Sign Detection and Comparison Networks with a Small Number of Transitions

Miloš D. Ercegovac† Tomás Lang‡

† Computer Science Department, University of California, Los Angeles, CA 90024

‡ Dept. of Electrical and Computer Engineering, University of California, Irvine, CA 92717

Abstract

We present an approach to reducing the average number of signal transitions (T_{av}) in the design of sign-detection and comparison of magnitudes. Our approach reduces T_{av} from $21n/8$ (n - operand precision in bits) to 4.5 in the case of iterative implementation, and from about n to roughly $k + n/2^{k-1}$ in the tree network implemented with k-bit modules. We also discuss comparison of small numbers. The approach is applicable to other arithmetic problems.

1 Introduction

We are initiating a research effort in the development of numerical computing systems which require a small amount of electrical power to operate. The total power dissipation in a chip is due to a variety of sources, including input/output circuits, buses, clock generation and distribution, on-chip memories, and computing structures. We concentrate on the computation structures and, in particular, on arithmetic structures. Consequently, the methods of power reduction we propose have to be complemented with others dealing with the rest of the system.

The computation structures can be represented at various levels of abstraction. In this study, we view the structures as gate networks and latches. In this domain, we explore the effectiveness of selecting several features, such as number representations, algorithms, and structures, to reduce the power consumption. These results are related to the circuit level of implementation by making suitable assumptions on the power dissipated by the gates and the latches. The methods proposed have to be combined with techniques applied to other levels, such as the use of low-power technologies and power-down strategies [3].

To evaluate the power dissipation of the proposed structures, it is necessary to utilize a model. We concentrate on a model suitable for CMOS technology because this is the prevalent technology today and it is predicted that it will continue to be so in the near future. For this case, the static power dissipation is negligible and the dynamic power dissipation is related to the number of signal transitions [3]. Consequently, in this research we concentrate on the reduction of the number of signal transitions and assume the proportionality between this measure and power dissipation.

CMOS implementations use two types of gates: static and dynamic, which have different characteristics in terms of the transitions produced [3]. The operation model is simpler for the case of dynamic gates because the gate outputs are reset at the beginning of each cycle and because there is at most one transition per output per cycle. On the other hand, static gates do not have these constraints, so the transitions depend on the previous state of the outputs and there can be multiple transitions per output per cycle. We will consider static gates here.

In this paper we describe the development of arithmetic modules which performs the operation with a small average number of transitions. The main technique used is to modify a conventional implementation to eliminate unnecessary transitions. We calculate the average number of transitions under the following assumptions:

1. The input vectors are applied simultaneously (all bits).
2. The transitions at the input of the circuit are not included.
3. The sequence of inputs are independent and obtained from a uniform distribution. We also consider a special case in which the operands are small integers (Sec. 3).
4. The gates are static CMOS.
5. All gates have the same delay.
6. We count the sum of high-to-low plus low-to-high transitions.
7. If two or more transitions arrive simultaneously at the inputs of a gate, at most one transition is pro-

1063-6889/95 $4.00 © 1995 IEEE

duced at the output. To determine whether a transition is produced we consider the output value before and after the occurrence of the simultaneous transitions. On the other hand, glitches can occur (and are accounted for) when transitions at the input of a gate are separated by one or more gate delays.

We consider here a class of functions in which there are one or more operand bit-vectors and a single bit result which depends on the bits of the operand vectors with a decreasing probability. That is, the result z depends on the subvector $(x_1, x_2, ... x_i)$ with probability P_i and $P_j > P_k$ for $j < k$. Examples of these functions are sign detection of the sum of two fractions in the 2's complement system, sign detection of a fraction in signed-digit representation, magnitude comparison, pattern detection.

Traditional algorithms for this class do not take advantage of this probability of dependency and therefore result in unnecessary transitions. We consider modifications in the algorithm to reduce this effect. We consider in detail the case of sign detection and then comment on the other cases.

Related Work

This research area is relatively new and quite active because of the importance of this issue for future digital systems. As discussed in [1] the design of systems for low power requires a combination of techniques at four levels: technology, circuits, architectures, and algorithms. We concentrate on the level of structures formed by CMOS gates and latches. Moreover, from the various techniques to reduce power dissipation, we have chosen to reduce the number of transitions (and will not consider, for example, the possibility of reducing the voltage swing; this is a complementary technique which can be used in addition to the reduction of the number of transitions). Consequently, we limit our review to this topic.

Brodersen et al. [1] discuss models for static and dynamic CMOS structures. Static structures, unlike dynamic ones, exhibit extra transitions due to the glitching effect, which depend on the logic design, delay skew, and the number of logic levels.

Chandrakasan et al. [4] discuss transformations that reduce the power dissipation. In particular, they suggest that to reduce the number of transitions it is convenient to transform the system to balance the delay of paths and to reduce the depth of the implementation. An example considers a multi-operand addition and compares an unbalanced with a balanced tree implementation; the conclusion, obtained by simulation, is that the capacitance switched by the unbalanced tree is a factor of 1.5 larger than that of the balanced tree for a four-operand addition and 2.5 larger for the eight-operand case.

Powell and Chau [7] propose an analytical model for estimating power dissipation: the parameters used are the number of gates, the clock frequency, the average number of transitions per cycle per gate, and the gate short-circuit current and dynamic current components. They apply this model to parallel multipliers. The number of transitions is estimated as a fraction of the number of gates.

The research on arithmetic structures for low power is new and little has been done. Callaway and Swartzlander [2] use a gate-level simulator to determine the average number of transitions and delay of several CMOS adders and multipliers. The primitive circuits used are limited to inverters and 2 to 4 input AND and OR gates. The results are obtained by simulating the operation of each scheme for 50,000 randomly distributed input patterns. They conclude that, assuming a figure of merit equally weighted on delay and number of transitions, the carry-lookahead adder is the best for the considered operand lengths of 16, 32, and 64 bits. Similarly, they compared multipliers of linear array and tree type (Wallace and Dadda) and conclude that the best multiplier is a Dadda multiplier. There is neither a discussion of the reasons for the difference in the transition counts nor any suggestions regarding the changes in the logic design which would influence these counts. No consideration is given to multiple transitions per output.

Ko et al. [5] discuss a self-timed method to reduce glitches in low power CMOS adder, yielding a 25% reduction in power with a small delay overhead. Nagendra et al. [6] present simulation results for different types of parallel adders, ranging form 8 to 64 bits. A comparison of different adders is given based on power-delay product.

Arithmetic structures have also been used as examples in more general studies of low-power structures. For example, Brodersen et al. [1], consider an adder and discuss the effect of its topology on the capacitance switched and on the total delay. Chandrakasan et al. [4] show the effect of path balancing and reduction of delays on power dissipation in multi-operand adders.

2 Sign detection of $x + y$

This algorithm determines the sign of a number represented by the sum of two fractions in 2's complement representation. This is, for example, used in division and square root to determine the sign of the last residual which is in a redundant (e.g., carry-save) form. Another application is the magnitude comparison of two numbers, which can be performed by determining the sign of their difference.

More specifically, we determine the sign of $x + y$, where x and y are fractions in two's complement representation. The fractions are represented by the bit vectors

$$X = x_0.x_1, ..., x_i, ..., x_n$$

$$Y = y_0.y_1, ..., y_i, ..., y_n$$

so that

$$x = -x_0 + \sum_{i=1}^{n} x_i 2^{-i}$$

$$y = -y_0 + \sum_{i=1}^{n} y_i 2^{-i}$$

The sign of $x + y$ is obtained then by the expression

$$sign = x_0 \oplus y_0 \oplus c_0$$

where c_0 is the carry out of position 1 when $x + y$ is performed.

First we consider a conventional iterative algorithm and then we develop a modified algorithm that has a small average number of transitions. Since these implementations have a large worst-case delay, we use a similar approach to tree-type networks.

2.1 Standard iterative algorithm

This algorithm is

$$c_{i-1} = p_i c_i + g_i$$

where $p_i = x_i \oplus y_i$ and $g_i = x_i y_i$.

$$sign = x_0 \oplus y_0 \oplus c_0$$

The implementation is shown in Fig. 1. For each cell, there are four gate outputs where transitions occur (labeled g_i, p_i, a_i, and c_i). Transitions occur for the generation of the p and g signals, the carry-propagation chains. Fig. 2, shows an example of these chains.

We determine now the corresponding average number of transitions.

- Pair (p_i, g_i)

  Consider the number of transitions in the pair (p_i, g_i). The three possible values of this pair are $(0,0)$, $(1,0)$, and $(0,1)$. Since the input values x_i and y_i are uniformly distributed, the probability of occurrence of these values is

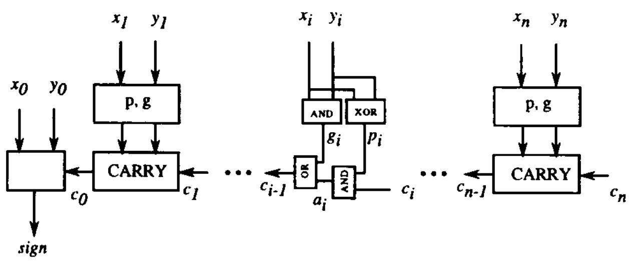


Figure 1: Conventional iterative network for sign detection.

Value	Probability
(0,0)	.25
(0,1)	.25
(1,0)	.50

Moreover, the number of transitions produced by a change is

	New value		
Previous value	(0,0)	(0,1)	(1,0)
(0,0)	0	1	1
(0,1)	1	0	2
(1,0)	1	2	0

Consequently, the average number of transitions per bit slice is

$$\begin{aligned} T_{av}(p_i, g_i) &= 0 \times 2^{-4} + 1 \times 2^{-4} + 1 \times 2^{-3} \\ &\quad +1 \times 2^{-4} + 0 \times 2^{-4} + 2 \times 2^{-3} \\ &\quad +1 \times 2^{-3} + 2 \times 2^{-3} + 0 \times 2^{-2} \\ &= 7/8 \end{aligned}$$

Since there are n bit slices the total is

$$T_{av}(p, g) = (7/8)n$$

- c_i

  We now determine the average number of transitions for the variable c_i. Consider the case in which

$$p_j = \begin{cases} 1 & \text{if } i + 1 \leq j \leq i + m \\ 0 & \text{if } j = i + m + 1 \end{cases}$$

  that is, there is a propagation chain of m bits to produce c_i (Fig. 2). In this case, c_i settles to a new value equal to g_{i+m+1}, but will first receive the old values of c_{i+1} to c_{i+m}. Since for these old values 0s and 1s are equally probable, the average number of transitions is $(m + 1)/2$. Since m can vary from 0 to $n-i$, and the probability of having m consecutive $p=1$ is $2^{-(m+1)}$ (for no p it is 1/2), we get the following average number of transitions for c_i

$$\begin{aligned} T_{av}(c_i) &= \sum_{m=0}^{n-i} 2^{-(m+1)}\frac{(m+1)}{2} \\ &= \sum_{m=0}^{\infty} 2^{-(m+1)}\frac{(m+1)}{2} \\ &\quad - \sum_{m=n-i+1}^{\infty} 2^{-(m+1)}\frac{(m+1)}{2} \\ &= \frac{2-(n-i+2)2^{-(n-i)}}{2} \end{aligned}$$

So, for all i we obtain

$$\begin{aligned} T_{av}(c) &= \sum_{i=1}^{n} \frac{2-(n-i+2)\times 2^{-(n-i)}}{2} \\ &= \sum_{j=1}^{n} \frac{2-(j+2)2^{-j}}{2} \approx n-2 \end{aligned}$$

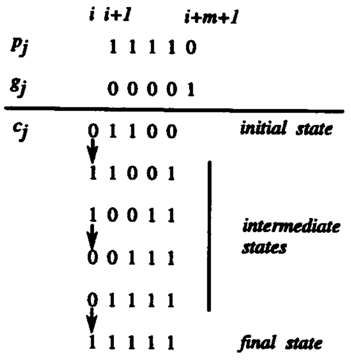


Figure 2: Example of carry transitions.

- a_i

For the output of the AND gate producing $a_i = p_i c_i$, we consider two events:

1. The change of p_i from old to new. There is a transition when $c_i(old) = 1$ and $p_i(old) \neq p_i(new)$. This occurs with probability 1/4, so this event produces a total average number of transitions equal to $n/4$.
2. The effect of the changes in c_i. This change propagates through the AND gate when $p_i(new) = 1$. Consequently, the corresponding number of transitions is $(1/2)T_{av}(c) = (1/2)(n-2)$.

Combining 1 and 2 we get (the number of transitions are added because the events occur at different times), ,

$$T_{av}(a) = \frac{3n}{4} - 1$$

The total average number of transitions is

$$T_{av} = \frac{7}{8}n + n - 2 + \frac{3}{4}n - 1 \approx \frac{21}{8}n$$

2.2 Low-transition iterative algorithm

Since only the most significant carry-propagation chain affects c_0, we now modify the iterative algorithm so that the transitions in the other chains are avoided.

As shown in Fig. 3, we produce a signal that propagates left-to-right to determine j, the position of the most-significant bit with $p_j = 0$. Then $c_0 = g_j$. This value can be placed on a bus or propagated back to the most-significant position.

Note that the signal h also inhibits the computations of p and g.

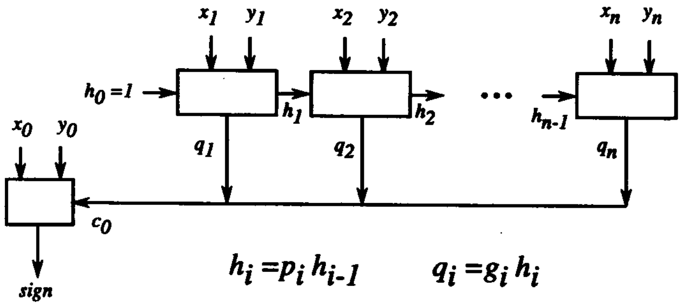


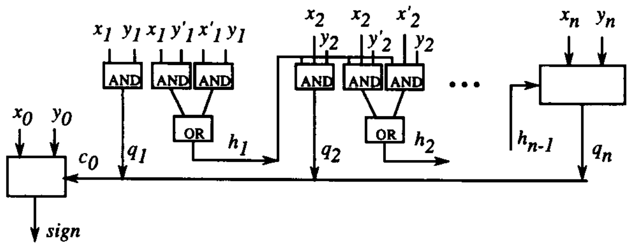
Figure 3: Reduced transition scheme.

We define the binary variable h_i, such that $h_i = 1$ indicates that the most-significant propagation chain includes bit i. That is, $h_i = 1$ implies $p_k = 1$ for $k \leq i$. So, the algorithm is

```
1. Initialize all $h_i = 0$ for $i \geq 1$.
2. $h_1 = x_1 y_1' + x_1' y_1$
3. For $i = 1$ to $n-2$ do
     $h_{i+1} = h_i x_{i+1} y_{i+1}' + h_i x_{i+1}' y_{i+1}$
4. $c_0 = x_1 y_1 + OR_{i=1}^{n-1}(h_i x_{i+1} y_{i+1})$
```

An implementation is shown in Fig. 4.

We now determine the average number of transitions. These transitions occur only in the most-significant carry-propagation chain, that is up to the most-significant cell for which $h_i = 0$.



x_1 y_1 x_1 y'_1 x'_1 y_1
x_2 y_2 x_2 y'_2 x'_2 y_2
x_n y_n
AND
OR
x_0 y_0
c_0
q_1
h_1
q_2
h_2
h_{n-1}
q_n
sign

Figure 4: Implementation reduced transition count.

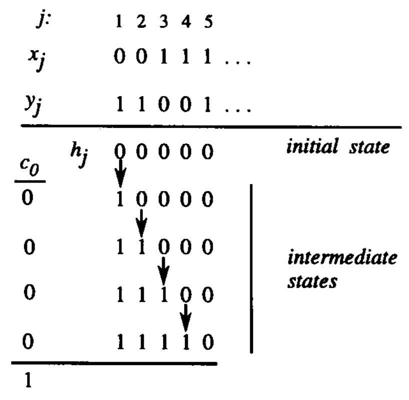


Figure 5: Example of transitions in h network.

If this most-significant propagation chain has length m, the number of transitions is (Fig. 5)

- For the h's, we have $2(m)$ transitions (one for the AND gate and one for the OR gate).
- For c_0 we have .5 transitions (one transition with probability .5)

Since the probability of a most-significant chain of length m is $2^{-(m+1)}$, the average number of transitions is

$$T_{av} = \sum_{m=0}^{n} 2^{-(m+1)} \times (2m) + .5 \approx 2.5$$

For the scheme to operate correctly, it is necessary to clear the $h's$ after each operation. Therefore, to obtain the total number of transitions it is necessary to add those due to the clearing. On the average, these transitions equal to the number of transitions during operation, resulting in

$$T_{av} = 4.5$$

Note that this average is a constant independent of n.

We have shown a relatively simple modification of the iterative algorithm that produces a reduction in the average number of transitions from about $(21/8)n$ to just 4.5. This has been done by eliminating unnecessary transitions. For example, for a floating-point representation of a 53-bit fraction the number of transitions is reduced from about 130 to 4.5.

However, the iterative algorithms presented have the disadvantage that the worst-case delay is proportional to n. We now extend the approach to faster algorithms.

2.3 Tree-type algorithm

The previous iterative algorithms have a worst-case delay that is proportional to n. Faster algorithms exist with worst-case delay proportional to $\log n$. We first describe such an algorithm and then modify it to reduce the number of transitions.

Consider a tree-type algorithm. The basic module has k input pairs

$$((P_1^{j-1}, G_1^{j-1}), ...(P_k^{j-1}, G_k^{j-1}))$$

and produces one output pair P^j, G^j. These modules are organized as a tree, as shown in Fig. 6, so that level j consists of $n/(k^j)$ modules. Moreover,

- The inputs to level 1 are the bit-level $p_i's$ and $g_i's$.
- The module function is described by the following expressions

$$P^j = AND_{i=1}^{k} P_i^{j-1}$$

$$G^j = G_1^{j-1} + P_1^{j-1} G_2^{j-1} + ... + P_1^{j-1} P_2^{j-1} ... P_{k-1}^{j-1} G_k^{j-1}$$

The value of c_0 corresponds to G of the last level. The worst case delay in this case is

$$Delay = Delay(p,g) + log_k(n) \times Delay(module) \quad (1)$$

We now determine the average number of transitions. As indicated in the iterative scheme, the average number of transitions in the stage that computes the pairs p_i, g_i is

$$T_{av}(p,g) = (7/8)n$$

Performing the same type of analysis for the first stage of modules we get for each pair P^1, G^1 the following probabilities:

Value	Probability
(0,0)	$p_{00} = 0.5(1 - 2^{-k})$
(0,1)	$p_{01} = 0.5(1 - 2^{-k})$
(1,0)	$p_{10} = 2^{-k}$

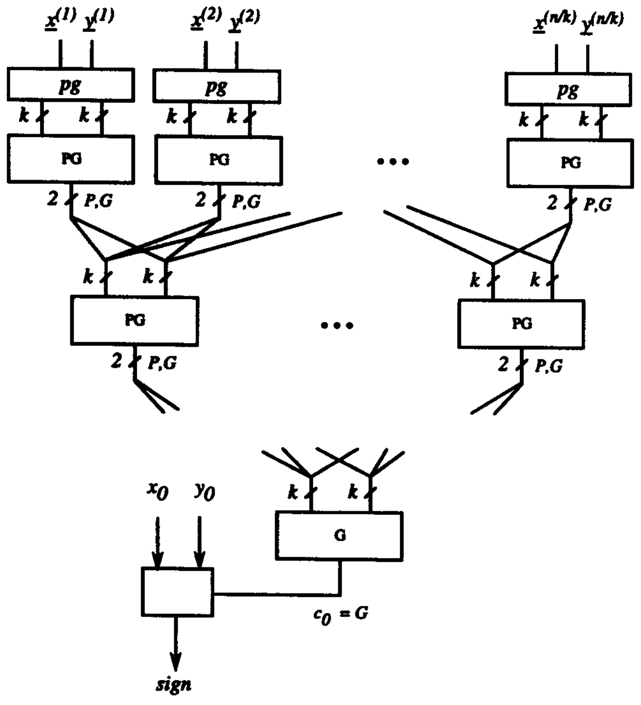


Figure 6: Tree network for sign detection.

If the expressions for P's and G's are implemented as two-level gate networks, the number of transitions is

	New value		
Previous value	(0,0)	(0,1)	(1,0)
(0,0)	0	2	1
(0,1)	2	≈ 2	3
(1,0)	1	3	0

Therefore, the average number of transitions for the (n/k) modules is

$$\begin{aligned} T_{av}(level1) &= (n/k)(0 \times (p_{00}^2 + p_{10}^2) + 1 \times (p_{00}p_{10} + p_{10}p_{00}) \\ &\quad +2 \times (p_{00}p_{01} + p_{01}p_{00} + p_{01}p_{01})) \\ &\quad +3 \times (p_{01}p_{10} + p_{10}p_{01})) \end{aligned}$$

which can be approximated by

$$T_{av}(level1) \approx \frac{n}{2k}(3 + 2 \times 2^{-k})$$

For a typical value of $k = 4$ we get $T_{av}(1) \approx (3/8)n$.

For the second level the value is obtained by replacing k by k^2 in the expression above. That is,

$$T_{av}(level2) \approx \frac{n}{2k^2}(3 + 2 \times 2^{-k^2})$$

For $k = 4$ this is $T_{av}(level2) \approx (3/32)n$. Since this is about 10% of $T_{av}(p,g)$+$T_{av}(level1)$, it can be neglected. Similarly for the rest of the stages.

Consequently, the average is

$$T_{av} \approx (\frac{7}{8} + \frac{3 + 2 \times 2^{-k}}{2k})n \qquad (2)$$

This is about n transitions. We now develop a low-transition variation.

2.4 Tree-type algorithm with reduced number of transitions

As before, we reduce the number of transitions by eliminating unnecessary transitions. Consider the implementation of Fig. 7 where the generation of p, g of all the bits except the most significant k is inhibited when P_l (the P signal of the leftmost module) is 0. This is correct since when this P is 0 the carry c_0 is equal to G_l. That is, only the most-significant chain determines c_0 and

$$c_0 = P_l'G_l + P_lG$$

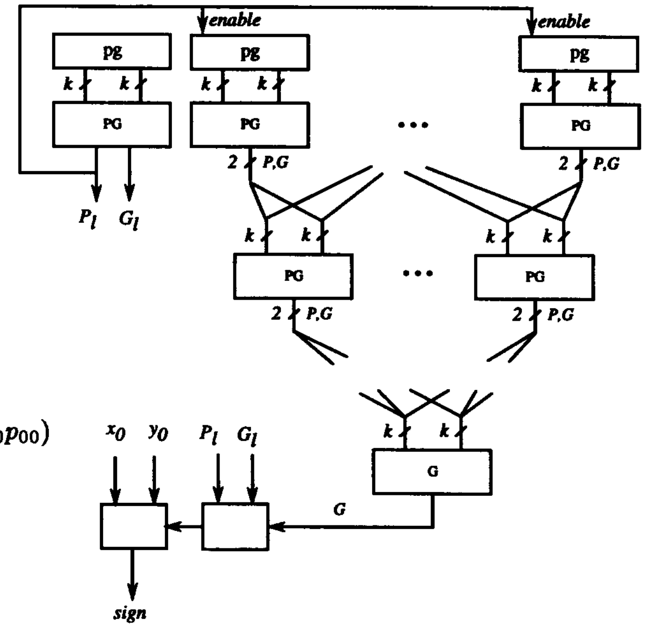


Figure 7: Tree network with low transition count.

This modification has an effect on delay and on number of transitions. The new worst-case delay is

$$Delay = 2Delay(p,g) + [(log_k n) + 1]Delay(module)$$

so that the increase in delay with respect to (1) is relatively small.

Number of transitions

As shown above, the number of transitions in the leftmost module is

$$T_{av}(l) \approx (7/8)k + (1/2)(3 + 2 \times 2^{-k})$$

In addition, the rest of the network has transitions when P_l changes from 1 to 0 or when the new $P_l = 1$. That is, there are transitions except when the P_l was 0 and remains 0. The probability of this transition-producing event is

$$(1 - (1 - 2^{-k})^2) \approx 2^{-k+1}$$

Consequently,

$$T_{av}(rest) \approx 2^{-k+1}[\frac{7}{8} + \frac{3 + 2 \times 2^{-k}}{2k}](n - k)$$

The total number of transitions is

$$T_{av} \approx [\frac{7}{8} + \frac{3 + 2 \times 2^{-k}}{2k}](k + 2^{-k+1}n)$$

As an example for $k = 4$ and $n = 64$ we get

$$T_{av} \approx 15$$

which corresponds to a reduction by a factor of about five with respect to (2).

3 Comparison of small numbers

The previous sections considered the case in which there is a uniform distribution of input values. However, in some practical situations this might not be the case, one example being a sequence of comparisons in which the inputs correspond to small numbers.

Specifically, consider that the input values have the s most-significant bits equal to 0. In such a case, when performing the subtraction for the comparison, the s most-significant p signals have value 1. If $s \geq k$, the scheme of inhibition presented in Section 2.4 would not work since the enabling signal would be always 1 and the p, g signals would be active. Consequently, to have a low-power scheme it is necessary to introduce some modifications.

A possibility is to make the inhibition dependent on the size of the integers that are to be compared (maximum size in a sequence of comparisons). The enabling signal is now produced by the most-significant module for which all input bits can be nonzero.

An implementation is shown in Fig. 8. Register R has one bit per k-bit byte of the operands. It is set to 1 if the corresponding byte of both operands to compare are assured to be 0. Consequently byte j is the most-significant byte which can have bits different than zero if $R_{j-1}R'_j = 1$.

Module j produces the control signal

$$Q_j = P_j R_{j-1} R'_j$$

so that the enabling signal for module i is

$$E_i = R_{i-1}R'_i + OR_{j=1}^{i-1} Q_j$$

Moreover, to have the most-significant disabled modules produce a propagate signal we make

$$P_i^* = P_i + R_i$$

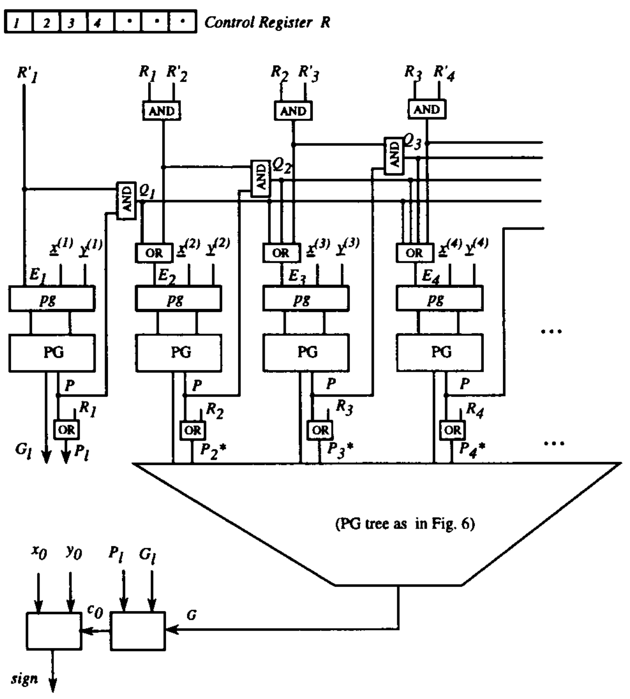


Figure 8: Tree network for comparison of small operands.

The following example illustrates the operation of the scheme when comparing x and y:

i	0	1	2	3	4
R_i		1	1	0	0
x	0	0000	0000	0011	1011
$-y$	1	1111	1111	1101	1001
E_i		0	0	1	0
P_i		0	0	0	0
P_i^*		1	1	0	0
Q_i		0	0	0	0
G_i		0	0	1	0

$$P_l = 1, \quad G_l = 0, \quad G = 1$$

Therefore, $c_0 = 1$, and $sign = 0$, i.e., $x > y$.

The mode of operation of this implementation is similar to that of the system in Section 2.4, that is, most of the time just the most-significant enabled module is active (since the probability that for it P=1 is low). Consequently, the average number of transitions is similar to that of the scheme of Section 2.4.

Additional transitions occur when the contents of register R is changed, but we assume that this occurs infrequently.

4 Characteristics of the Schemes and Other Examples

The schemes presented are based on the elimination of unnecessary transitions. In particular, this idea has been applied to an example in which the output is determined by the most-significant carry chain. Moreover, the probability of a long chain is low, so that it is advantageous to eliminate these unnecessary transitions.

Other examples of this type are

- Sign detection of a number in signed-digit representation.
- Determination of the position of the leading 1 in a word. This is used, for instance, for normalization in a floating-point adder.
- Detection of a pattern. This is similar to the example presented, but the reduction in the number of transitions depends on the length of the pattern.

Acknowledgments. This research has been supported in part by the NSF Grant MIP-9314172 "Arithmetic Algorithms and Structures for Low-Power Systems."

References

[1] R. Brodersen, A. Chandrakasan and S. Sheng, "Low-Power Signal Processing Systems," *VLSI Signal Processing V*, Eds. Yao, Jain, Przytula, and Rabaey, IEEE Press, 1992, pp.3–13.

[2] T. K. Callaway and E.E. Swartzlander, "Optimizing Arithmetic Elements for Signal Processing," *VLSI Signal Processing V*, Eds. Yao, Jain, Przytula, and Rabaey, IEEE Press, 1992, pp.91–100.

[3] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen, "An Approach for Power Minimization using Transformations," *VLSI Signal Processing V*, Eds. Yao, Jain, Przytula, and Rabaey, IEEE Press, 1992, pp.41–50.

[4] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, Vol.27, April 1992, pp.473–484.

[5] U. Ko, P.T. Balsara, and W. Lee, "A Self-Timed Method to Minimize Spurious Transitions in Low Power CMOS Circuits," *1994 IEEE Symposium on Low Power Electronics*, pp. 62–63, 1994.

[6] C. Nagendra, R.M. Owens, and M.J. Irwin, "Low Power Tradeoffs in Signal Processing Hardware Primitives," *VLSI Signal Processing, VII*, Eds. J. Rabaey, P.M. Chau, and J. Eldon, IEEE Press, 1994, pp.276–285.

[7] S.R. Powell and P.M. Chau, "Estimating Power Dissipation of VLSI Signal Processing Chips: the PFA Technique," *VLSI Signal Processing, IV*, Eds. H.S. Moscovitz, K. Yao, and R. Jain, IEEE Press, 1990, pp.250–259.