International journal of Engineering Research-Online
A Peer Reviewed International Journal
Articles available online http://www.ijoer.in

Vol.1., Issue.3., 2013

RESEARCH ARTICLE

ISSN: 2321-7758

DESIGN OF FLEXIBLE DUAL CORE PROCESSOR USING VHDL

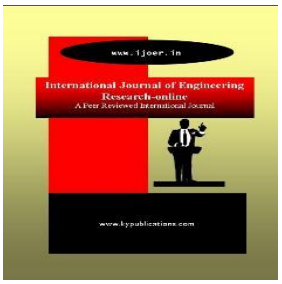
RAHUL PANDEY, VIJENDRA KUMAR PATEL, SACHIN KUMAR TYAGI, VIVEK MAHESHWARI, KAVERY VERMA, DEEKSHA CHANDOLA

Department of Electronics, JAYPEE Institute of Information Technology
Noida, India

Article Received: 04/12/2013

Article Revised on:12/12/2013

Article Accepted on:14/12/2013



ABSTRACT

In this paper a flexible multicore processor is designed using VHDL. Here flexibility means the "Special Instructions loading in different modes". It might be possible to divide particular sets of instruction for different applications. This design employs two different cores and a scheduler. The cores incorporate ALU and register files. Scheduler manages the incoming workload to these cores. The paper provides an insight view of whole design along with simulation results for a specific assembly core run on this design.

Keywords— Core, ALU, Opcode decoder, Scheduler, multi-core, VHDL

INTRODUCTION

Few years ago, increasing the clock speed was the preferred tactic by manufacturers to gradually increase the performance of computers. However, from certain speeds there are some limitations. Some effects of these physical limitations can be the high heat dissipation, problems with the synchronization of the signals, or the high energy consumption. Therefore now, multicore microprocessors are being developed to increase the performance of the devices without increasing the clock frequency of the microprocessors.

The core of the processor consists of a processing unit and Cache. The first step of design is to design an ALU and then full design of core is done. Then these cores are attached with the scheduler. The scheduler enables the timing of different processes with the application, loading the instructions and synchronization.

The processor core is designed with three stage pipeline: Opcode fetch, Opcode decoder, Datapath. The opcode fetch generates the opcode i.e. binary associated bits, opcode decoder then decodes the instruction i.e. it specifies the information about the control signals and the operation of ALU involved in the instruction. It is the middle stage of the pipeline. Now the datapath unit, which is the actual execution unit in the core, processes the instruction and stores the data. Both the cores are managed with the help of scheduler. Scheduler distributes the workload between them.

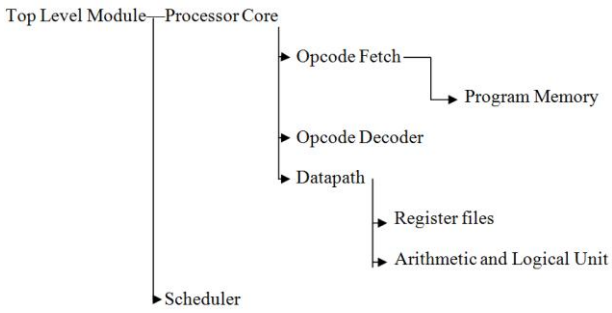


Figure 1. Hierarchy of the design

RELATED WORK

The idea of increasing control bits and reuse them at the scheduler stage for adding more flexibility is new and haven't been processed. The common theme in much of this work is a choice between two target approaches: pipelining for speed variation or flexibility for different dedicated processors. The dedicated processors are made using the different instruction set. But with the proposed approach the design may cover a wide area of applications without sacrificing the speed. The design proposed in a compromise between both speed and flexibility. It provides a good flexibility with a complex pipelined structure. The feature of loading instructions at scheduler may become very efficient in multicore processors for different dedicated threads in accordance with the applications. The design is simulated with Modelsim 10.1b, a simulation tool of Mentor Graphics.

DESIGN OF CORE

Core is the execution unit of the processor. These cores might be designed for general tasks or some dedicated tasks. Three stage pipelining is used to design the core. The same kind of Core can come in different flavors that differ in the clock frequency that support, bus sizes, the size of internal caches and memories and the capabilities of the I/O ports they provide.

The processor core is primarily characterized by the instruction set that it provides. One could also say that the CPU core is the implementation of a given instruction set. The details of the instruction set will only be visible at the next lower level of the design. At the current level different cores (with different instruction sets) will still look the same because they all use the same structure. Only some control signals will be different for different cores. We have used Harvard architecture in the design of memory. Harvard architecture means that the program memory and the data memory of the CPU are different.

Opcode Fetch Unit: In this opcodes are generated. The opcode fetch stage is the simplest stage in the pipeline. It is the stage that put life into the CPU core by generating a sequence of opcodes that are then decoded and executed. The opcode fetch stage is sometimes called the sequencer of the CPU.

Since we use the Harvard architecture with separate program and data memories, we can simply instantiate the program memory in the opcode fetch stage. The main purpose of the opcode fetch stage is to manipulate the program counter (PC) and to produce opcodes. The PC is a local signal. In this the program memory used is a dual port memory. This means that two different memory locations can be read or written at the same time.

The opcode fetch stage is also responsible for part of the interrupt handling. Interrupts have generated in the I/O block by setting a signal INTVEC (defined for enabling the interrupt) to a value with the highest bit set. The highest bit of INTVEC indicates that the lower bits contain a valid interrupt number.

OPCODE DECODER: Designing an opcode starts with asking a number of questions. The answers are found in the specification of the opcode. The answers identify the outputs that need to be set other than their default values. While the instructions are quite different, the questions are always the same:

- What operation shall the ALU perform? Set different signals accordingly. These signals control the operation of ALU.

- Is a destination register or destination register pair used? If so, set signal specified for registers write enable.
- Does the opcode access the memory? If so, set different signals responsible for memory accordingly.
- Is the program counter modified excluding incrementing it? If so, set the signals which dynamically control the program counter.

The opcode decoder is the middle state of our CPU pipeline. Therefore its inputs are defined by the outputs of the previous stage and its outputs are defined by the inputs of the next stage. Most data buses of the CPU are contained in the data path. In contrast, most control signals are generated in the opcode decoder. We start with a complete list of these control signals and their purpose. There are two groups of signals: select signals and write enable signals. Select signals are used earlier in the execution of the opcode for controlling multiplexers. The write enable signals are used at the end of the execution to determine where results shall be stored. Select signals are generally more time-critical that write enable signals.

EXECUTION UNIT: Execution unit consists of arithmetic and logical unit and registers. ALU may be considered as the CPU of the computer system. High performance embedded systems requires a low power and faster arithmetic and logical units. The design proposed in the paper consists of some empty control inputs which may be loaded with different set of instructions in accordance with the need in the application. Different applications have some commonly used processes. These processes run with some specific set of instructions. Whenever the process is executed, these instructions are also executed simultaneously. But if these set of instructions are made independent with specific control inputs then the speed of ALU increases with some added flexibility for different new instructions. This is very useful for dedicated processors.

The arithmetic unit in the design deals with the IEEE-754 standards. This enables the arithmetic unit to process very low and even very high floating point binary numbers. The range calculated in decimal to be operated in the design is $2^{-126}(1.0)$ to $2^{127}(2\text{-}2^{-23})$. All the units adder, subtractor and multiplier are pipelined and hence the speed of ALU is increased. For further improvement in the speed, the distributed arithmetic technique is used for multiplication. This also reduces the memory requirements in the further implementations and proves to be very efficient.

This unit uses floating point number with precision IEEE 754 format. In this representation 23 bit mantissa, 8 bit exponent and a signed bit is used. The range which a IEEE 754 precision format can deal in decimal is $2^{-126}(1.0)$ to $2^{127}(2\text{-}2^{-23})$. This scheme is very efficient for the arithmetic calculations in binary. The arithmetic unit consists of partitions: adder and subtractor; multiplier and divisor.

The adder consists of three stages: comparator, shifter, rounding and normalization. A 23 bit with carry look ahead technique is designed independently. The 23 bit CLA adder is used to carry out the operations between mantissa. A signal 'op' and the signed bits of both the numbers carries out the information for decision of addition or subtraction. For any operation either addition or subtraction, the exponent of both the numbers must be compared and the lower one's mantissa is shifted right to the difference of both. Then the overflow is checked and rounding is done. Thus a three stages pipelined structure is used in it.

For multiplication of two floating point numbers requires sum or difference of exponent and multiplication of their mantissa. The signed bit is calculated by Ex-or operation between the signed bits of both the numbers. It might be possible for an overflow of exponent of output after adding both the exponent. In this case the overflow flag is made high and exponent is set to its maximum.

After the processing of exponent, multiplication of mantissa a 23 bit multiplier is needed. In the design the distributed technique is used for this multiplication. In it the inner product technique is used. It's been found that it might reduce the gate count upto 50%. The sum of product may be represented as

$$Y = \sum_{k=1}^{K} A_k X_k$$

Here X_k is considered as 2's complement binary number and may be written as:

$$X_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n}$$

Here k is taken as the number of input and N is length of data. This is the basic technique involved in distributed arithmetic for multiplication. The multiplication in the design is implemented using this concept only. This technique might not be best in all the applications but it doesn't seem to be poor in any type of application. In digital filters this technique provides very efficient results.

By using this technique a 23 bit multiplier module is made and is used to multiply the mantissa. But the problem is now that the multiplication output is of 46 bit but in the output also it is necessary to have a mantissa of 23 bit. To resolve this problem, an exclusive shifter is used which works by checking the exponent. The mantissa is shifted left till either the exponent reaches to minimum or last bit of output found to be one. Then the 23 bits from this output are selected from MSB and rounded for the mantissa of the result. For division a 23 bit division module is made and instead of adding the exponents, the exponents are subtracted. Thus all these operations are performed by the ALU.

SCHEDULER

Multicore processors promise high execution efficiency under diverse workloads, and program scheduling is critical in exploiting this efficiency. The proposed method projects the core's configuration and the program's resource demand to a unified multi-dimensional space, and uses weighted Euclidean distance between these two to guide the program scheduling.

The design uses a scheduling strategy that performs dynamic job grouping activity at runtime and convey the detailed analysis by running simulations. In addition, job processing granularity size is introduced to facilitate the job grouping activity in determining the total amount of jobs that can be processed in a resource within a specified time. This evaluates a dynamic scheduling strategy that maximizes the utilization of Grid resource processing capabilities, and reduces the overhead time and cost taken to execute the jobs on the Grid. The proposed job scheduling strategy takes into account: the processing requirements for each job, the grouping mechanism of these jobs, known as a job grouping, according to the processing capabilities of available resources, and the transmitting of the job grouping to the appropriate resource.

SIMULATION RESULTS

The designed processor uses 330 instruction codes as designed. A sample assembly language code with designed instructions is simulated with the help of the designed processor in VHDL:

ADIW R_R26,0003H
ADIW R_RSP,F007H
ADIW R_R26,703CH
AND R_R26, R_RSP
LDI R28,1aH
JMP ff03H

The simulation results for each instruction is shown in the following figure:

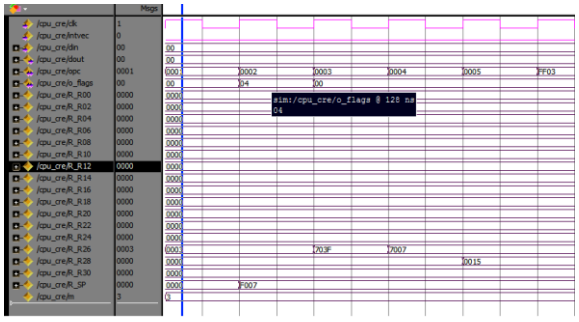


Figure 2 Simulation Results after ADIW R_R26,0003H

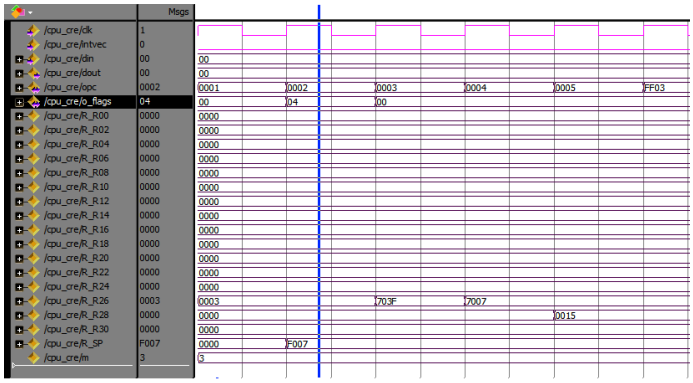


Figure 3 Simulation Results ADIW R_RSP,F007H

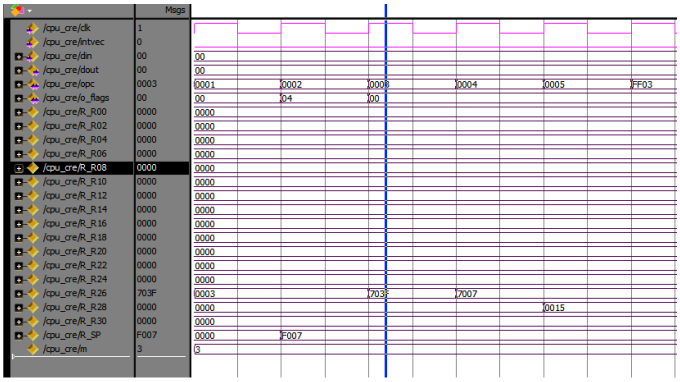


Figure 4 Simulation Results ADIW R_R26,703CH

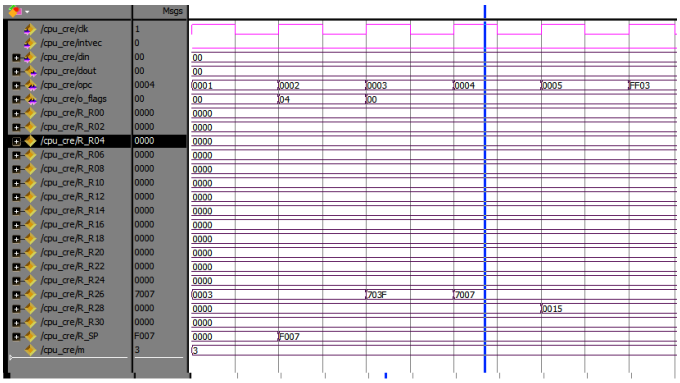


Figure 5 Simulation Results AND R_R26, R_RSP

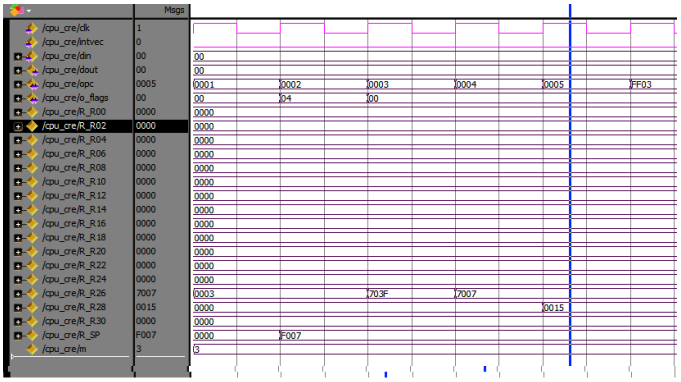
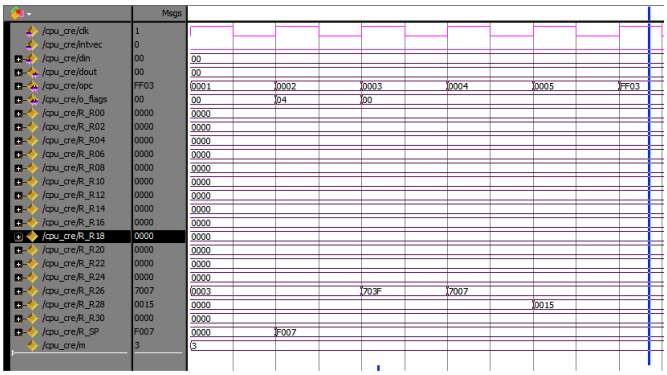


Figure 6 Simulation Results LDI R28,1aH



FFigure 7 Simulation Results JMP ff03H

ACKNOWLEDGMENT

The authors thank the assistance provided by Sapan Kumar Gupta, Asst. Professor, DGI Greater Noida, Rahul Singh, Network Engineer, Data Center, BHU, Varanasi and Aditya Pathak of YLM Tube Solutions & Service India Pvt. Ltd. Additional thanks go to Vishwajeet Sahu and Pushkar Mishra for their great effort and inspiring and assisting us for this paper .Authors are very thankful to JIIT Noida for providing support to use the licensed version of ModelSim 10.1 b in institution.

REFERENCES

[1] Sapan Kumar Gupta, Rahul Pandey, Manuj Kumar, " Future Generation Processor" , International Journal of Engineering Research and Technology, ISSN 2278-0181, Volume 1, Issue 6 (2012)

[2] Sapan Kumar Gupta, Ankit Gupta, et.al. " Multicore signing off: What is the future" , International Journal of Electronics and Electrical Engineering, ISSN 0974-2174, Volume 5, Number 5 (2012), pp 397-400

[3] Sapan Kumar Gupta, Manuj Kumar, Rahul Pandey, "Processor Technology Advancement" , International Journal of Engineering Trends and Technology , ISSN 2231-5381 Volume 3 Isuue 3(2012)

[4] Sapan Kr. Gupta et.al. "Design of a Faster and Flexible 32 bit ALU implementing single precision IEEE-754 floating point numbers using VHDL", Proceedings of ICTRC 2013,ISBN SRM University, Modi Nagar

[5] Sapan Kumar Gupta, Sachin Tyagi et.al. " Hyperthreading: A Solution for Higher Performance" International Journal of Electronics and Electrical Engineering, ISSN 0974-2174, Volume 5, Number 5 (2012), pp 393-395

[6] ANSI/IEEE Std 754-1985, 1985. IEEE Standard for Binary Floating-Point Arithmetic, IEEE, New York.

[7] Shao J., Y. Ning, Z. Xiao-Yan, 2008. An IEEE Compliant Floating-Point Adder with the Deeply Pipelining Paradigm on FPGAs. In the Proceeding 2008 International Conference on Computer Science and Software Engineering, pp: 50-53.

[8] R. Kumar, et al, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction", *Micro-36, pp 81-92, Dec.*2003.

[9] Singh R.R., A. Tiwari, V.K. Singh, G.S. Tomar, 2011. VHDL Environment for Floating Point Arithmetic Logic Unit-ALU Design and Simulation. In the Proceedings of the 2011. International Conference on Communication Systems and Network Technologies (CSNT), pp: 469-472.

[10] Stanley A. White Application of Distributed Arithmatic to Digital Signal Processing:A Tutorial

[11] G Intel White Paper "Built-in Protection in Laptop PCs Improves Compliance with New Healthcare Rules"

[12] Intel Architectural Server White Paper "Power Management in Intel® Architecture Servers"

[13] Shekhar Borkar"Thousand Core Chips—A Technology Perspective"

[14] "Fifty Years of Microprocessor Technology Advancements: 1965 to 2015"ISA white paper

[15] Viswanath Krishnamurthy, Tyler Sondag, and Hridesh Rajan "Predictive Thread-to-Core Assignment on a Heterogeneous Multicore Processor"

[16] Baskaran Ganesan "Introduction to Multi-Core" Intel White Paper

[17] Bryan Schauer "Multicore Processors – A Necessity"

[18] C. S. Ballapuram, A. Sharif and H. S. Lee, "Exploiting Access Semantics and Program Behavior to Reduce Snoop Power in Chip Multiprocessors", ASPLOS XIII, pp 60-69, Mar 2008

[19] M. Becchi, Patrick Crowley, "Dynamic thread Assignment on Heterogeneous Multi-processor Architectures", *Computing Frontiers* 2006, pp 29-40, May 2006