A Redundant Binary Euclidean GCD Algorithm

Shrikant N. Parikh

Programming Systems
IBM
Westlake, Texas 76299

David W. Matula

Dept. of Comp.Sci. and Eng.
SMU
Dallas, Texas 75275

Abstract

An efficient implementation of the Euclidean gcd algorithm employing the redundant binary number system is described. The time complexity is O(n) utilizing O(n) 4-2 signed 1-bit adders to determine the gcd of two n-bit integers. The process is similar to that employed in SRT division. The efficiency of the algorithm is competitive, to within a small factor, with floating point division in terms of the number of shift and add/subtract operations. The novelty of our algorithm is based on properties derived from our scheme of normalization of signed bit fractions. Our implementation is noted to be well suited to systolic hardware design.

1 Introduction

A redundant binary version of the Euclidean algorithm for the greatest common divisor (gcd) is developed. Both our methodology and the performance achieved is similar to that of the SRT division process [1,4,7,12,14,17,18, 20,21].

The traditional form of the original Euclidean algorithm [5] employs recursive integer division and is based on the fact that $\gcd(p - kq, q) = \gcd(p, q)$ for any k. The standard binary form of the Euclidean algorithm employs binary shift and subtract, and thus relies specifically on the fact that $\gcd(p - 2^i q, q) = \gcd(p, q)$. Using carry look ahead adders, the gcd of two n-bit numbers can be found by an implementation of the binary form of the Euclidean algorithm in time $O(n \log n)$ with $O(n)$ hardware. The binary form relies on strict adherence to the monitoring of shifts and argument normalization to determine when to swap the "remainder" $p' = p - 2^i q$ and the "divisor" q, corresponding to what would be the next recursive division in the traditional algorithm.

As noted in [11], the direct attempt to introduce redundant binary constant time subtraction into the binary form of the Euclidean algorithm yields non trivial side effects in compensating for the lack of an appropriate notion of normalization of signed bit numbers. Our first contribution in this paper is the introduction of a convenient and useful notion of normalization of signed bit fractions. The resulting normalized signed bit fraction will have magnitude in the range $(\frac{1}{4}, 1)$, and may be confirmed to be normalized by simply checking at most three leading signed bits.

Our principal contribution is then the specification of a redundant binary form of the Euclidean algorithm (RBEA). The algorithm treats the n-bit integers p, q as $n + 1$ signed bit fractions $b_0.b_1b_2 \cdots b_n$ which are each normalized to the range $(-1, -\frac{1}{4})$ or $(\frac{1}{4}, 1)$ by a modified shift operation, with b_0 available for a subsequent possible overflow condition. A brief computation on a few leading bits of p, q (equivalently, table lookup) allows us to determine a particular one of the three terms $p - q$, $p - 2q$, or $2p - q$ which must have magnitude less than $\frac{1}{2}$. This term replaces p and is appropriately left shifted at least once until normalized according to our signed bit normalization criteria.

Note that the selection of one of the terms $p - 2q$, $2p - q$, or $p - q$ to replace p in the presence of redundant binary representation of p and q is analogous to the combined SRT division steps of i) digit selection (table lookup), ii) divisor multiple formation, and iii) partial remainder update. As in SRT division, we are similarly assured that the new value may be appropriately shifted at least once. As a consequence of our redundant binary normalization criteria and the term selection process, we then are able to prove that Algorithm RBEA determines the gcd of two n-bit integers employing at most $2n$ shift/subtract operations. With this algorithm we thus need only $O(n)$ time with $O(n)$ 4-2 signed 1-bit adders.

In section 2 we state for reference the Euclidean gcd algorithm in both its traditional and binary forms.

In section 3 we introduce criteria for the normalization of signed bit fractions. We specify our redundant binary Euclidean algorithm in section 4, and then prove it may be implemented in $O(n)$ time with $O(n)$ 4-2 signed 1-bit adders.

Regarding extensions we first note that the alternative right shift binary gcd algorithm has been shown [3] to be implementable in an elegant systolic design. Algorithm RBEA appears similarly viable for systolic design. In addition, algorithm RBEA may readily be extended [11] to also obtain a solution α, β of

$$\alpha p - \beta q = \gcd(p, q),$$

according to the generalized Euclidean gcd result. Thus our methodology provides the foundation for an efficient systolic design yielding the more comprehensive results of the expanded Euclidean gcd algorithm, results which can not be obtained from the right shift binary gcd algorithm.

2 Euclidean Algorithm

A summary of the Euclidean algorithm and its binary version is given in this section based on notation and discussion from [11].

Algorithm EA (Euclidean Algorithm)

For any (p, q), $q \neq 0$, this algorithm computes the gcd of p, q.

1. Let $b_{-2} = p$; $b_{-1} = q$.
2. For $i = 0, 1 \ldots$, while $b_{i-1} \neq 0$ determine a_i as the quotient and b_i as the remainder (of the same sign as b_{i-2}) of the division of b_{i-2} by b_{i-1}, so $b_i = -b_{i-1} \times a_i + b_{i-2}$.

The algorithm terminates when $b_i = 0$. At this point b_{i-1} is the gcd of p and q.

The binary version [11] replaces each division by a sequence of shift-and-subtracts climaxed by a swap. The swap corresponds to the interchange of role of divisor and remainder for initiating the next division step of the traditional Euclidean algorithm.

Algorithm BEA (Binary Euclidean Algorithm)

Given initializations p,q of the registers P,Q such that (p,q) $\neq$ (0,0), this algorithm terminates with the gcd value right justified in the P register. U_P and U_Q are auxiliary registers initialized to 1 which identify the unit-position of each argument.

```
While P and Q not normalized do
   leftshift P, Q, UP, UQ;
While Q ≠ 0 do begin
   While Q not normalized do
      leftshift Q and UQ;
   Loop
      While P not normalized do
         Exit Loop if UQ= UP;
         Leftshift P and UP;
      End;
      If sign(P) = sign(Q) then
         P := P−Q
         else P := P+Q;
   EndLoop;
   Swap(P,Q); Swap(UP,UQ);
End;
While UP≠ 1 do rightshift P and UP.
```

In order to introduce redundant binary representation to expedite the addition/subtraction operations in Algorithm BEA, an appropriate notion of normalization of redundant binary operands must be developed.

3 Redundant Binary Representation and Normalization

Redundant binary representation employs the signed bits $\{\bar{1},0,1\}$. We here introduce a notion of normalization of signed bit fractions that guarantees such a number has magnitude in the range $(\frac{1}{4},1)$, where verification of normalization is determined by the leading three signed bits.

The signed bit string with radix point after b_0, $p = b_0.b_1b_2 \ldots b_k, b_i = \bar{1}, 0, 1$ for all i, of length $k+1$ is termed a signed bit fraction if and only if

$$-1 < \sum_{i=0}^{k} b_i 2^{-i} < 1.$$

A signed bit fraction is in complement form whenever $b_0 \neq 0$. b_0 is termed the complement bit, and when it is nonzero, its sign must be opposite to that of the sign of the *necessarily* nonzero value of the remaining portion of the number. The complement bit may thus be interpreted as a magnitude complementing bit in the sense that the absolute value of $b_0.b_1b_2 \ldots b_k$ is $1 - |x|$, where

$$x = \sum_{i=1}^{k} b_i 2^{-i} \neq 0.$$

The signed bit fraction $b_0.b_1b_2\ldots b_k$ is termed normalized whenever exactly one of b_0 and b_1 is nonzero, where also $b_2 \neq -b_1$, when $b_0 = 0$.

Thus we may define both standard and complement normalized forms.

Standard form: $b_0 = 0,\ b_1 \neq 0$ and $b_2 \neq -b_1$.
Complement form: $b_0 \neq 0,\ b_1 = 0$.

A signed bit fraction that is not normalized is termed unnormalized.

Observation 1: A normalized signed bit fraction always is in the range ($\frac{1}{4}$,1) or (-1,-$\frac{1}{4}$). An unnormalized signed bit fraction with $b_0 = 0$ must be in the range (-$\frac{1}{2}$, $\frac{1}{2}$).

Given the unnormalized signed bit fraction $b_0.b_1b_2\ldots b_k$, the complement bit may be forced to zero by the following decomp function.

$$\text{decomp}(p) = \begin{cases} 0.b_0b_2b_3\ldots b_k, & \text{if } b_0 = -b_1 \neq 0, \\ p, & \text{if } b_0 = 0. \end{cases}$$

Note we cannot have $b_0 = b_1 \neq 0$ since $-1 < p < 1$ for signed bit fraction. Note further that the decomp function does not change the value of p. The result may or may not be normalized.

The operation termed simplifying ("absorbing") shift (simshift) is defined for every noncomplemented unnormalized signed bit number $p = 0.b_1b_2\ldots b_k$ to yield a signed k bit fraction given by

$$\text{simshift}\,(p) = \begin{cases} 0.b_1b_3b_4\ldots b_k, & \text{if } b_1 \neq 0 \wedge b_1 = -b_2 \\ 0.b_2b_3b_4\ldots b_k, & \text{if } b_1 = 0. \end{cases}$$

Observation 2:

1. simshift(p) has twice the value of p,
2. simshift never creates a complemented signed bit fraction,
3. simshift(p) has length one bit less than p.

Normalization, a unary operation, defined for all nonzero signed bit fractions can be stated as follows:

Normalization Algorithm:

If p not normalized do p = decomp(p);
While p not normalized do p = simshift(p);

Lemma: Given a signed bit fraction $p \neq 0$ of length $k+1$,

$$p = \sum_{i=0}^{k} b_i 2^{-i} \neq 0,\ b_i = -1, 0, 1 \text{ for all } i,$$

then the normalization of p will be achieved after at most $k-1$ simshift operations, and this bound is best possible.

To implement the Euclidean algorithm for two signed nonzero numbers, the step of selectively determining their difference when similarly signed or their sum when oppositely signed is aided by introducing the following diff operation:

$$a \text{ diff } b = \begin{cases} a - b \text{ for } ab \geq 0, \\ a + b \text{ for } ab < 0. \end{cases}$$

The diff operation always returns a magnitude equal to the difference of the original two magnitudes.

Observation 3: For the normalized signed bit numbers a, b, the range of a diff b will be (-$\frac{3}{4}$, $\frac{3}{4}$).

It is useful to contrast standard binary normalization with redundant binary normalization:

Standard binary representation: The range of a normalized binary fraction is [$\frac{1}{2}$,1), (-1, -$\frac{1}{2}$]. The difference of two positive normalized binary fractions is never normalized, and when nonzero can always be normalized by one or more left shift operations.

Redundant binary representation: The range of a normalized signed bit fraction is ($\frac{1}{4}$, 1), (-1, -$\frac{1}{4}$). The result of the diff operation of two normalized signed bit fractions when nonzero can always be normalized by zero or more simshift operations. Note, the difference of two normalized signed bit fractions might be normalized.

4 Digit Selection

The digit selection process refers to a selection of one of the terms p diff q, p diff $2q$, or $2p$ diff q, as the new value of p. Specifically, after p and q are

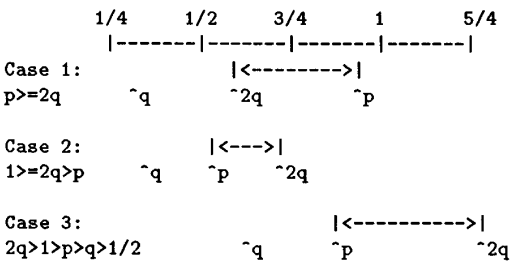

Figure 1: Examples of diff operations

normalized, the result of the p diff q operation is previewed (by look ahead) to check if it would be normalized or not. The occurrence of a normalized result would guarantee the resulting magnitude to be greater than $\frac{1}{4}$, in which case, depending on the signs of p, q, and p diff q, one of the terms ($2p$ diff q) or (p diff $2q$) is then chosen as the new value of p. With this selection, the value of p can be guaranteed to be less than $\frac{1}{2}$. This is diagrammatically shown above. Without loss of generality consider the case where p and q are positive, and $1 > p > q > \frac{1}{4}$. In each of three cases, given $p - q > \frac{1}{4}$, the position of p, q and the difference between $2q$ and p are illustrated with $|2q - p| < \frac{1}{2}$ in all cases (see Figure 1).

Lemma: Given $|p \text{ diff } q| > \frac{1}{4}$, where p, q are normalized signed bit fractions, then

$$|p \text{ diff } 2q| < \tfrac{1}{2} \text{ if } |p| > |q|,$$
$$|2p \text{ diff } q| < \tfrac{1}{2} \text{ if } |p| < |q|.$$

Lemma: Given $\frac{1}{4} < |p|, |q| < 1$,

$$\text{Min}(|p \text{ diff } q|,\ |p \text{ diff } 2q|,\ |2p \text{ diff } q|) < \frac{1}{2}.$$

Corollary:

After the selection of a diff operation term such that the magnitude of the result is less than $\frac{1}{2}$, it will always be possible to normalize the result by a process including at least one simshift operation.

Note that if a complement condition initially exists, it will be possible to first perform the decomp function. Note further that the imposition of at least one simshift can result in the b_0 bit becoming non zero. We are assured this result corresponds to a complement form signed bit fraction as the initial result was guaranteed to be of absolute value less than $\frac{1}{2}$ before the simshift operation.

This digit selection is based on the result of the operation (p diff q), specifically on the state of the result – whether it was normalized or not normalized. This in turn suggests that it is necessary and sufficient to know at most the first five positions of p and q (positions 0,1,2,3,4) if an alternative table lookup were used, since the carry propagation in redundant binary addition is limited to two places. Digit selection can be obtained either from a table or by implementing in hardware, simple combinational logic to perform a diff operation on the first five bits, and checking for normalization of the result.

Algorithm RBEA (Redundant Binary Euclidean Algorithm)

Given initializations p,q of the registers P,Q such that (p,q) $\neq$ (0,0), this algorithm terminates with the gcd value right justified in the P register. U_P and U_Q are auxiliary registers initialized to 1 which identify the unit-position of each argument.

```
While P and Q not normalized do
   Simshift P, Q and leftshift UP, UQ;
LoopA
   While Q not normalized do
      Simshift Q and leftshift UQ;
      Exit LoopA if UQ overflows;
   LoopB
      While P not normalized do
         Exit LoopB if UP=UQ;
         Simshift P and leftshift UP;
      If (P diff Q) is normalized then
         If sign(P diff Q) ≠ sign(P) then
            If UP≠UQ then
               leftshift UP; P:=2P diff Q;
            Else exit LoopB;
         Else P:=P diff 2Q;
      Else P:=P diff Q;
      P:=decomp(P);
      Simshift P;
      If UP≠UQ then leftshift UP;
      Else leftshift UPand exit LoopB;
   End LoopB;
   Swap(P,Q); Swap(UP,UQ);
End LoopA;
While UP≠1 do rightshift P, UP.
```

Theorem 1: Given the signed bit integers p and q, the number of diff operations in the

execution of algorithm RBEA is at most the sum of the lengths of p and q.

Proof: In each major cycle ending with a swap operation, each diff operation is followed by one or more simshift operations on register P. The number of diffs is bounded above by the number of simshift operations. A simshift operation reduces the length of one of the arguments by one, and the algorithm terminates when $q = 0$. Since the number of simshifts is bounded by the total of the lengths of p and q, the number of diff operations is at most the sum of the lengths of p and q.

□

Since n-bit redundant binary addition is a constant time operation with O(n) 4-2 signed 1-bit adders, the gcd of two n-bit numbers can be determined by Algorithm RBEA in O(n) time with O(n) bit level processors.

Observation 4 The absolute value of the chosen term (p diff q), (p diff $2q$), ($2p$ diff q) in Algorithm RBEA can always be simshifted. This fact can be used to combine the simshift and diff operation thereby avoiding the cost of a simshift immediately following a diff operation.

Algorithm RBEA terminates with a number of diff operations (previously shown to be equivalent to add/sub operations) not exceeding $j + k$, where j and k are the lengths of the two operands. This is a guaranteed upper bound. This upper bound is achieved by a guaranteed progress of at least one bit after each diff operation. This phenomenon is made possible by the digit selection process. The algorithm in the form shown needs swap operations but these can be eliminated by using two symmetrical registers in hardware. In practice, simulation of the algorithm with various length arguments has shown that the number of diff operations per bit of input ('progress') approaches 0.41.

Acknowledgements

We wish to thank Søren P. Johansen for helpful comments and improvements in the statement of Algorithm RBEA.

References

[1] D.E.Atkins, "The Theory and Implementation of SRT Division," *Report No. 230*, Dept. of Computer Science, University of Illinois, June, 1967.

[2] A.Avizienis, "Signed-digit Number Representation for Fast Parallel Arithmetic," *IRE Transaction in Electronic Computing*, Vol. 10, pp. 389-400, 1961.

[3] R.P.Brent, H.T.Kung, "A Systolic Algorithm for GCD Computation," *Proc. 7th IEEE Symp. on Comp. Arith.*, pp 118-125, 1985

[4] M.D.Ercegovac, "A Higher-Radix Division with Simple Selection of Quotient Digits," *Proceedings of Sixth IEEE Symposium on Computer Arithmetic*, pp 94-98, June, 1983.

[5] Euclid, *Proposition 1 I& 2 of Elements*, Book 7, appx.300BC.

[6] M.J.Foster and H.T.Kung, "The Design of Special Purpose VLSI Chips," *IEEE Computer*, Vol.13, pp.26-40, Jan. 1980.

[7] C.C.Freiman, "Statistical Analysis of Certain Binary Division Algorithm," *Proc. IRE, Vol.49*, pp 91-103, 1961.

[8] G.H.Hardy and E.M.Wright, *An Introduction to the Theory of Numbers*, 4th ed., Oxford, England: Clareton Press, 1959.

[9] Y. Harata et al., "High-Speed Multiplier LSI Using a Redundant Binary Adder Tree," *International Conference on Computer Design*, 1984.

[10] D.E.Knuth , *The Art of Computer Programming: Vol II,Seminumerical Algorithms*, (Section 4.5), Addison-Wesley Publishing Company, 1981.

[11] P.Kornerup and D.W.Matula, "Finite Precision Rational Arithmetic: An Arithmetic Unit," *IEEE Transactions on Computers*, Vol C-32, No.4, pp. 378-388, April 1983.

[12] S. Kuninobu et at., "Design of High-Speed Multiplier and Divider Using Redundant Binary Representation," *Proc. of the 8th ISCA*, pp 80-86, May 1987.

[13] O.L. MacSorley, "High-speed Arithmetic in Binary Computer," *Proc. IRE*, Vol. 49, no. 1, pp. 67-91, Jan. 1961.

[14] G.Metze, "A Class of Binary Divisions Yielding Minimally Represented Quotients," *IRE Transactions on Electronic Computers*, Vol. EC-10, December 1962.

[15] S. Parikh, *An Architecture For a Rational Arithmetic Unit,* Ph.D. dissertation, Southern Methodist University, December 1988.

[16] G.W.Reitwiesner, "Binary Arithmetic", *Advances in Computers*, Vol. 1, F.L.Alt, Ed. New York: Academic, 1960.

[17] J.E.Robertson, "A New Class of Digital Division Methods," *IRE Trans. El. Comp.*, Vol EC-7, no.3, pp.218-222, Sept. 1958.

[18] N.R.Scott, *Computer Number Systems and Arithmetic,* Prentice-Hall Inc., NJ, 1985, Section 7.11.

[19] B. Shirazi et al., "RBCD: A Redundant Binary-coded Decimal Adder," *IEE Proceedings - Computers and Digital Techniques*, Vol. 136, No.2, pp.156-160, 1989.

[20] G.S.Taylor, "Radix 16 SRT Dividers with Overlapped Quotient Selection Stages," *Proceedings of 7th Symposium on Computer Arithmetic*, 1985.

[21] K.D.Tocher, "Techniques of Multiplication and Division for Automatic Binary Computers," *Quart. J. Mech. Appl. Math.*, Vol. 11, pt.3, pp. 364-384, 1958.

[22] N. Takagi et al., "High-Speed VLSI Multiplication Algorithm With a Redundant Binary Addition Tree," *IEEE Trans. on Computers*, Vol. C-34, No.9, pp. 789-796, September 1985.