

Comments

Comments on "Evaluation of $A + B = K$ Conditions Without Carry Propagation"

Behrooz Parhami

Abstract—A special carry-free circuit for the evaluation of conditions of the type $A + B = K$ is proposed by Cortadella and Llaberia, and its usefulness for reducing the negative effects of conditional branches in pipelined architectures is noted. It is shown that the same advantages are offered by another equally simple circuit based on carry-save redundant numbers and (3, 2)-counters. This alternative circuit has other potential applications and much of it may in fact be already present in some ALU's.

Index Terms—Addition, carry propagation, carry-save numbers, comparators, conditional branches, parallel counters, pipelined architectures, redundant number representation.

I. INTRODUCTION

A special add-on circuit, comparable in area complexity to a ripple-carry adder, has recently been proposed for evaluating conditions of the type $A + B = K$ without the need for performing a carry-propagate addition [1]. Internal "propagate" and "generate" signals $p_i = a_i \oplus b_i$ and $g_i = a_i \wedge b_i$ from the addition $A + B$ are combined with the bits k_i of K , in a local way, and the resulting n -bit vector Z is fed to an n -input AND circuit computing the final test result $E = z_n \wedge z_{n-1} \wedge \dots \wedge z_1$. The proposed circuit is positioned between the logic for producing the p and g signals and the ALU's main adder. Usefulness of the circuit, named "fast adder-comparator," is demonstrated for reducing conditional branch penalties in a pipelined CPU.

In this note, I utilize the simple equivalence of $A + B = K$ and $A + B + (2^n - 1 - K) = 2^n - 1$ to convert the problem into a 3-operand carry-save addition [4] and a subsequent all-ones detection for the resulting carry-save redundant number.

II. THE ALTERNATIVE CIRCUIT

Let A, B , and K be n -bit 2's-complement binary numbers. To determine if $A + B - K = 0$, one can add A, B , and $K^{\text{comp}} = 2^n - 1 - K$ (bitwise complement of K) and check if the sum is equal to $2^n - 1$ which is represented by the n -bit all-ones vector. Instead of completely evaluating $A + B + K^{\text{comp}}$, one can use a row of full adders with no carry propagation, also known as (3, 2)-counters [4], to find two numbers S and C satisfying $A + B + K^{\text{comp}} = S + C$. The problem then reduces to checking the condition $S + C = 2^n - 1$ or $S = 2^n - 1 - C = C^{\text{comp}}$. This is accomplished by a row of n two-input XOR gates feeding an n -input AND circuit that produces the final comparison result E .

The conversion of the sum $A + B + K^{\text{comp}}$ to the two numbers S and C , that together can be viewed as a single redundant radix-2 number with the digit set $\{0, 1, 2\}$, is a standard technique in computer arithmetic [4]. The verification of $S = C^{\text{comp}}$ is also a special case of a reduction method that I have used previously

Manuscript received December 31, 1992; revised June 16, 1993.

The author is with the Department of Electrical & Computer Engineering, University of California, Santa Barbara, CA 93106-9560.
IEEE Log Number 9213764.

for zero, sign, and overflow detection in generalized signed-digit arithmetic [2], [3].

III. COMPARISON

The alternative circuit defined in Section II consists of n full adders, n two-input XOR gates, and an n -input AND circuit. It can be positioned before the inputs to the ALU's main adder and implies logarithmic delay when the n -input AND is implemented as a tree of smaller AND gates. With respect to speed, the two alternatives are virtually identical. The critical path in the design of [1] goes through 3 XOR gates (one to generate p_i and two in the adder-comparator cell) and n -input AND circuit. In my design, the critical path contains a full-adder cell (2 XOR delay), an XOR gate, and an n -input AND circuit. While p_i still needs to be generated for the ALU's fast adder, the corresponding circuit is no longer in the path of the add-on comparator.

The hardware complexity of my design is higher than that of [1] by about one XOR gate per bit slice. However, there are two redeeming factors. First, I use standard full adder cells that have been fully optimized for many different technologies. Second, my circuit is potentially more useful in that the 3-to-2 reduction can also be used to speed up multiple-operand addition and multiplication in some applications. If such a 3-to-2 reduction circuit is already present in the ALU to facilitate multiplication, then the cost of my approach is indeed very small.

REFERENCES

- [1] J. Cortadella and M. Llaberia, "Evaluation of $A + B = K$ conditions without carry propagation," *IEEE Trans. Comput.*, vol. 41, no. 11, pp. 1484-1488, Nov. 1992.
- [2] B. Parhami, "Generalized signed-digit number systems: a unifying framework for redundant number representations," *IEEE Trans. Comput.*, vol. 39, no. 1, pp. 89-98, Jan. 1990.
- [3] —, "On the implementation of arithmetic support functions for generalized signed-digit number systems," *IEEE Trans. Comput.*, vol. 42, no. 3, pp. 379-384, Mar. 1993.
- [4] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*. New York: Holt, Rinehart & Winston, 1982.

Comments on "Decomposition of Complex Multipliers Using Polynomial Encoding"

Rajendra Katti

Abstract—We present a better way of decomposition of complex multipliers using polynomial encoding than the method presented in the paper, "Decomposition of Complex Multipliers Using Polynomial Encoding." The decomposition described in this paper makes use of smaller multipliers which results in smaller ROM's if ROM table look-ups are used to implement multipliers.

Manuscript received March 15, 1993.

The author is with the Department of Electrical Engineering, North Dakota State University, Fargo, ND 58105.
IEEE Log Number 9213766.

Index Terms—Computer arithmetic, complex multiplication, cyclic convolution, polynomial rings, ROM's.

I. INTRODUCTION

Decomposing a large-word length multiplier into smaller word length ones is important when ROM table look-ups are to be used [1]. If two n -bit numbers are to be multiplied then a ROM having 2^{2n} words needs to be used for table look-up operations. For large values of n such ROM's are not feasible.

In [1], one way to decompose a complex multiplier using n -bit word lengths into smaller multipliers each relying on $(\frac{n}{4} + 1)$ -bit word lengths is given. In these comments, we give a method of decomposing a complex multiplier using n -bit word lengths into smaller multipliers each relying on $\frac{n}{2^m}$ -bit word lengths ($0 < m \leq \log_2 n$). If we choose m equal to 2 then the smaller multipliers rely on $\frac{n}{4}$ -bit word lengths and not $(\frac{n}{4} + 1)$ -bit word lengths. If the smaller multipliers use a ROM, then the ROM would be one fourth the size of the ROM needed in [1]. The large complex numbers are encoded as polynomials of degree $(2^{m+1} - 1)$ in the ring of polynomials modulo $(x^{2^{m+1}} - 1)$. Complex multiplication is then performed with a 2^{m+1} point cyclic convolution.

II. THE ENCODING

Consider a complex number,

$$A = R(A) + jI(A),$$

where $R(A)$ and $I(A)$ are the n -bit real and imaginary parts of A . Without loss of generality, we assume that $n = 2^b$ for some b . Let us decompose $R(A)$ and $I(A)$ into 2^m parts with 2^{b-m} bits in each part. If m equals 2 then $R(A)$ and $I(A)$ can be written as

$$\begin{aligned} R(A) &= R_3(A)2^{3n/4} + R_2(A)2^{n/2} + R_1(A)2^{n/4} + R_0(A) \\ I(A) &= I_3(A)2^{3n/4} + I_2(A)2^{n/2} + I_1(A)2^{n/4} + I_0(A). \end{aligned}$$

The complex number A can be expressed as

$$A = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7,$$

where

$$\begin{aligned} a_0 &= 2^{3n/4}R_3(A) \\ a_1 &= 2^{3n/4}I_3(A) \\ a_2 &= -2^{n/2}R_2(A) \\ a_3 &= -2^{n/2}I_2(A) \\ a_4 &= 2^{n/4}R_1(A) \\ a_5 &= 2^{n/4}I_1(A) \\ a_6 &= -R_0(A) \\ a_7 &= -I_0(A), \end{aligned}$$

and $x = e^{j\pi/2} = j$. The complex number A has been encoded as a polynomial of degree 7 with $\frac{n}{4}$ -bit coefficients. In [1], a complex number was encoded in a similar manner but with $x = e^{j\pi/4} = \frac{1}{\sqrt{2}} + \frac{j}{\sqrt{2}}$. This results in the coefficients being $(\frac{n}{4} + 1)$ -bits each and a division by $\sqrt{2}$ is required in order to compute the coefficients. In the decomposition proposed in this paper we have eliminated division by $\sqrt{2}$, resulting in error free multiplication. We have also reduced the size of the coefficients from $(\frac{n}{4} + 1)$ bits to $\frac{n}{4}$ bits, resulting in ROM's that are one fourth the size of ROM's needed in [1].

III. COMPLEX MULTIPLICATION

Denote a complex number by the coefficients in its polynomial encoding. Therefore, two complex numbers A and B can be represented as

$$\begin{aligned} A &= (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \\ B &= (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7). \end{aligned}$$

The product of A and B ,

$$C = A \times B,$$

is the product of two polynomials modulo $(x^8 - 1)$. Direct computation yields,

$$C = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6 + c_7x^7, \quad (1)$$

where

$$c_k = \sum_{i+j=k} a_i b_j. \quad (2)$$

Note that there are eight terms in c_k and the addition of i and j is modulo 8 addition. Substituting $x = e^{j\pi/4}$ in (1) and evaluating, we get

$$C = P + jQ,$$

where

$$\begin{aligned} P &= c_0 - c_2 + c_4 - c_6 \\ Q &= c_1 - c_3 + c_5 - c_7. \end{aligned}$$

Evaluating P and Q requires less additions than required in [1].

IV. EXAMPLE

Consider $n = 8$, $A = 150 + j230$ and, $B = 102 + j218$, then

$$\begin{aligned} A &= (2^6 \times 2, 2^6 \times 3, 2^4 \times -1, 2^4 \times -2, 2^2 \times 1, 2^2 \times 1, -2, -2) \\ B &= (2^6 \times 1, 2^6 \times 3, 2^4 \times -2, 2^4 \times -1, 2^2 \times 1, 2^2 \times 2, -2, -2). \end{aligned}$$

From (2), we get

$$\begin{aligned} c_0 &= 7216 \\ c_1 &= 37104 \\ c_2 &= 31856 \\ c_3 &= -13352 \\ c_4 &= -7956 \\ c_5 &= 4104 \\ c_6 &= 2244 \\ c_7 &= -1600. \end{aligned}$$

From (1), we get

$$C = -34840 + j56160.$$

V. CONCLUSION

A way of decomposing large-word length complex multipliers into multipliers of smaller word length has been presented. This method reduces the ROM size to one fourth the size required in [1] and produces no error as division by $\sqrt{2}$ has been eliminated.

REFERENCES

- [1] A. Skavantzios and T. Stouraitis, "Decomposition of complex multipliers using polynomial encoding," *IEEE Trans. Comput.*, vol. 41, no. 10, pp. 1331-1333, Oct. 1992.

Comments on "A Characterization of Binary Decision Diagrams"

Ingo Wegener

Abstract—The above mentioned paper presents a characterization of BDD's in terms of the complexity of some computational problems. In these comments, some incorrectly stated restrictions on the "number of repeated variables" are corrected and results on the translation problem (to include EXOR and NEXOR gates) are generalized.

Index Terms— Binary Decision Diagrams, complexity results, free BDD's, ordered BDD's, repeated BDD's.

I. INTRODUCTION

For the motivation of the investigation of the different variants of binary decision diagrams (BDD's) refer to [2]. We only repeat the definitions necessary to understand this correspondence (for figures see [2]). A BDD is a labeled, directed, acyclic graph with one source and two sinks. The sinks are labeled by 0 and 1. The other nodes are labeled by Boolean variables and have two outgoing edges labeled by 0 and 1. The BDD represents the Boolean function f defined in the following way. In order to compute $f(a)$ we start at the source. At a node with label x_i we follow the edge labeled a_i . The label of the sink finally reached is the value of $f(a)$. A free BDD is a BDD where on each path and for each variable x_i there is at most one node labeled x_i . Otherwise the BDD is called repeated. A variable x_i is called repeated if there is some path containing at least two nodes labeled x_i . Hence, also the notion BDD with k repeated variables is well defined. A BDD is called ordered if there exists a total order \leq on the variables such that $x_i \leq x_j$ if the BDD contains an edge from a node labeled x_i to a node labeled x_j .

In Section II, we correct some incorrectly stated restrictions on the "number of repeated variables" of two theorems. In Section 3 we consider the translation problem which is the problem to construct an ordered BDD for some Boolean function f given as a circuit.

II. ON THE COMPLEXITY OF ALGORITHMS

The satisfiability problem for a BDD is to decide whether there exists some input a such that the 1-sink is reached for input a . Chakravarty has proved ([2, lemma 4.5]) that the satisfiability problem for BDD's with r repeated variables and m nodes can be solved in time $O(2^r m)$. From this lemma the following result (2, theorem 4.6) is deduced. The satisfiability problem for BDD's with $r \leq (\log m)^k$ repeated variables (k constant) can be solved in polynomial time $O(m^{k+1})$. Since only polynomial time algorithms

Manuscript received September 16, 1993; revised October 25, 1993. This work was supported in part by DFG Grant We 1066/7-1.

The author is with FB Informatik, LS II, Universität Dortmund, 44221 Dortmund, Germany.

IEEE Log Number 9215088.

can be considered as efficient algorithms it is essential to describe for which choice of the parameters the running time of an algorithm is polynomially bounded. But the conclusion of the stated theorem is wrong, if $k > 1$.

If $r = (\log m)^k$, then

$$2^r m = m^{(\log m)^{k-1}} m.$$

If $k > 1$, the exponent of m is not constant and the function is not a polynomial with respect to m . If $r \leq k \log m$ (or equivalently $r \leq \log(m^k)$) for some constant k , then

$$2^r m \leq m^{k+1}$$

is a polynomial with respect to m . Theorem 4.6 in [2] becomes only correct, if the assumption $r \leq (\log m)^k$ is replaced by $r \leq k \log m$. This holds also for the second result in Theorem 4.6 on the signal probability problem. By the same arguments the upper bound $(\log n)^k$ in [2, Theorem 2.2.4] on the number of HFPs (head fanout points) has to be replaced by $k \log n$ in order to correct the theorem. Hence, [2, Theorem 2.2.4 and Theorem 4.6] hold only under much stronger assumptions than stated in [2].

III. THE TRANSLATION PROBLEM

The translation problem is the problem to construct an ordered BDD for a Boolean function f from a circuit representation of f [2]. The algorithm in [2] does not work in the presence of EXOR or NEXOR gates. We show that the algorithm (and the related theoretical results for the translation problem) in [2] can be generalized to include EXOR and NEXOR gates.

Theorem 1: For Boolean functions f on n variables represented by fanout free circuits with binary gates an ordered BDD of size bounded by $n^{\log 3} \leq n^{1.585}$ can be constructed in time $O(n^{\log 3})$. (Here \log is the logarithm with base 2.)

Proof: Fanout free circuits with binary gates are binary trees. Our algorithm treats the gates and variables in preorder. In a preprocessing step we compute for each gate the number of variables the gate depends on. Let $f = f_1 \otimes f_2$ where \otimes is a binary Boolean operator and f_1 and f_2 are the functions represented by the subcircuits. Since the circuit is fanout free, f_1 and f_2 depend on disjoint sets of variables V_1 and V_2 . If $|V_1| \geq |V_2|$, we choose a variable ordering where the variables in V_1 are tested before the variables in V_2 . If $|V_1| < |V_2|$, the variables in V_2 are tested first.

It is already discussed in [2] how the gates which are not of type EXOR (or NEXOR, the negation of EXOR) can be treated. We consider an EXOR gate (NEXOR gates are treated similarly). W.l.o.g., $|V_1| \geq |V_2|$. Then ordered BDD's for f_1 and f_2 are computed recursively. If these modules are combined as in Fig. 1, the resulting ordered BDD computes $f = \text{EXOR}(f_1, f_2)$.

Hence, in all cases we obtain an ordered BDD for f by combining at most one ordered BDD for f_1 or \bar{f}_1 , at most one ordered BDD for f_2 and at most one ordered BDD for \bar{f}_2 . Let $C(n)$ be the largest size of an ordered BDD we obtain by starting with a fanout free circuit in n variables. We do not count the sinks. Then our recursive algorithm leads to the following estimations.

$$C(1) = 1$$

and

$$C(n) \leq \max \{C(n-k) + 2C(k) \mid 1 \leq k \leq \lfloor n/2 \rfloor\}.$$