 Performance of Equal and XOR Gates

The equal gate and XOR are the essential building blocks in any analog decoder where the state of transmitted information can be represented by a trellis [1]. Consider a strict binary equal gate and XOR gate, then any sequence of code (1101, 0110, 0011...etc.) can be represented by stringing together these gates' trellis diagrams. This is an extremely powerful conclusion. Why?

Our probability calculation on an arbitrary trellis diagram (i.e. our soft equal and soft XOR) maps to this exact trellis diagram of all potential codes. Thus, if we can ensure that we process this trellis both in the forwards and backwards direction, we can obtain a probability calculation at every single node! In the application of decoding, there might be a bit or two on the trellis that we wish to observe, thus

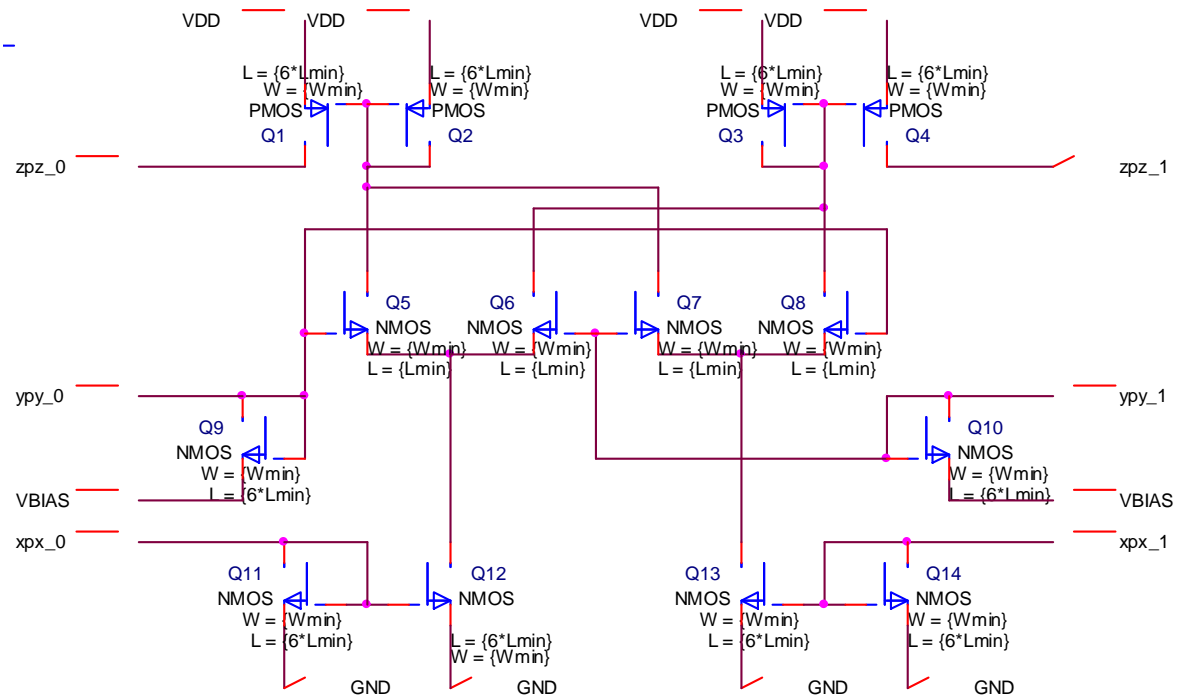


Figure 7: Transistor schematic implementing the soft XOR function as defined in eq. 7.



calculating the probability of the bit's state is as simply as parsing the probability gates in the forward direction and in the backwards direction and observing the output at that node.

Therefore it is of great importance to obtain basic performance estimates of these probability modules respond when given a stimulus. Power consumption, output delay, maximum operating frequency, and transistor sizing are considered in the following analysis of the soft equal and XOR gates.

## 5.1 Design Considerations

Several questions come to mind when presented with the above schematics. In specifying parameters such as voltage and transistor sizes it is important to understand the context in which the circuit will be performing. As mentioned previously, these circuits are suited well for decoding applications. Why? First, they can indicate, through very little computational cost, the probability of whether a bit is a one or zero. Secondly, analog circuits biggest criticism, the lack of accuracy, is not an issue in this application. This statement deserves careful consideration.

Suppose that after passing through several gates we operate at 50% of maximum current value, then we leave ourselves vulnerable to noise contamination and we lose the meaning of what we wanted to compute. Now suppose, that after several probability computations we scale our probability current in a manner presented in equations 16 and 17. Because our circuits give *indications* of a 1 or 0, worrying about whether or not we can accurately bring our current back to its maximum operating value is irrelevant. Instead, we can preserve our computations against the circuit's inherent noise by periodically "raising" the current level; this is analogous to the natural error correction of digital logic, where an input can be pulled high or set low after each stage of logic.

### 5.1.1 DC Operating Voltage

All of the simulation results presented in this section used a supply voltage  $V_{DD}$  of 1.5V. The supply voltage  $V_{DD}$  is a trade-off between power consumption and signal noise. For example, lowering  $V_{DD}$  results in lower current draw from the supply which results in more power savings, however this is at the expense of degraded transistor operation, the output currents may not rise to the maximum operating level. However, because some error is tolerated, a large supply voltage is also not necessary. Instead the voltage  $V_{DD}$  should be chosen to ensure maximum power savings while tolerating some percentage of error. In these circuits the tolerated error,  $\epsilon$ , after one gate, was set so that  $\epsilon \leq 10\% (P_1(z = 1) + P_0(z = 1))$ .

In addition to  $V_{DD}$ , the circuit diagrams in figures 6-8 use a bias voltage called  $V_{BIAS}$  to assist in the probability calculation. In all the simulations  $V_{BIAS} = 25\%(V_{DD})$ . Recall that equation 16 states that the output of a current mirror  $I_{OUT}$  can be scaled according to the difference in  $V_s$ . Voltage  $V_{BIAS}$  attempts to reduce

this scaling effect and allow this second stage of current mirrors to have a more ideal operation, that is  $I_{OUT} = I_{IN}$ . Because we can never be sure of the voltage  $V_s$  at the transistors in the differential pair, a bias voltage is selected to *reduce* the scaling of  $I_{OUT}$ .

### 5.1.2 Transistor Sizing

CMOS implementations have the benefit of occupying less die area than their BJT counterparts in layout. Therefore, the circuit must operate correctly, but should occupy an area smaller than a BJT implementation. With this constraint in mind, sizing in these circuits employs where possible, a minimum  $W/L$  ratio. In digital design, the sizing of the transistors (in particular, the transistor's width) can allow the designer to favor different types of input conditions. The designer does this to decrease the delay of that particular gate. In the case of these circuits, widening the transistors does not decrease the rise time. In fact, widening the transistors will simply increase power consumption, current levels will need to be raised, and a larger supply voltage is needed to calculate the circuit's outputs. Therefore digital techniques to improve gate delay do not apply. Our only other recourse is create a plot similar to figure 1 and set the current at level just below  $V_t$ , so that we operate at the maximum current density in the subthreshold region.

Transistor length is also important. Short channel effects, such as the Early effect can modulate our transistor's current output. By increasing the transistor length we can protect against any short channel effects. As a rule of thumb in any of the circuits presented in this paper, current mirror transistors lengths' were given by  $L_{current\_mirror} = 6L_{min}$ . This sizing rule allowed for proper operation.

## 5.2 Transient Analysis of Equal and XOR

These analog gates make use of the inherent nonlinearities of the subthreshold region. In classical transistor analysis, the small signal model is used to obtain the circuit's transfer function. By finding the transfer function's response to a step function, the system's stability can be analyzed. In addition, the delay of the transfer function can be found. This type of analysis requires a linear system, which we clearly do not have. Therefore transient analysis of these two gates is performed in Orcad's PSPICE simulation tool. Performance metrics such as power consumption, delay, and frequency can be quickly observed.

The equal and XOR gates in these test cases are two input examples. In a decoding application, this corresponds to a 2 bit:1bit operation. In addition, the test cases were simplified. We assume in all test cases that if  $P(0) = I_{sat}$  or  $P(1) = I_{sat}$ , where the compliment is 0 or off. The test case is not taking advantage of the probabilistic nature of the gates, however it illustrates our claims about the equal and XOR gate and yields initial performance estimates. To ensure the output nodes were properly loaded, a copy of the gate under test was connected to the outputs.

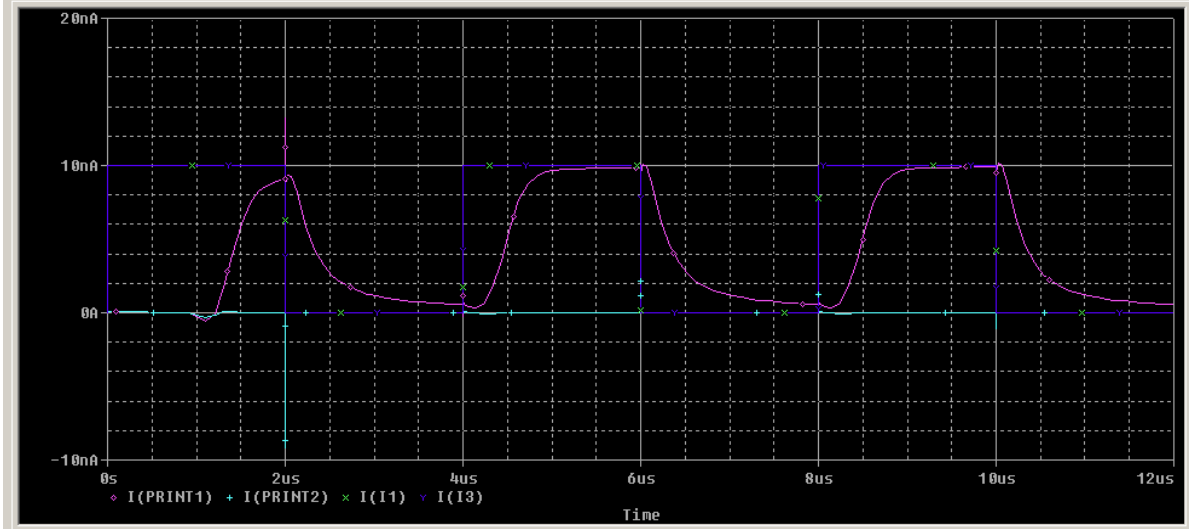


Figure 9: Output response for  $p_z(0) = 1$  from soft equal gate described by eq. 7

### 5.2.1 Equal Gate Output Response

The equal gate was tested for both cases defined in equation 6. The results can be summarized in the following table. The first column  $P(z)$ , indicates where the test function indicates a 1 or 0 at the gates output. *Power* lists the device's power consumption in Watts. The third and fourth columns list the device's rise time,  $\tau_r$  and fall time  $\tau_f$ , measure in seconds. *Frequency* lists the maximum achievable clocking frequency of the circuit, in Hz, and column five is the *power - delay product*, a useful metric in evaluating these gates. The power-delay product is a measure of the gate performance, lower power-delay product means that power is better "translated" into speed of operation.

$P(z)$	<i>Power</i> (pW)	$\tau_r$ (ns)	$\tau_f$ (ns)	<i>Frequency</i> (MHz)	<i>Power * Delay</i> ( $10^{-18}$ )(J)
0	7	750	250	1	5.25
1	7	750	250	1	5.25

The following PSPICE plots illustrate the circuits response to a current step function. Figure 9 is the circuits response to input conditions  $p_x(0) = 1$  and  $p_y(0) = 1$ . Therefore the output is  $p_z(0) = 1$ . The circuit's response to input conditions  $p_x(1) = 1$  and  $p_y(1) = 1$  can be found in Appendix A. This is in agreement with equation 7. The delayed output from  $0\mu s - 2\mu s$  is due to capacitance charging of the circuit's transistors, the long channels of the current mirrors contribute heavily to this start up cost.

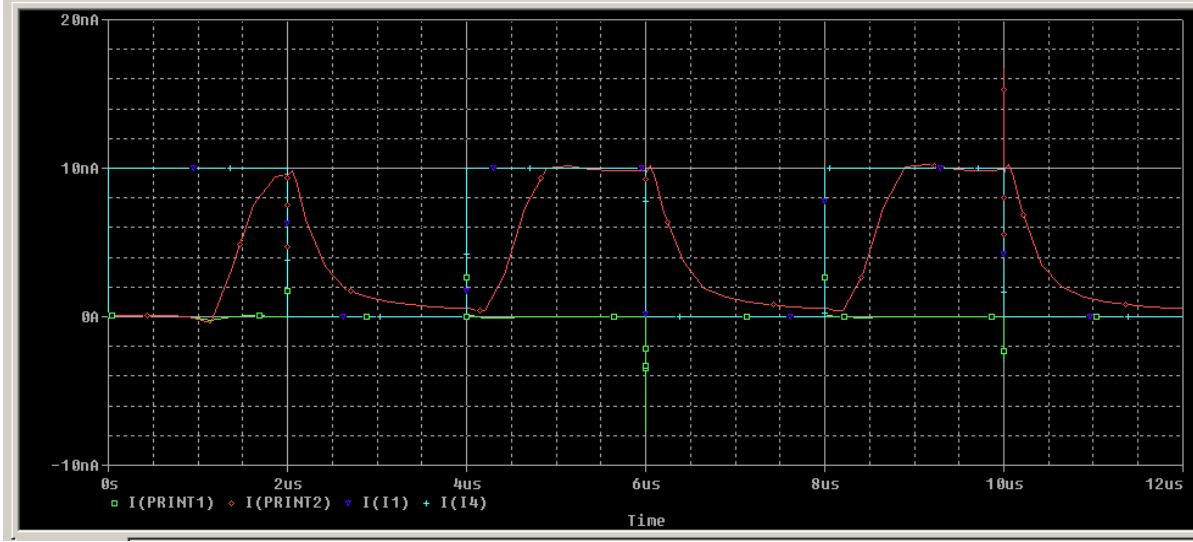


Figure 10: Soft XOR response for test case  $p_x(x_0 = 1) \oplus p_y(y_1 = 1) = p_z(z_1 = 1)$

### 5.2.2 XOR Gate Output Response

The table expression the XOR's performance is given below. A sample response of  $p_z(1) = 1$  is given in figure 10. The circuits outputs are in accordance with equation 8. The other test case for  $p_z(0) = 1$  is included in appendix A. The following table describes the XOR's characteristics similar to manner in the equal gate analysis.

$P(z)$	Power (nW)	$\tau_r$ (ns)	$\tau_f$ (ns)	Frequency (MHz)	Power * Delay( $10^{-14}$ )(J)
0	30	750	250	1	2.25
1	30	750	250	1	2.25

## 5.3 Analog vs. Digital Soft Logic Gates

This section sheds light on the performance of these gates versus their digital counterparts. In order to compare the two, this section makes several assumptions about a digital add and a digital multiply.

Let a 1 bit add in a modern process have the following characteristics:

- Output delay.,  $\tau_d < 1ns$
- result in 1 clock cycle
- power  $> 1\mu W/bit$

Let a 1 bit multiply in a modern process have the following characteristics:

- Output delay.,  $\tau_d < 3ns$
- result in 3 clock cycles

- power > 1 $\mu$ W/bit

The above performance estimates, are grossly simplified, but they illustrate a strong case for analog implementations of these functions. Recall that a soft equal gate is analogous to a one bit multiply. In terms of multiplication an equal gate requires 2 *1-bit multiplications*. An XOR gate achieves 4 *bitwise multiplies* and 2 *bitwise adds*. Given the information from the tables above, we can now make a case for these analog gates.

### 5.3.1 Equal Gate Analysis

Apply the metrics above to obtain a first order comparison of the gate's performance. For simplicity assume the processor has one floating point multiplier available. Keep in mind a small PDP factor indicates better performance than a large one.

- Analog:@1MHz, we achieve 2 multiplies
- Digital: Multiply 2 bits <3ns/clock, 3 clock cycles: 9ns/multiply = 18ns delay
- Digital 18ns per calculation
- Analog 1 $\mu$ s per calculation
- Digital versus analog operating frequency: 55Mhz vs. 1Mhz
- Power-delay product ( $10^{-18}$ ): Digital: 18,000 vs Analog: 5.25

While the digital implementation can run at a higher clocking frequency than its analog counterpart, the power-delay metric allows the analog results to shine. Assuming a digital multiply power consumption is on the order of 1 $\mu$ W/bit may be too high, however is safe to assume it will never be on the order of picowatts.

### 5.3.2 XOR Gate Analysis

The procedure is similar to the equal gate. In this case, assume that an FP add and multiply can not be pipelined.

- Analog:@1MHz, we achieve 4 multiplies and 2 adds
- Digital: Multiply 2 bits (four times) <3ns/clock, 3 clock cycles: 12 clock cycles = 36ns for multiply delay
- Digital: Add 2 bit <1ns/clock, 1 clock cycle: 2 clock cycles = 4ns for addition delay
- Digital: 40ns per calculation
- Analog: 1 $\mu$ s per calculation
- Digital versus analog operating frequency: 25MHz vs. 1Mhz
- Power-delay product ( $10^{-13}$ ): Digital: 1.2 vs Analog: .225 (assuming 3 $\mu$ W digital power consumption).

Again the allotment of power for the digital process may be too generous, however it is clear that analog outperforms its digital counterpart in the PDP metric.

## 5.4 Performance Remarks

Having created general cases for digital implementations of these gates, it is easy to see that in the power-delay metric, the analog solution outperforms its digital counterpart. The PDP metric allows less emphasis to be placed on the clocking speed and more emphasis to be placed on the clocking ability given power restraints. With ALU clocking speeds today approaching close to 7GHz [7], one may question whether 1MHz clocking rate is competitive. In a modern process, for example  $.18\mu$  the minimum transistor sizes are significantly smaller than a  $1.5\mu$  counterpart, lowering the parasitics and improving performance.

## 6 Issues and Future Work

The key question is how to apply these circuits in decoding applications. Lustenberger does an excellent job in describing decoding examples of different complexity in [1]. We know that any codeword or sequence of data can be represented by a corresponding trellis diagram. Each individual trellis section corresponds to a function variable on a factor graph. The output characteristics of that trellis depend on the local function  $f(x, y, z)$  that the variable is passed into. Therefore, we can create a factor graph that is a visual depiction of this trellis diagram. Furthermore, we can map our equal and XOR probability modules to this trellis, and observe any given variable on that factor graph.

What happens if the trellis diagram does not have a specific entry point and exit point? In this case the factor graph has no clear entry and exit point and becomes cyclical. This model accurately describes any codes that are "tail-biting." Their origins are rooted in advanced information theory however Lustenberger's explanations are fairly clear and after this introductory paper, should be easier to grasp.

Like a trellis of arbitrary states, the possible paths of this project branch in every direction. In order to fully study the circuits presented in this paper, the next step would be to layout these schematics and compare the performance of the ideal circuits presented with those simulated with interconnect delay. In addition, tape out of these schematics for actual test may prove whether the MOSIS process allows for proper analog behavior. Having created a standard library with similar pitch and wiring paths, one, after studying some introductory information theory, a potential project is the creation and simulation of the block decoder presented in [1]. A really aggressive schedule would seek to fully understand turbo codes, a high performing channel encoding scheme, and move to creating a tool to automate schematic design and layout.

## 7 Conclusion

This paper presented an introduction to decision networks and a method for solving these networks using basic probability. It was shown that these probability calculations can be performed using subthreshold operating CMOS transistors. The large signal analysis for basic circuit operation was given and circuit schematics that implement these probability calculations were presented. Correct operating techniques were given as well as an important first look into the gates' response given input. In addition, the trade-offs in design were discussed and a high level comparison of these circuits versus their digital counterparts was given. With proof that these analog circuits exceeded their digital counterparts in several metrics, several interesting projects focusing on decoding applications were proposed.

## References

- [1] Lustenberger, Felix. "On the Design of Analog VLSI Iterative Decoders," Ph.D. Thesis, Nov. 2000
- [2] Kschischang, Frank R. et al. "Factor Graphs and the Sum-Product Algorithm," Oct. 19, 2000.
- [3] Gilbert, Barrie. "A precise four-quadrant multiplier with subnanosecond response," *IEEE Journal of Solid-State Circuits*, 3:365, 1968.
- [4] Mead, Carver. *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.
- [5] Liu, Shih-Chii et al. *Analog VLSI: Circuits and Principles*. MIT Press, 2002.
- [6] Luckenbill, Sam. "Building Bayesian Networks with Analog Subthreshold CMOS Circuits," Yale University, 2002.
- [7] D. Deleagnes et al., "LVS Technology for the Intel Pentium 4 Processor on 90nm Technology." *Intel Technology Journal*, vol. 8, no. 1, Feb. 2004, pp. 49-59.

## Appendix A - Additional Equal and XOR PSPICE Simulations

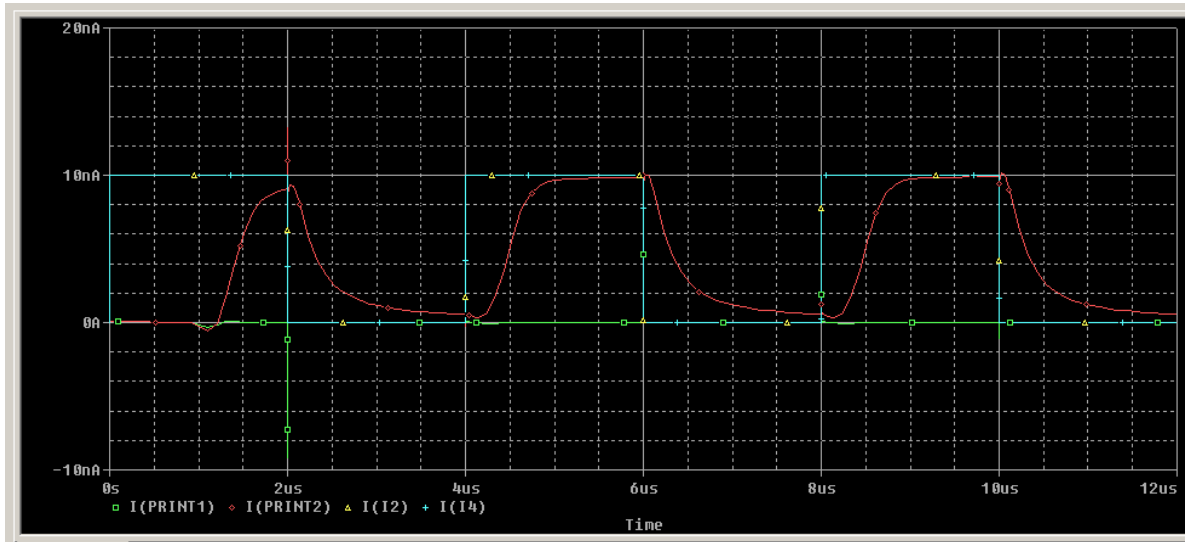
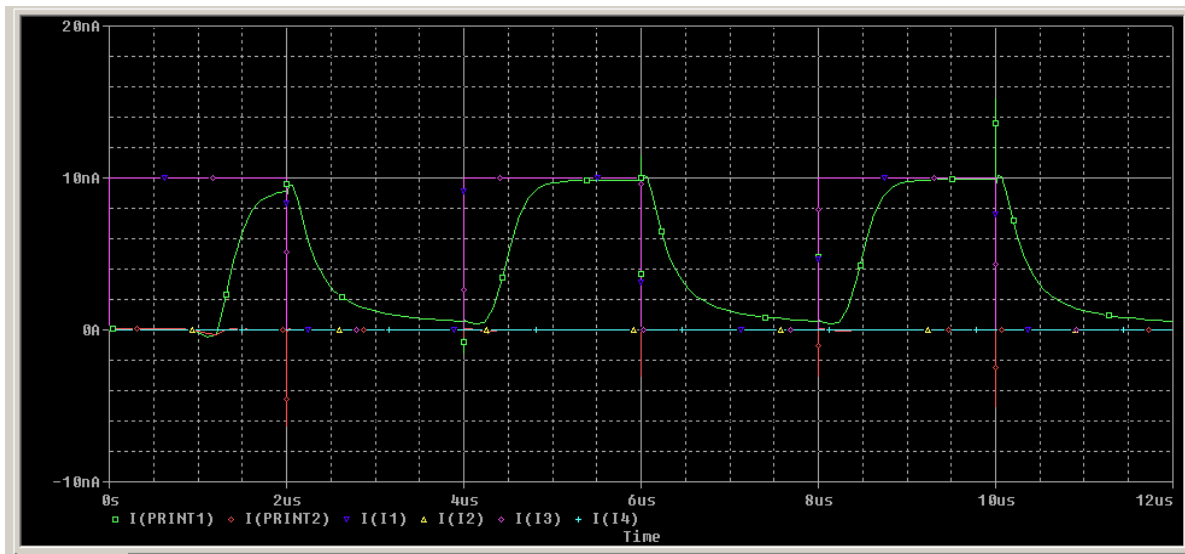


Figure 11: Soft equal transient response for  $p_x(x_1 = 1)p_y(y_1 = 1) = p_z(z_1 = 1)$ .



Soft XOR transient response for  $p_x(x_0 = 1) \oplus p_y(y_0 = 1) = p_z(z_0 = 1)$ .