International Journal of Electrical and Electronics Engineering Research (IJEEER)
ISSN 2250-155X
Vol. 3, Issue 4, Oct 2013, 59-66

APPROACH TO DESIGN GREATEST COMMON DIVISOR CIRCUITS BASED ON METHODOLOGICAL ANALYSIS AND VALUATE MOST EFFICIENT COMPUTATIONAL CIRCUIT

DARSHANA UPADHYAY, JAHANVI KOLTE & KRITIKA JALAN
Institute of Technology, Nirma University, Chharodi, Ahmedabad, Gujarat, India

ABSTRACT

In elementary arithmetic, the greatest common divisor is used to simplify expressions by reducing the size of numbers involved. Greatest common divisor (GCD) of given numbers is the largest number that divides all of the given numbers without leaving any remainder. This paper presents the hardware simulation of different methods employed to compute Greatest common divisor of any two numbers (8-bit binary) in simulator. For this purpose, four different methods were worked out, of which, three were dynamic implementations namely, Euclid's method, Divisibility Check Method, Dynamic modulo and one was static implementation, static modulo method. These algorithms were then compared for their space & time complexity. For Space complexity, number of different components, like basic gates, memory units, plexers, arithmetic operation units, etc. used were compared and for time complexity, clock pulses required were measured for a few set of numbers.

KEYWORDS: Dynamic Modulo Method, Euclid’s Method, Divisibility Check Method, Static Modulo Method, Time Complexity Analysis , Space Complexity Analysis

INTRODUCTION

Greatest Common Divisor (GCD) of two non-negative numbers is the largest integer that divides both the numbers leaving 0 remainders.

$$GCD\ (a, b) = max\ \{k \in \mathbf{Z} : k|a \wedge k|b\}.$$

GCD computation is a central task in computer algebra, in particular when computing over rational numbers or over modular integers [5]. Greatest Common Divisor can be found using many methods. In this paper, we present four such methods along with their hardware implementation and analysis of their Time Complexity and Space complexity. The four methods are Method of Equating 0 Remainders, Static Modulo Method, Dynamic Modulo Method and Euclid's Method. The Method of Equating 0 Remainders divides both the numbers with a number starting from the smaller number to 1. When both the remainders are 0, clock pulse stops and that is our GCD value. The Static Modulo Method finds the remainder of two given numbers. Later the divisor and the remainder in the first iteration become the dividend and divisor in second iteration. This continues until remainder becomes zero. The Dynamic Modulo Method differs from Static Modulo Method in the way that it uses registers. In Static Modulo Method, the no. of times the loop runs say l, so l no. of modules must be implemented in this method unlike the dynamic modulo method where you store the output of 1 module in register and use it later. The Euclidean Algorithm is regarded as the grandfather of all algorithms in number theory today because it is the oldest nontrivial algorithm that has survived to the present day [1]. In this method, the smaller number is subtracted from the greater number and the result is stored in the greater number (before subtraction). This continues till both the numbers become equal. That number is the GCD of given numbers. The Euclidean algorithm solves this problem in time quadratic in size n of inputs [5].

The purpose of this paper is to show various hardware implementation of a circuit to calculate GCD. Different digital components have been used to employ the above algorithms. Comparator, Divider and Multiplexer have been used to implement the circuits. Some circuits also use counter and subtractors.

HARDWARE SIMULATION OF METHODS

Under this section, the hardware implementation of all four methods are shown and based on this, the conclusion is reached.

Divisibility Check Method

The Divisibility Check Method divides both the numbers with a number starting from the smaller number to 1. When both the remainders are 0, clock pulse stops and that is our GCD value.

Algorithm

Input: Two Unsigned Numbers a and b.

Output: Greatest Common Divisor of input.

```
Counter = 0; GCD =0; small=0; big=0; remS=0; remB=0;
    if (a< b)
      then
            small = a; big=b;
else
            small = b; big=a;
end
Counter = small;
repeat:
    remB = big % Counter;
    remS = small % Counter;
  if remB == remS
      then
      GCD = Counter;
else
      Counter--;
        until Counter>0;
end
```

Circuit

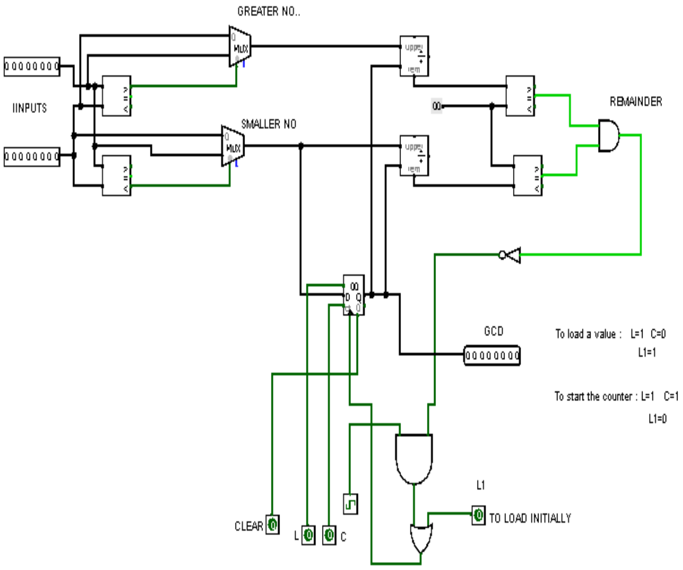

Figure 1: Divisibility Check Method

Static Modulo Method

The Static Modulo Method finds the remainder of two given numbers. Later, the divisor and the remainder in the first iteration become the dividend and divisor in second iteration respectively. This continues until remainder becomes zero. The Static Modulo Method differs from dynamic method in the way that it does not employ any registers. The number of times the loop runs (say n), n modules has to be implemented here unlike dynamic modulo method where the output of one module is stored in register and used later.

Algorithm

Input: Two Unsigned Numbers a and b.

Output: Greatest Common Divisor of input.

GCD =0; small=0; big=0; rem=0;

if (*a*< *b*)

then

small = a; *big=b*;

else

small = b; *big=a*;

end

here:

rem = *big* % *small*;

if (*rem* == 0)

then

GCD = *small*;

else

big = *small*;

small = *rem*;

goto *here*;

end

Note: This method has a restriction on the numbers of input. They are in accordance with the number of modules in the circuit.

Circuit

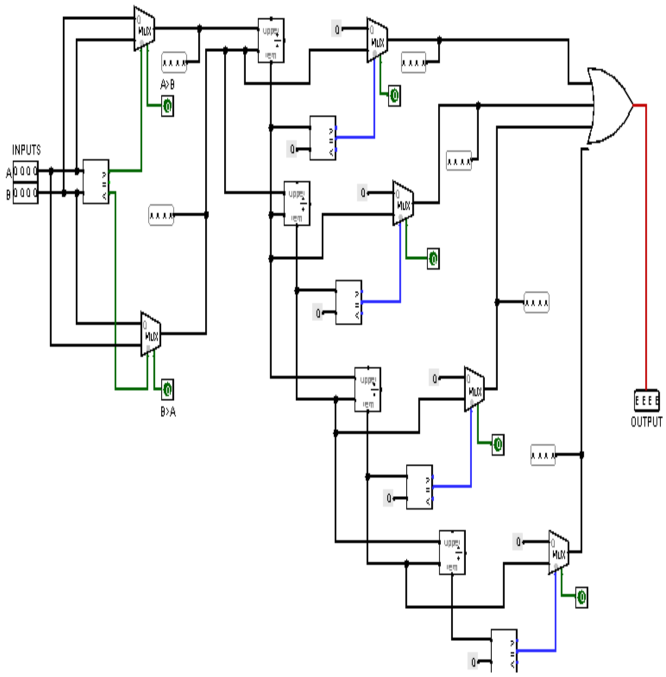

Figure 2: Static Modulo Method

Dynamic Modulo Method

The Dynamic Modulo Method finds the remainder of two given numbers. Later the divisor and the remainder in the first iteration become the dividend and divisor in second iteration. This continues until remainder becomes zero. The number of hardware components used in this method is considerably reduced as compared to Static Modulo Method.

Algorithm

Input: Two Unsigned Numbers a and b.

Output: Greatest Common Divisor of input.

GCD =0; small=0; big=0; rem=0;

if (*a*< *b*)

then

small = a; *big=b*;

else

small = b; *big=a*;

end

here:

rem = *big* % *small*;

if (*rem* == 0)

then

GCD = *small*;

else

big = *small*;

small = *rem*;

goto *here*;

end

Note: This method has same algorithm as previous one. Only, this has no restriction as it is dynamic implementation.

Circuit

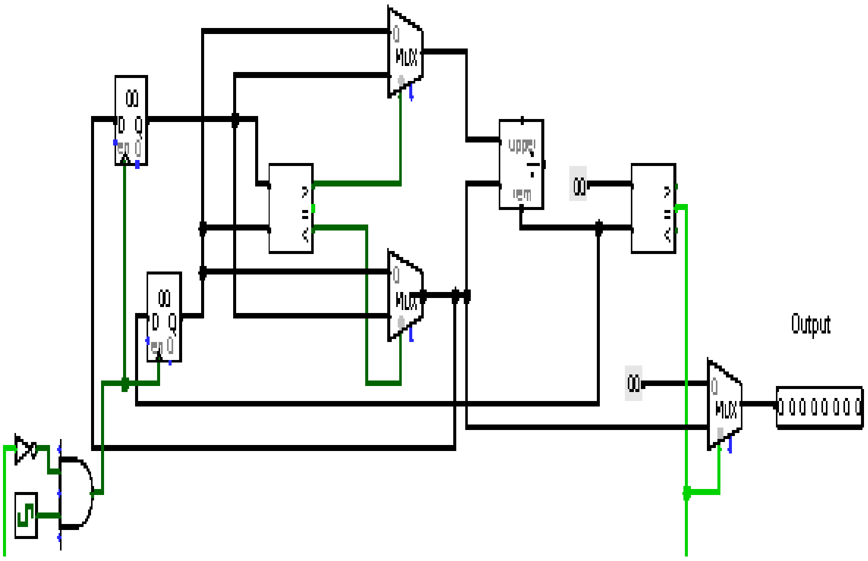

Figure 3: Dynamic Modulo Method

Euclid's Method

In this method, the smaller number is subtracted from the greater number and the result is stored in the greater number (before subtraction). This continues till both the numbers become equal. This equal number is the GCD of given numbers. The Euclidean Algorithm is regarded as the grandfather of all algorithms in number theory

Algorithm

Input: Two Unsigned Numbers a and b.

Output: Greatest Common Divisor of input.

GCD =0; small=0; big=0; rem=0;

here:

if (*a*< *b*)

then

small = a; *big=b*;

else

small = b; *big=a*;

end

big = big - small;

a = *big* ; b= *small*;

if (*big == small*)

then

GCD = *small*;

else

goto *here*;

end

Circuit

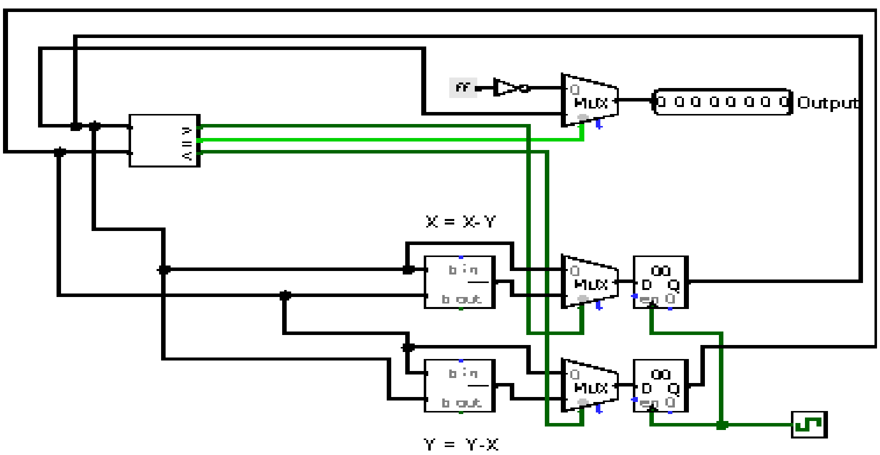

Figure 4: Euclid's Method

ANALYSIS

Here, the time and space complexity of each method has been computed as shown below.

Time Complexity

This is computed by counting the number of clock pulses required to get the GCD output by each method for a given set of values. The values are taken in four sets. Set one with both inputs as small numbers. Set two with one number small and the other big. Set three with both numbers medium. Set four with both numbers big. Thus in this way, all types of cases are covered. (Big and small are considered within the range of 1 to 100)

Table 1: Time Complexity Comparison Chart

Numbers/Algorithms	Euclid's Method (Clock Pulse)	Dynamic Modulo (Clock Pulse)	Equating 0 Remainders (Clock Pulse)
5,7	4	2	4
9,10	8	1	7
12,16	3	1	8
4,65	19	1	3
11,77	6	1	10
29,42	8	3	28
50,52	25	1	48
96,99	[1]31	1	93

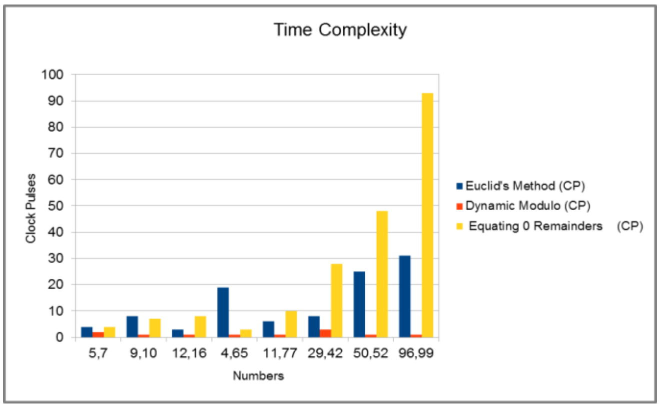

Figure 5: Time Complexity Comparison Chart

The analysis shows us that, among dynamic methods, Method of Equating 0 Remainders is less complex when it comes to smaller numbers and its complexity gradually increases as the value of the smaller number rises.(E.g. 42, 29 and 96, 99).Euclid's algorithm is less complex or works comparatively better when we have higher order numbers. (E.g. 96 and 99).The dynamic modulo method requires very less clock pulses to compute GCD in all the three sets of experimented values. The static modulo method does not consist of any memory unit in its implementation.Thus, no clock pulses are required to compute GCD. It means that it also has less time complexity. But it’s hardware complexity is very high.

Space Complexity

To compute GCD calculation methods in terms of space complexity, the number of hardware components required to build the circuit are taken into consideration.

Table 2: Space Complexity Comparison Chart

Hardware Components	Equating Zero Remainders (8 Bits)	Euclid’s Method (8 Bits)	Dynamic Modulo (8 Bits)	Static Modulo (4 Bits)
Memory Unit	1 Counter	2 Registers	2 Registers	-

[1] TableI Time Complexity Comparison Chart shows clock pulses required by various algorithms to compute specific problems of GCD.

Table 2: Contd.,

Plexers	2 Mux	3 Mux	3 Mux	6 Mux
Arithmetic Units	4 Comparator 2 Divider	1 Comparartor 2 Subtractor	2 Comparator 3 Divider	4 Comparator 4 Divider
Basic Gates	1 NOT 2 AND 1 OR	-	1 NOT 1 AND	1 OR

From the above analysis, it is clear that the number of hardware components requiredimplement Euclid's Method has the least space complexity.

PERFORMANCE EVALUATION

All the circuits are formulated using Logisim Simulator 2.7.1 .

CONCLUSIONS

From the analysis we can conclude that among all the four methods, Euclid's method has least number of components i.e. it is the best suited method in terms of space complexity as it may be used to generate almost all the most important traditional musical rhythms used in different cultures throughout the world. It is a key element of the RSA algorithm, a public-key encryption method widely used in electronic commerce. It is a basic tool for proving theorems in modern number theory, such as Lagrange's four-square theorem and the fundamental theorem of arithmetic. But Dynamic Modulo Method is the best method in terms of time complexity as the number of clock pulses required to obtain the answer is considerably reduced as it can be used to get a very fast, practical and deterministic algorithm for triangularizing integer matrices, Pad approximation, iterative solution of linear systems, Eigen value problems, orthogonal polynomials, coding theory.

REFERENCES

1. B. Vallee, 2003, Dynamical analysis of a class of Euclidean Algorithms., *The Computer Science*, 297(1-3): pp. 447-486.

2. Haroon Altarawneh, 2011, A Comparison of Several Greatest Common Divisor (GCD) Algorithms, *International Journal of Computer Applications* (0975 - 8887), Volume 26, No.5

3. Jeffrey Shallit and Jonathan Sorenson, 1994, Analysis Of Left-Shift GCD Binary Algorithm, *J.Symbolic Computation*, pp. 473-486

4. Jonanthan Sorenson, 1994, Two fast GCD Algorithms, *Journal Of Algorithms* 16, pp. 110-11

5. P. Emeliyanenko, (2010a) A complete modular resultant algorithm targeted for realization on graphics hardware, *PASCO 10*, pp. 35-43, New York, NY, USA. ACM ;(2010b) Accelerating Symbolic Computations on NVIDIA Fermi, *GTC 10*, Poster presentation; (2010c)Modular Resultant Algorithm for Graphics Processors, *ICA3PP 10*, pp. 427-440, Berlin, Heidelberg, Springer-Verlag.[2]

6. T. Jebelean, 2010, Comparing Several GCD Algorithms " Che Wun Chiou, Fu Hua Chou , Yun-Chi Yeh , Speeding up Euclid's GCD algorithm with no magnitude comparisons , *International Journal of Information and Computer Security table of contents archive* Volume 4 Issue 1.

[2] TableII Space Complexity Comparison Chart shows hardware components required by circuits of various algorithms to compute GCD.