INSIDE EUCLID’S ALGORITHM

WILLARD L. EASTMAN†

Abstract. The polynomial version of Euclid’s algorithm is expanded to remove the inherent polynomial division. The expanded algorithm exhibits a two loop structure. The choice of which loop to execute at a given iteration depends on whether the iteration completes, or does not complete, a polynomial division. It is shown that one of the loops can be deleted, producing a clean version of the algorithm suitable for implementation in VLSI. The new version of Euclid’s algorithm is computationally equivalent to the standard long division version, but is more efficient in terms of hardware. Processing cells are presented for a two-dimensional systolic array architecture capable, with pipelining, of computing polynomial gcd’s in constant time. The new version of Euclid’s algorithm bears a strong resemblance to the Berlekamp-Massey algorithm.

Key words. Euclid’s algorithm, polynomial division, greatest common divisor, VLSI implementation

1. Introduction. In Book VII, Proposition 2 of his Elements [1], Euclid gave his famous algorithm for finding the greatest common divisor \( \gcd(s, t) \) of two integers \( s \) and \( t \). Euclid’s algorithm can be immediately adapted to find the greatest common divisor \( \gcd(f(x), g(x)) \) of two polynomials \( f(x) \) and \( g(x) \) over a field \( F \), with \( \deg(f(x)) \geq \deg(g(x)) \). In the extended version, the algorithm also produces a pair of polynomials \( a(x) \) and \( b(x) \) satisfying

\[
\gcd(f(x), g(x)) = a(x)f(x) + b(x)g(x).
\]

Many applications for the polynomial version of Euclid’s algorithm have appeared in recent years. Sugiyama et al. [2] used Euclid’s algorithm to solve the key equation in decoding Goppa codes. Mills [3] developed a continued fraction algorithm equivalent to Euclid’s algorithm for finding the linear recurrence of lowest degree satisfied by a given sequence. Welch and Scholtz [4] studied the relationship between Mills’ algorithm and the Berlekamp-Massey algorithm [5–6]. McEliece and Shearer [7] extended the work of Sugiyama et al. and gave an application to finding Padé approximants. Brent et al. [8] applied Euclid’s algorithm to the inversion of Toeplitz matrices. Sugiyama [9] extended his earlier work further to the solution of Wiener-Hopf equations. This work implies that an extended version of Euclid’s algorithm can be used to solve Toeplitz or Hankel systems of equations (with arbitrary right-hand-sides). A number of important signal processing applications require the solution of Toeplitz systems of equations. Finally, Shao et al. [10] used Euclid’s algorithm in their design of a decoder for deep space telemetry.

In view of the large number of important applications for the computation of polynomial gcd’s, versions of the algorithm especially suited for very large scale integrated (VLSI) circuit implementation are of interest. Brent and Kung [11]...
have given a linear array design based on the conventional long division version of Euclid’s algorithm, capable of computing polynomial gcd’s in time proportional to the sum of the degrees of the polynomials. In section 4 we present cells for a two-dimensional systolic array capable of computing a sequence of polynomial gcd’s in constant time. This design is based upon a modified version of Euclid’s algorithm developed in section 3. The modified version is computationally equivalent to the standard long division version of the algorithm, but realizes a significant saving in hardware. This modified version could also be used to improve the design of Brent and Kung [11], as well as the Reed-Solomon decoder design of Shao et al. [10], which is based on [11].

For brevity, we shall work with the original version of Euclid’s algorithm rather than with the extended version which obtains \( a(x) \) and \( b(x) \). Euclid’s algorithm and its properties are well known. See, for example, Knuth [12]. At the \( j \)th iteration of Euclid’s algorithm a new remainder polynomial \( r^{(j)}(x) \) is defined by

\[
r^{(j)}(x) = r^{(j-2)}(x) - \left\lfloor \frac{r^{(j-2)}(x)}{r^{(j-1)}(x)} \right\rfloor r^{(j-1)}(x),
\]

where \( \left\lfloor \frac{s(x)}{t(x)} \right\rfloor \) denotes the quotient polynomial obtained when \( s(x) \) is divided by \( t(x) \). The algorithm is initialized by setting

\[
r^{(-1)}(x) \leftarrow f(x) \\
r^{(0)}(x) \leftarrow g(x)
\]

and terminates at the first \( j \) for which \( r^{(j)}(x) = 0 \), at which time

\[
gcd(f(x), g(x)) = r^{(j-1)}(x).
\]

We state the algorithm in the form of a program using Iverson’s notation [13] and employing four polynomials: a quotient polynomial \( q(x) \), an ‘old’ remainder polynomial \( r^O(x) \), a ‘new’ remainder polynomial \( r^N(x) \), and a ‘temporary’ remainder polynomial \( r^T(x) \). Program 1 is divided into two sections or boxes: an initialization box consisting of two statements which initialize the ‘old’ remainder polynomial \( r^O(x) \) by \( f(x) \) and the ‘new’ remainder polynomial \( r^N(x) \) by \( g(x) \), and a recursion box consisting of four specification statements followed by a comparison of \( r^N(x) \) against 0. The program terminates when the comparison is satisfied by equality; otherwise the entire loop contained in the box is repeated. Thus, an iteration consists of the execution one time of all statements in the recursion box. At the beginning of the \( j \)th iteration, \( r^O(x) \) is \( r^{j-2}(x) \), the remainder polynomial obtained at the \( j-2 \)nd iteration, and \( r^N(x) \) is \( r^{j-1}(x) \), the remainder obtained at the \( j-1 \)st iteration; at the conclusion of the \( j \)th iteration, \( r^O(x) \) is \( r^{j-1}(x) \), and \( r^N(x) \) is \( r^j(x) \). The initialization box thus defines \( r^{-1}(x) \) and \( r^0(x) \). At termination the gcd \( \gcd(f(x), g(x)) \) is given by \( \gamma r^O(x) \) for the field element \( \gamma \) which renders the gcd monic.

Our objective is parallel scalar computation in an array of computing elements involving only local communication among elements. Such an array is sometimes