FPGA Implementation of Variable Precision Euclid’s GCD Algorithm

Qasem Abu Al-Haija¹, Sharifah Mumtazah Syed Ahmad², Islam Alfarran³

¹College of Engineering, King Faisal University, Hufof 31982, P.O. 380, Saudi Arabia.
²Faculty of Engineering, Universiti Putra Malaysia, 43400, Serdang, Selangor, Malaysia
³College of Applied Studies and Community Service, King Faisal University, Hufof, Saudi Arabia

Abstract: Introduction: Euclid's algorithm is well-known for its efficiency and simple iterative to compute the greatest common divisor (GCD) of two non-negative integers. It contributes to almost all public key cryptographic algorithms over a finite field of arithmetic. This, in turn, has led to increased research in this domain, particularly with the aim of improving the performance throughput for many GCD-based applications. Methodology: In this paper, we implement a fast GCD coprocessor based on Euclid's method with variable precisions (32-bit to 1024-bit). The proposed implementation was benchmarked using seven field programmable gate arrays (FPGA) chip families (i.e., one Altera chip and six Xilinx chips) and reported on four cost complexity factors: the maximum frequency, the total delay values, the hardware utilization and the total FPGA thermal power dissipation. Results: The results demonstrated that the XC7VH290T-2-HCG1155 and XC7K70T-2-FBG676 devices recorded the best maximum frequencies of 243.934 MHz down to 39.94 MHz for 32-bits with 1024-bit precisions, respectively. Additionally, it was found that the implementation with different precisions has utilized minimal resources of the target device, i.e., a maximum of 2% and 4% of device registers and look-up tables (LUT's). Conclusions: These results imply that the design area is scalable and can be easily increased or embedded with many other design applications. Finally, comparisons with previous designs/implementations illustrate that the proposed coprocessor implementation is faster than many reported state-of-the-art solutions. This paper is an extended version of our conference paper [1].

Keywords: Digital arithmetic, FPGA, Integrated circuit synthesis, Euclid's algorithm, GCD.

1. Introduction

Over the past decade, a rapid advancement in digital hardware design has led to a major revolution that took the place of the once dominant digital design era without using traditional methods of logic design. Millions of logic gates and tens of thousands of flip-flops can co-exist in a single design technology tool via field programmable gate arrays (FPGAs) [2]. FPGA devices contain a matrix of configurable logic blocks (CLBs) connected via a programmable network that can be utilized by writing a software program using hardware description languages (HDLs) such as VHDL programming [3] and design synthesize [4]. The efficient design-based FPGA technology has recently emerged in many fields and applications, such as high-performance computing, networking, security and cryptography, as in [5, 6]; fault tolerance applications, as in [7]; and many other regular and irregular applications.
Cryptoprocessors design [8] involves the use of different number theory algorithms at the upper level of the design, which can be built based on the digital arithmetic [9] that can be easily described by HDLs for FPGA verification purposes. Number theory [8, 10] was largely separated from other fields of mathematics since it is topically related to elementary arithmetic. Its applications have rapidly increased in recent years for areas such as coding theory, cryptography and statistical mechanics. The “Euclidean algorithm” [8] and “Sieve of Eratosthenes” [10] are both recent candidates to implement. This paper focuses on the iterative Euclid's algorithm to compute the GCD of two non-negative integer numbers. Its efficiency and simplicity make it attractive to many applications, especially those that are related to public key cryptography using finite field arithmetic operations. An RSA (Rivest, Shamir, and Adleman) cryptosystem is a good example of a GCD application that uses crypto-algorithms [5], and the trusted platform module (TPM) uses RSA as a building block [6]. Another example of a GCD application is multiplicative inverse calculation. The contributions of this paper can be summarized as follows:

- A state-of-the-art review of various design techniques of GCD algorithm (hardware, software or hybrid).
- Details on the hardware implementation of a variable data path GCD coprocessor using efficient modules, including a schematic diagram, a finite state machine, and a full RTL diagram for a 32-bit GCD (Appendix).
- Comparative performance evaluation of the FPGA implementation for GCD using seven different FPGA chip families.
- Discussion of the synthesize results related to the area of the design, the total delay of the design, minimum delay, maximum frequency and total FPGA thermal power dissipation complexities.
- Comparison of the proposed GCD implementation with many state-of-the-art works.

The remainder of this paper is organized as follows. Section 2 discusses the related works on GCD designs and implementations. Section 3 provides a brief background of GCD arithmetic along with a detailed numerical example for illustration purposes. Section 4 discusses the complete hardware implementation and specifications. Section 5 contains experimental results with their associated discussions, including performance measures, hardware utilization of the proposed implementation, FPGA total thermal power dissipation and a state-of-the-art benchmarking study. Finally, Section 6 concludes the paper.

2. Literature Review

Recently, many hardware/software solutions have attempted to address the efficient design of iterative number theory algorithms, such as the Euclidian algorithm. The most commonly used solutions include FPGA design and synthesis, hardwired microprogramming, and software-based simulations via high-level programming languages.
For instance, the FPGA design with its various chip families was the dominant method of implementing a high-speed GCD processor. Upadhyay and Patel [13] proposed a 4-bit hardware design for the Euclidean-calculated GCD using a narrative method with modular arithmetic based on subtraction (replacing the remainder by repeated subtraction) using basic digital components, such as multiplexers, comparators, registers, and a full subtractor. They concluded that the proposed design provides less complexity in terms of both hardware requirements and execution time. Additionally, Kohale and Jasutkar [14] studied the performance (area-speed) and power dissipation for the FPGA design of an 8-bit GCD processor using Euclid’s and Stein’s algorithms with Spartan 6 as the hardware chip technology. Their experimental results showed that Spartan 6 improved the power consumption by 42% and increased the performance speed over previous generation devices (i.e., Spartan 3). They also found that Stein’s algorithm has better results than Euclid’s algorithm with less power consumption and better performance. In their related work [15], they targeted the Xilinx Spartan-3 chip family via VHDL to develop an FPGA design for the GCD based on two computation methods: Euclid’s and Stein’s algorithms. Their experimental results were generated using Xilinx ISE 9.1i and showed that Euclid's GCD algorithm recorded a better performance with fewer slice registers and required bounded input/output blocks (IOB). It also recorded the minimal power consumption at 24 mW.

Moreover, Shah et al. [16] utilized the idea of reversible logic to break the conventional speed-power trade-off, which they claimed had a close match to quantum computing devices. To authenticate their research, various combinational and sequential circuits were implemented, such as an 8-bit GCD processor the use of reversible gates. Their FPGA design of the GCD processor recorded a maximum frequency of 456 MHz at an operand size of 8-bits using the Spartan-3 XC3S50 family. Furthermore, Willingham and Kale [17] proposed an asynchomatic system that uses Euclid’s algorithm to calculate the GCD of two integers that contain both repetition and decision to implement arbitrarily complex computational systems. They showed that, under typical conditions in a 0.35-μm process, a 16-bit implementation can perform a 24-cycle test vector in 2.067 μs with a power consumption of 3.257 nW. Boland et al. [18] applied a word-length optimization technique to implement every arithmetic operator throughout a custom FPGA-based accelerator via the IEEE-754 standard single or double precision arithmetic. They implemented the FPGA design of Euclid's GCD algorithm using Xilinx Coregen and obtained a maximum frequency range from 230-180 MHz as the number of fractional bits varies from 5 to 10.

Recently, the FDFM- (few DSP slices and few block memories)-based approach has been proposed and efficiently utilized into different FPGA design applications, such as the FDFM-based designs/implementations for Euclid's-computed GCD unit in [19 and 20]. Zhou et al. [19] implemented their processor core that executes Euclid's GCD algorithm using few DSP slices and few blocks of RAM in a single FPGA. This processor core (called the GCD processor core) has been built using the FDFM approach embedded in the Xilinx Virtex-7 family FPGA XC7VX485T-2. Their experimental results showed that the performance of this FPGA implementation using the 1280 GCD processor cores is 0.0904 μs per GCD computation for two 1024-bit integers, which is 3.8 times faster than the best GPU implementation and 316 times faster than a sequential implementation on an Intel Xeon CPU. In a related work, the same authors in [20] used the same implementation environment and adopted 1408 processors working in parallel and independently compute the GCD. These authors showed that this new core runs at 0.057 μs per GCD computation of two 1024-bit RSA moduli, which
is 6.0 times faster than the best GPU implementation and 500 times faster than a sequential implementation on an Intel Xeon CPU.

The use of BIST (built in self-test) technology as an additional microchip controller had an impact on the very large scale integrated (VLSI) design according to Devi et al. [21], who focused on the dramatic impact of the VLSI as it increases the complexity of the circuits. Per the Altera corporation [22], one solution can be used to avoid the overhead of the VLSI design by adding an extra IC chip with a self-test ability. Thus, they proposed a VHDL implementation of the GCD processor with the BIST capability using the Xilinx Spartan-3 chip family. Then, they compared the area overhead for both schemes (with/without BIST). The experimental results showed that the BIST implementation for the GCD increased the area overhead but eliminated the need to acquire high-end testers. Again, Kohale and Jasuktar [23] proposed an FPGA design with the BIST controller of the arithmetic logic unit (ALU) to calculate the 8-bit GCD of two positive integers using Euclid’s and Stein’s algorithms. They compared the design using various Xilinx Families (with and without the BIST technique). The selection of the Xilinx Family depends on the lowest power consumption of the ALU. Thus, they concluded that the Spartan 3E FPGA family was preferable for the GCD design with the BIST feature, as it recorded the lowest power dissipation number of 34 mw. In some related works, the authors of [24] applied BIST technology as they proposed new 4-bit and 8-bit GCD processors based on the BIST controller using Euclid’s and Stein’s algorithms. They applied the proposed FPGA design to three Xilinx Spartan 6 target devices, namely the XC3S50, XC4VFX12, and XC6SLX4. Comparisons regarding the number of look-up tables (LUT’s) showed that the XC6SLX4 device was the most efficient device, as it registered the minimum required area of the design.

The software-based solutions are valid for specific design situations. In [25], Upadhyay et al. proposed an 8-bit hardware design GCD processor using four different algorithms, including Euclid’s method, the divisibility check method, the dynamic modulo method and the static modulo method. They simulated their work using Logisim Simulator 2.7.1 and compared the designs in terms of both space and time complexity. The conducted experiments of [25] showed that Euclid’s method was the best-suited method in terms of space, while the dynamic modulo method was the best method in terms of time complexity since the number of clock pulses was considerably reduced. Additionally, Hemmer et. al. [26] reported on several generic implementations for univariate polynomial GCD computations over the integers, particularly over algebraic extensions. They designed a new polynomial software package that became a part of the Cgal release 3.4. Regarding the GCD, their FPGA implementation of the GCD using the hybrid approach computed Euclid’s algorithm (approximately) in 1 m sec at a data-path size of 1024-bits. Furthermore, Ellerwe and his research group [27] described an environment to accelerate fault simulation by hardware emulation on FPGA digital circuits. The proposed approach allows the simulation speed to be increased by 40 to 500 times compared to the state-of-the-art software-based fault simulation. The study included the FPGA design of the 32-bit GCD as a benchmark, which recorded a maximum frequency of 25 MHz with and without fault dropping and 20 MHz with three-valued logic. Based on the experiments, it can be concluded that it is beneficial to use emulation for circuits that require large numbers of test vectors while using simple but flexible algorithmic test vector generating circuits (BIST).

Another noticeable method that has been recorded in the literature is the use of mixed solutions such as the conversion of the high-level to the HDL language. In [28], the authors presented an optimization technique of flow paths (a compiler for converting high-level stack-based languages (Java, C++, C#, and VB) to VHDL for use on an FPGA or application-specific integrated circuit
(ASIC), as new self-propagating flow paths that execute faster and are less resource-intensive. They conducted several comparisons. They synthesized their proposed Euclid’s GCD design for a Xilinx Spartan-6 XC6SLX75, which reported a maximum frequency of 200 MHz for the 32-bit size. Different from these previous works, the major contribution of our work focuses on efficient FPGA implementation and the synthesis of Euclid’s GCD using different datapath sizes and FPGA device technologies in terms of timing issues, such as the critical path delay and the maximum frequency of a digital GCD processor. In this paper, we implemented the proposed GCD using VHDL. In addition, a comparative synthesizing study will be presented for several implementation options using different FPGA devices in terms of delay and maximum frequency. The comparison with other existing designs and implementations showed that the proposed coprocessor implementation improved larger scale performance.

3. Euclid’s GCD algorithm-Revisited

The GCD of two numbers is the highest common divisor/factor to both numbers. Two numbers whose GCD is 1 are called co-prime or relatively prime. There are many algorithms that can be used to compute the GCD [8, 10]. Euclid's algorithm was chosen due to its proven efficiency and simplicity. It also arrives at the solution faster within a single cycle [10]. Euclid’s algorithm computes the GCD of two non-negative integers (at least one of which is non-zero). The well-ordering principle states that every non-empty set of positive integers has a smallest element. Assume that \( a \geq b > 0 \) for integers \( a \) and \( b \). To find the GCD of \((a, b)\), the division algorithm [9] tells us that:

\[
a = q_1b + r_1, \text{ where } 0 \leq r_1 < b.
\]

If \( r_1 = 0 \), then \( b \) divides \((a, b)\) = \( b \). If \( r_1 \neq 0 \), divide \( b \) by \( r_1 \) to produce integers \( q_2 \) and \( r_2 \), such that:

\[
b = q_2r_1 + r_2, \text{ where } 0 \leq r_2 < r_1.
\]

If \( r_2 = 0 \), we stop the process. Otherwise, we continue to get \( r_1 = q_3r_2 + r_3 \), where \( 0 \leq r_3 < r_2 \). This process continues until we get a zero remainder \( r_{\text{N+1}} \). We arrive at the following system of equations:

\[
a = q_1b + r_1, \text{ where } 0 < r_1 < b
\]

\[
b = q_2r_1 + r_2, \text{ where } 0 < r_2 < r_1
\]

\[
r_1 = q_3r_2 + r_3, \text{ where } 0 < r_3 < r_2
\]

\[
r_2 = q_4r_3 + r_4, \text{ where } 0 < r_4 < r_3
\]

\[
\vdots
\]

\[
r_{\text{N-2}} = q_{\text{N-1}}r_{\text{N-1}} + r_{\text{N}}, \text{ where } 0 < r_{\text{N}} < r_{\text{N-1}}
\]

\[
r_{\text{N-1}} = q_{\text{N}}r_{\text{N}} + 0 \Rightarrow r_{\text{N}} \text{ is GCD} (a, b).
\]

Additionally, selecting an appropriate FPGA kit depends on the application itself. There is a clear trade-off between the two dominant FPGA companies of Altera and Xilinx. The comparison of kits for the main and common features provided in each chip family can be found in [4]. For a better understanding and explanation of the implemented Euclid’s GCD algorithm, we give an illustration example as follows:

\[
\text{GCD} (1622650076, 984943661) = 1
\]

\[
1622650076 = 984943661 \times 1 + 637706415 \Rightarrow 984943661 = 637706415 \times 1 + 347237246
\]

\[
637706415 = 347237246 \times 1 + 290469169 \Rightarrow 347237246 = 290469169 \times 1 + 56768077
\]

\[
290469169 = 56768077 \times 5 + 6628784 \Rightarrow 56768077 = 6628784 \times 8 + 3737805
\]

\[
6628784 = 3737805 \times 1 + 2890979 \Rightarrow 3737805 = 2890979 \times 1 + 846826
\]

\[
2890979 = 846826 \times 3 + 350501 \Rightarrow 846826 = 350501 \times 2 + 145824
\]

\[
350501 = 145824 \times 2 + 58853 \Rightarrow 145824 = 58853 \times 2 + 28118
\]
58853 = 28118 * 2 + 2617 \rightarrow 28118 = 2617 * 10 + 1948
2617 = 1948 * 1 + 669 \rightarrow 1948 = 669 * 2 + 610
669 = 610 * 1 + 59 \rightarrow 610 = 59 * 10 + 20
59 = 20 * 2 + 19 \rightarrow 0 = 19 * 1 + 1
19 = 1 * 19 + 0

4. Implementation and Environment

To benchmark the proposed implementation, we have developed our implementation using VHDL as the hardware description language and tested the implementation code using seven different FPGA chips such as the Altera Cyclone IV (EP4CGX-22CF19C6) and the Xilinx Virtex-7 (XC7VH290T-2-HCG1155). Additionally, at the software phase, different programs were used to accomplish the work of this paper, including Altera Quartus II [22] as the full platform for the Altera kit, including the hardware synthesis; the ModelSim-Altera 10.1d simulation and verification [22]; the Xilinx ISE Design Suite version 14.2 [4] to synthesize the implementation using six Xilinx FPGA devices, in addition to the Altera (to benchmark and compare); and Maple Worksheets 17 for mathematical verification purposes [29]. Moreover, a high-performance multiprocessor platform has been used in the coding, simulation, verification, synthesis, and testing phases. The platform specifications are shown in Table 1. Furthermore, the implementation was synthesized for variable numbers of bit sizes (from 32 to 1024 bits).

Table 1. Simulation Platform Specifications.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor / OS</td>
<td>4th Gen. Intel-I7 Quad-Core [3.4 GHZ, 8 MB Shared Cache] / Win 8.1 64 bit</td>
</tr>
<tr>
<td>Memory / Hard Drive</td>
<td>16 GB DDR3 - 1600 MHz / 2 TB 7200 RPM SATA</td>
</tr>
<tr>
<td>Graphics/ Screen</td>
<td>2 GB AMD Radeon R7 240 [DVI, HDMI, &amp; DVI-VGA] / 23” LED Display</td>
</tr>
</tbody>
</table>

Figure 1. (a) The Euclidian GCD algorithm (b) The finite state diagram of the GCD
In this work, we have implemented the Euclidean algorithm (given in figure 1.a) to iteratively compute the GCD by performing the repeated modular multiplication method. The two-parameter-loop is repeatedly executed until the second input is greater than zero while exchanging the input registers. It will put the remainder in the second input register. Finally, the output will be ready after finishing the loop in the first input register. The finite state diagram of the proposed implementation is depicted in figure 1.b. It shows the state values along with the transition condition.

The top view of the GCD processor is illustrated in figure 2.a. The GCD processor has two (N-1 bits) numbers as input values, three control signals (reset, enable and a synchronized clock), N-1 bits number as an output, and an acknowledgement signal. The internal hardware architecture is illustrated in figure 2.b. The implementation consists of the following:

- Three main registers are used, including two registers to hold inputs and one to hold the output.
- The modular multiplication unit uses an interleaved algorithm for faster performance. The modular multiplication is needed to preserve the products less than the input operands.
- The subtraction unit is used for repeated reductions of the swapped operands.
- One equality comparator can be built only from NOR gates. It iteratively tests the output results.

We believe that our proposed implementation is efficient for several possible reasons. For instance, the use of Interleaved modular multiplication as a core of computing Equid’s GCD improves efficiency. Also, the use a maximum possible number of concurrent VHDL statements in implementing Euclid’s GCD algorithm also improves efficiency. Furthermore, the use of new FPGA chip technology offered better hardware utilization and enhanced performance.

5. Cost Factors Results and Discussion

The experimental data presented in this section were generated using both Altera Quartus II and Xilinx Synthesizer ISE tools. The target chip technologies were set to the Altera Cyclone IV (ep4cgx-
22cf19e6/ep4ce115 (f29c7), the Xilinx Virtex-7 (XC7VH290T-2-HCG1155), the Xilinx Virtex-5 (XC5V1X20T-2-FF323), the Xilinx Spartan-6 (XC7Z010-2-CLG400), the Xilinx Artix (70XC7A100T-2-CSG324), the Xilinx Kintex-7 (XC7K70T-2-FBG676), and the Xilinx Zynq (XC7Z010-2-CLG400). We applied them all to our VHDL code for the GCD-processor.

The bar chart in figure 3 compares the maximum frequency values (in MHz) of variable implementation lengths (32-, 64-, 128-, 256-, 512-, and 1024-bits) for the seven FPGA devices. It can be clearly seen that the lowest frequencies are recorded for the FPGA implementation using Altera with 104.2, 77.1 and 43.4 for 32-, 64-, and 128-bits, respectively. Additionally, no numbers have been recorded for higher bit lengths due to the capability of such an educational device. The next two devices have higher rates. The maximum frequencies for the implementations based the Artix-7 and the Vertix-5 are greater than Altera by almost 33% and 14%, respectively. The performances for the implementations based on the Vertix-7 and Zynq devices were identical and have equal frequencies, with an average enhancement of 48% in overall frequency relative to Altera. The Spartan 6 version showed a similar tendency as the Altera version except for the 32-bit implementation length. The highest numbers belonged to the implementation based on the Kintex-7 device, with an 11% increase in frequency compared to the Vertix 7 for the 32-bit length.

In contrast, minimum period values (in nano-seconds) shown in figure 4 were much lower in all device families. The highest numbers were recorded in the Artix-7 and the Vertix-5 with the 1024-bit length (44.1 ns and 42.7 ns, respectively). The lowest numbers were recorded in the Kintex-7, the Vertix-7, and the Zynq with 25 ns for the same bit length. Critical path delay values for the Spartan 6 device were 38.1 ns for the same length. No numbers have been listed for the delays in the Altera for bit lengths more than 128-bit. The figures for other bit lengths were relatively uniform and range from 4.1 ns for the 32-bit Kintex-7 implementation to 18.9 ns for the 128-bit Altera implementation and up to 28.4 ns for the 512-bit Spartan-6 implementation, which consumes about twice as much as Vertix-7 and Zynq with similar bit lengths. Thus, higher bit length implementations (256, 512, and 1024) can be implemented with the Vertix-7, Zynq, and Kintex-7 device families since they recorded the best figures for maximum frequency.
Figure 4. Minimum period values (ns)

Figure 5 shows the total delay values. These values are estimated by multiplying the expected longest source clock period (from source rise to destination rise) by the number of logic stages (levels). The number of levels is that required by the synthesizer to travel from the input of a flip-flop or latch through logic and routing and arrive at the output of the chip before the next clock edge. This includes the clock-to-Q delay of the source flip-flop and the path delay from that flip-flop to the output pad (ISE 14.1 Synthesizer, 2014). The figure clearly states that the longest delay period is related to the Artix-7 for the 1024-bit length with 88.2 ns. The best delay time is related to the Kintex-7, Vertex-7 and Zynq, with equal delays of 50.1 ns for the same bit length.

Figure 5. Total delay values (ns)

From the obtained results, we can see that the minimum delay and maximum frequency occur when the precision/datapath size is 32 bits with the Xilinx Kintex-7 XC7K70T-2-FBG676 applied. When the operands precision increases, the delay linearly increases. The delay increases as the number of bits increases. For a higher datapath size such as 1024 bits, the maximum frequency has been recorded for the Xilinx Vertex-7 XC7VH290T-2-HCG1155 and the Xilinx Kintex-7 XC7K70T-2-FBG676. Even though some of the previous designs and implementations might be different in the architecture, datapath size and devices technology, the comparisons between our implementation and others are valid, as they show that our proposed implementation is competitive with many dedicated
solutions. For instance, the FPGA implementation of the GCD in [16] recorded a maximum frequency of 456.09 MHz at an operand size of 8-bits for the Reversible GCD control unit using the Spartan-3 XC3S50 family. Instead, our GCD processor with the same chip family but a higher version (Spartan-6 XC7Z010-2-CLG400) computes the GCD 1.8 times faster and would be 2.03 times faster if used with the Kintex-7 version. Furthermore, [28] the synthesized Euclid’s GCD implementation for the Xilinx Spartan-6 XC6SLX75 and the implementation tools reported a maximum frequency of 200 MHz for the 32-bit size. For our processor, it is 217.086 MHz synthesized with the Xilinx Spartan-6 XC7Z010-2-CLG400 (the same FPGA Family) and 243.9 MHz synthesized with the Xilinx Kintex-7 XC7K70T-2-FBG676. Therefore, our processor throughputs are 1.09 and 1.93 times faster, respectively.

Table 2. Hardware Utilization Using the XC7VH290T-2-HCG1155.

<table>
<thead>
<tr>
<th>Precision</th>
<th>32-bit</th>
<th>64-bit</th>
<th>128-bit</th>
<th>256-bit</th>
<th>512-bit</th>
<th>1024-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>376 (0%)</td>
<td>729 (0%)</td>
<td>1435 (0%)</td>
<td>2844 (0%)</td>
<td>5658 (1%)</td>
<td>11290 (2%)</td>
</tr>
<tr>
<td>LUTs</td>
<td>508 (0%)</td>
<td>964 (0%)</td>
<td>1637 (1%)</td>
<td>2697 (1%)</td>
<td>5315 (2%)</td>
<td>10518 (4%)</td>
</tr>
</tbody>
</table>

Table 2 shows the hardware utilization results for the GCD coprocessor when implemented using the Vertix7 (device: XC7VH290T-2-HCG1155) represented by the number of utilized registers (the total number of registers in the target device is 437600) and the number of utilized Lookup Tables - LUTs (the total number of LUTs in the target device is 218800). It is clear that the implementation with different precisions utilizes fewer resources of the target device. The largest implementation length (i.e., 1024 bit) utilizes a maximum of 2% and 4% of device registers and LUTs, respectively. This indicates that the implementation area is scalable and can be easily increased or embedded with many other design applications.

Table 3. Total FPGA Thermal Power Dissipation (mW) using the Altera Cyclone IV E (EP4CE115 F29C7)

<table>
<thead>
<tr>
<th>Precision</th>
<th>8-bit</th>
<th>16-bit</th>
<th>32-bit</th>
<th>64-bit</th>
<th>128-bit</th>
<th>164-bit</th>
<th>256-bit</th>
<th>512-bit</th>
<th>1024-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Power</td>
<td>4.0</td>
<td>7.0</td>
<td>13.0</td>
<td>26.0</td>
<td>50.0</td>
<td>64.0</td>
<td>100.0</td>
<td>200.0</td>
<td>300.0</td>
</tr>
<tr>
<td>Static Power</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
<td>135.0</td>
</tr>
<tr>
<td>Total FPGA</td>
<td>139.0</td>
<td>142.0</td>
<td>148.0</td>
<td>161.0</td>
<td>185.0</td>
<td>199.0</td>
<td>235.0</td>
<td>335.0</td>
<td>435.0</td>
</tr>
</tbody>
</table>

Table 3 shows the total FPGA thermal power dissipation (mW) values consumed from applying the GCD algorithm with different datapath lengths (8-bit to 1-kbit)) to the Altera Cyclone IVE (ep4ce115 f29c7) FPGA kit, where \( \text{Power}_{\text{Total,FPGA}} = \text{Power}_{\text{I/O}} + \text{Power}_{\text{Static}} \). The estimated power results for the design with different precisions from 8-bit to 164-bit were generated using the powerplay early power estimator tool in the Quartus II CAD simulation pack. The 164-bit design was the largest datapath that allowed for the power estimation tool due the number of I/O pins provided to by the target FPGA kit. The 512 pins cover, two 164-bit inputs, one 164 bits output result, and other pins, are for control signals such as clock, enable, acknowledge and reset. The power values for the larger designs (i.e., 256, 512 and 1024 bits) can be extrapolated from the general trend for power figures.
The total FPGA design power is mostly affected by the I/O power, while the term of power is constant (i.e., static power), as articulated in figure 6.

![Figure 6. Total FPGA thermal power dissipation (mW)](image)

To sum up, this paper generates attractive synthesized results that can be used to implement a GCD processor using parallel arithmetic units and redundant multipliers and adders, such as those that are commonly used in cryptographic systems over a known finite field. It was found that choosing the best chip technology would increase the throughput of the arithmetic operations.

6. Conclusions

In this paper, we propose an efficient FPGA implementation for a GCD processor based on Euclidian's algorithm using Altera FPGA devices that improve the computational process. The performance of the proposed implementation is studied in terms of both critical path delay (ns) and the maximum frequency (MHz) to compare the performance of the proposed coprocessor using different implementations and simulations. In addition, the synthesized results of this paper targeted seven different chip technologies (one Altera chip and six Xilinx chips) and six different datapath sizes (32- to 1024-bits). It was found that the Xilinx Vertix-7 XC7VH290T-2-HCG1155 and the Xilinx Kintex-7 XC7K70T-2-FBG676 could be used as the fastest FPGA chip devices. Eventually, maximum frequencies of 243.934 MHz for 32-bit datapaths down to 39.94 MHz for 1024-bit data paths have been achieved, which show that the proposed coprocessor implementation has a throughput efficiency of up to two times faster than other state-of-the-art implementations.

References


[22]. ALTERA Corporation, (2013) 'Introduction to the Quartus® II Software'.


