HARDWARE IMPLEMENTATION OF MODULUS OPERATION

P.Y.V. Hemantha*, W. A. S. Wijesinghe

Department of Electronics, Wayamba University of Sri Lanka, Kuliyapitiya, Sri Lanka
hemantha1215@gmail.com

ABSTRACT

This paper describes the hardware implementation of modulus operation. The modulus operation is a basic mathematical operation which has wide range of applications. With the popularity of programmable hardware, it has been exploiting to accelerate m athematical operation in many applications such as cryptography. One of the main mathematical operations in many cryptographic algorithms is the modulus operation. For hardware acceleration of those applications, it is necessary to have hardware-based modulus operators. In this study we develop an algorithm to find modulus operation of two numbers and ported into a Field Programmable Gate Array(FPGA) with Verilog Hardware Description(HDL) language using a Finite State Machine (FSM).Simulation results show the calculation is correct. This hardware algorithm of modulus operation can be used for applications such as cryptography.

Key words: *Modulus, FPGA, Verilog HDL.*

1.0 INTRODUCTION

In computing, modulo (sometimes called modulus) operation finds the remainder of division of one number by another. Given two positive numbers, **a**(the dividend) and **b**(the divisor), **a**modulo**b** (abbreviated as **a** mod **b** is the remainder of the Euclidean division of **a**by**b.** For instance, the expression "5 mod 2" would evaluate to 1 because 5 divided by 2 leaves a quotient of 2 and a remainder of 1, while "9 mod 3" would evaluate to 0 because the division of 9 by 3 has a quotient of 3 and leaves a remainder of 0; there is nothing to subtract from 9 after multiplying 3 times 3. That means "5 mod 2" is 1 and "9 mod 3" is 0[1].

Modulus operation is proposed for FPGAs that addresses the issue of scalability, flexible performance and silicon efficiency for the hardware acceleration of modulus operation. This paper proposes the hardware implementation of the modulus operation and Hardware

Description Language (HDL) uses to FPGA configuration. The novelty and the main interest in this paper is the orientation towards the hardware implementation. The result is hardware implement of the modulus operation for a fast, cheap and efficient. The benefits of hardware implementation were given below.

- Performance

  Taking advantage of hardware parallelism, FPGAs exceed the computing power of digital signal processors (DSPs) by breaking the paradigm of sequential execution and accomplishing more per clock cycle. BDTI, a noted analyst and benchmarking firm, released benchmarks showing how FPGAs can deliver many times the processing power per dollar of a DSP solution in some applications.[2] Controlling inputs and outputs (I/O) at the hardware level provides faster response times and specialized functionality to closely match application requirements.

- Time to market

  FPGA technology offers flexibility and rapid prototyping capabilities in the face of increased time-to-market concerns. You can test an idea or concept and verify it in hardware without going through the long fabrication process of custom ASIC design.[3]You can then implement incremental changes and iterate on an FPGA design within hours instead of weeks. Commercial off-the-shelf (COTS) hardware is also available with different types of I/O already connected to a user-programmable FPGA chip. The growing availability of high-level software tools decreases the learning curve with layers of abstraction and often offers valuable IP cores (prebuilt functions) for advanced control and signal processing.

- Cost

  The nonrecurring engineering (NRE) expense of custom ASIC design far exceeds that of FPGA-based hardware solutions. The large initial investment in ASICs is easy to justify for OEMs shipping thousands of chips per year, but many end users need custom hardware functionality for the tens to hundreds of systems in development. The very nature of programmable silicon means you have no fabrication costs or long lead times for assembly. Because system requirements often change over time, the cost of making incremental changes to FPGA designs is negligible when compared to the large expense of respinning an ASIC.

- Reliability

  While software tools provide the programming environment, FPGA circuitry is truly a "hard" implementation of program execution. Processor-based systems often involve several layers of abstraction to help schedule tasks and share resources among multiple processes. The driver layer controls hardware resources and the OS manages memory and processor bandwidth. For any given processor core, only one instruction can execute at a time, and processor-based systems are continually at risk of time-critical tasks preempting one another. FPGAs, which do not use OSs, minimize reliability concerns with true parallel execution and deterministic hardware dedicated to every task.

- Long-term maintenance

  As mentioned earlier, FPGA chips are field-upgradable and do not require the time and expense involved with ASIC redesign. Digital communication protocols, for example, have specifications that can change over time, and ASIC-based interfaces may cause maintenance and forward-compatibility challenges. Being reconfigurable, FPGA chips can keep up with future modifications that might be necessary. As a product or system matures, you can make functional enhancements without spending time redesigning hardware or modifying the board layout[2].

2.0 EXPERIMENTAL PROCEDURE

2.1 Modulus Algorithm

This is the algorithm for the modulus function. If mod[a, b]= modvalue, {a=divident , b=divisor}.

```
mod (a, b)
{
        i=0, tb=0;
        while (a>tb)
        {
                i=i+1;
                tb=b*i;
        }
        tb =b*(i-1);
        modvalue=a-tb
}
```

2.2 Status Diagram

The elements that constitute a state diagrams are rounded boxes representing the states and arrows showing transitions to the next state. The activity section depicts the activities the object performs while it is in that state. Every state diagram starts with an initial state, which is the state where the object is created. Right after the initial state, objects change their states, and the next state is determined by conditions based on activities. In some cases, state diagrams represent a super state, which is a condition created when many transitions lead to a particular state. The super state depicts that all states inside this diagram transition to a redundant state, making the diagram more complex. A transition in a state diagram is a progression from one state to another and is triggered by an event that is internal or external to the entity modeled. An action is an operation that is invoked by an entity that is modeled. A very traditional form of state diagram for a finite machine is a directed graph[3].

The below status diagram was related to the 2.1 algorithm.

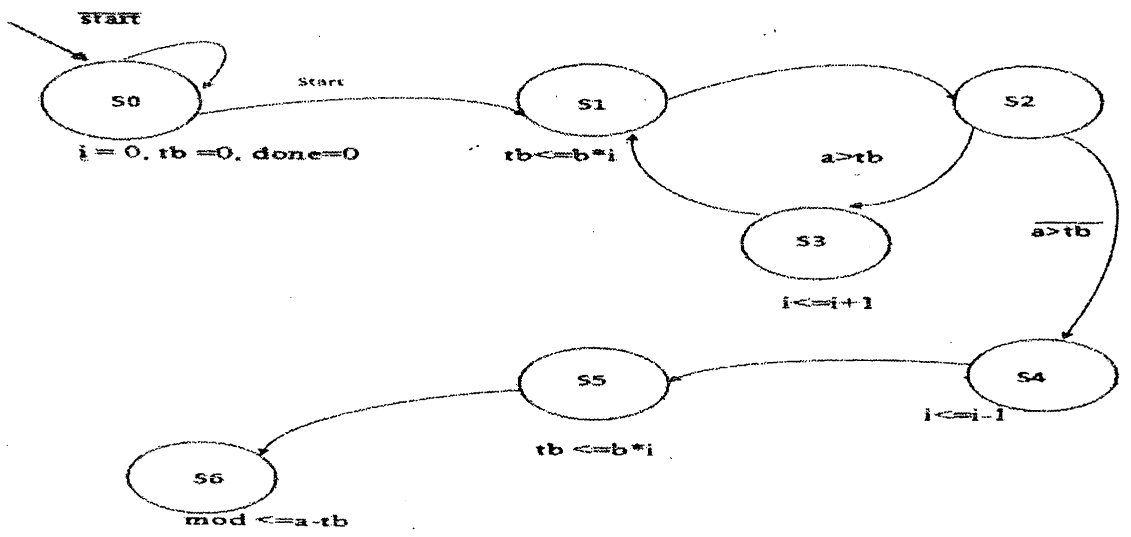

Figure 1: Status diagram for modulus operation

2.3 Hardware implementation

Hardware Description Language (HDL) used for the program FPGA. Programmed the algorithm for modulus function by using Icarus verilog and Spartan 3 Field Programmable Gate Array (FPGA) development board is used for that purpose. Output is displayed in computer. Serial communication was the link from FPGA to computer[4].

3.0 RESULTS AND DISCUSSION

At the highest level, FPGAs are reprogrammable silicon chips. Digital computing tasks in software and compiles them down to a configuration file or bit stream that contains information on how the components should be wired together. In addition, FPGAs are completely reconfigurable and instantly take on a brand new "personality" when recompile a different configuration of circuitry. In the past, FPGA technology could be used only by engineers with a deep understanding of digital hardware design. The rise of high-level design tools, however, is changing the rules of FPGA programming, with new technologies that convert graphical block diagrams or even C code into digital hardware circuitry.

FPGAs provide hardware-timed speed and reliability. Reprogrammable silicon also has the same flexibility of software running on a processor-based system, but it is not limited by the number of processing cores available. Unlike processors, FPGAs are truly parallel in nature, so different processing operations do not have to complete for the same resources. Each independent processing task is assigned to a dedicated section of the chip, and can function autonomously without any influence from other logic blocks[5].

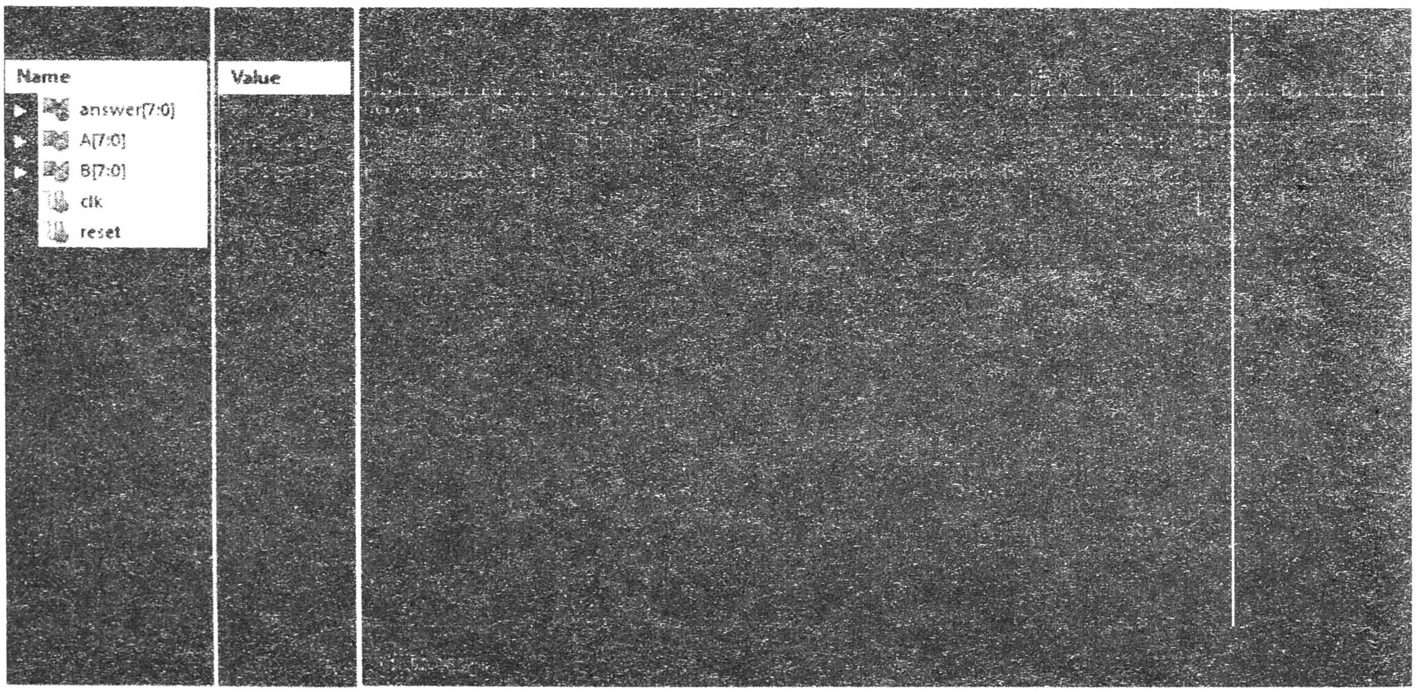

Figure 2: GTK wave form for one of modulus function

4.0 CONCLUSION

In this study we have attempted to implement modulus operation in hardware. Since the modulus operator is one of basic mathematical operations, hardware-based algorithm for modulus operation is necessary to applications such as cryptography where hardware acceleration is concerned. Simulated results show that the hardware algorithm gives the correct results. One drawback of the algorithm is that it takes different clock cycles to calculate the final result for different input values.

REFERENCES

[1]. S.E. Eldridge, C.D. Walter, Hardware Implementation of Montgomery's Modular Multiplication Algorithm, 42(6)(1993)693-699

[2]. A. Boute, T. Raymond, The Euclidean definition of the functions div and mod, ACM Transactions on Programming Languages and Systems , TOPLAS, ACM Press , New York, NY, USA, 14 (2), (1992) 27–144.

[3]. How to write FSM in Verilog http://www.asic-world.com/tidbits/verilog_fsm.html

[4]. BDTI Industry Report(2nd edition),FPGAs for DSP, Berkeley Design Technology Inc, 2006

[5]. http://www.xilinx.com/tools/isim.htm