Power Consumption Reduction in CPU Datapath using a Novel Clocking Scheme

Rajesh Kannan Megalingam, Shekhil Hassan T, Vivek P, Ashwin Mohan, Tanmay Rao M.
Amrita Vishwa Vidyapeetham, Amritapuri, Kollam - 690525, Kerala, India
rajeshm@amritapuri.amrita.edu, vivek_p@ieee.org, shekhil@ieee.org, ashwinmohan@ieee.org, tanmaymrao@ieee.org

***Abstract*— Power consumption and performance are the crucial factors that determine the reliability of a CPU. In this paper, we discuss about some techniques that can be used for Instruction Level Parallelism which enhances the performance of the CPU by reducing the CPI there by reducing power consumption. We have also discussed about the power saving scheme using proper clocking strategies. We have mainly focused on implementing the simplified RISC pipeline datapath in HDL using two different clocking schemes to reduce the power consumption. We have adopted a new method which uses dual edge triggered clock that can decrease the power consumption of a pipelined datapath considerably, without sacrificing the throughput of the CPU. Finally, we have given the experimental result for our implementation of simplified RISC datapath.**

Keywords-ILP; power; LDPR; BHT; BTB; dual edge triggered clock; RISC pipeline datapath

I. INTRODUCTION

Pipelining is an important method that increases the efficiency of the CPU. In a pipelined microprocessor, parallel execution of different instructions takes place by making use of different functional units of the datapath simultaneously. Using Pipelining, multiple instructions can be overlapped in execution. In pipelining, each instruction is completed in many stages or segments. Micro-operations occur in each stage. Pipeline helps in reducing the total execution time of the program and it increases instruction throughput.

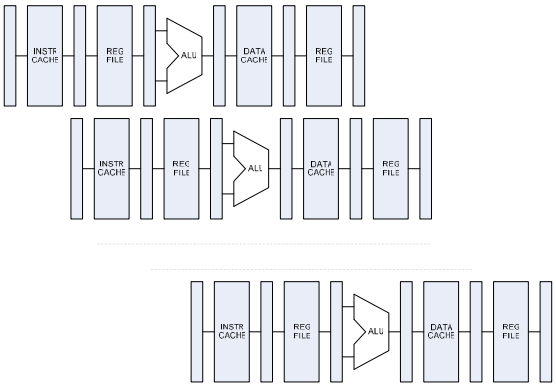

Figure 1. Basic overlap among the segments in a simple pipeline datapath

978-1-4244-4520-2/09/$25.00 ©2009 IEEE

A pipeline processor consists of a sequence of data-processing circuits called stages or segments, which collectively perform a single operation on a stream of data operand passing through them. Fig.1 shows the basic overlap among the segments in a simple pipeline datapath which has 5 pipeline stages Fetch, Decode, Execute, Data Memory and Write Back [1].

There are many microprocessor applications, typically battery-powered embedded applications, where energy consumption is the most critical design constraint. In CMOS technology, most of the power consumption happens during transistor switching which is referred to as dynamic power. Clock power is a major contributor of the overall power because the clock is fed to most of the circuits in the processor and it switches (on and off) every cycle. Clock gating can be used to reduce the clock power whenever a sequential circuit is not using clock. But this is not a good method to be adopted since the combinatorial delay will be added to the clock and can affect the synchronization.

II. POWER DISSIPATION.

There are two types of power dissipation in the various blocks that are used in the datapath. They are static dissipation and dynamic dissipation.

A. Static Dissipation

Static dissipation is due to leakage current or other current drawn continuously from the power supply. The main cause of static dissipation is due to reverse bias leakage current between diffusion regions and substrate. The sub-threshold conduction also contributes to the static dissipation.

The static power dissipation is given by the product of device leakage current and the supply voltage. It usually occurs in pseudo nMOS gates, where there is direct path between power and ground

B. Dynamic Dissipation

Dynamic power is due to switching transient current and due to charging and discharging of load capacitances. The dynamic power dissipation is of more importance to us this occurs mainly during the transition from 0 to 1 and from 1 to 0. Current also charges and discharges the output capacitive load and this capacitive charging and discharging current is dominant in dynamic dissipation .The dynamic power P_d is given by

$$P_d = C_L V_{DD}^2 f_p$$

C_L - Load capacitance

f_p - Frequency of switching

This equation shows that the power consumption is directly proportional to the operating frequency. Hence, if we can decrease the operating frequency, it will reduce the power consumption by a significant amount. This will enhance the performance of the CPU to a great extent.

III. TECHNIQUES FOR ENHANCING INSTRUCTION LEVEL PARALLELISM

Clock power is a major factor in determining the total power of the system. Thus power dissipation can be decreased if we can reduce the number of clock cycles for instructions. By enhancing the instruction level parallelism we can optimize the power usage. There are various techniques for achieving it.

A. LDPR:

LDPR (Longest Data Dependence Path Ratio) is a parameter by which we can estimate the amount of instruction level parallelism among a set of instructions. Instruction window is divided into various segments and for reducing power consumption they can be disabled when they are unused. Number of segments that a program can utilize is limited, based on the amount of ILP estimated using the LDPR. Highest ILP is available when LDPR is near zero and lowest ILP is available when the LDPR is near one. LDPR can be calculated by the graphical representation and measurement of the longest dependence path (LDP). It is calculated by taking the ratio of LDP and total number of instructions. ILP rate can be increased by normalizing the value of LDPR. This can be achieved by taking a small block of instructions with higher LDPR (ILP low) executed first followed by a block with lower LDPR (ILP high). This results in normalizing the value of LDPR and hence increasing ILP. LDPR indicates the amount of resource needed for the execution of the specific block of code and thus the segments which are not used can be switched off, and hence reduce the power consumption [2].

B. Branch Prediction Buffer

Branches are major limiting factor in determining the amount of ILP. When we encounter a conditional branch instruction, we cannot know whether the branch will be taken or not unless the execute stage is completed. Branch prediction buffer is a memory with many locations and each location contains a prediction bit. This prediction bit determines whether the branch will be taken or not. Branch prediction buffer is also known as branch history table (BHT). BHT is indexed by the low order (few of LSB) bits of Program Counter (PC) during instruction fetch and fetched with the instruction. If the instruction is decoded as branch, the prediction bit is checked and fetching begins from the target as soon as the PC is known. When a branch is taken, the prediction bit is asserted, or else, the prediction bit is inverted. In some cases, frequent inversion of the prediction bit of the same instruction will create ambiguity. In order to resolve this, we can clear the prediction bit each time when the branch is untaken. Fig. 2 shows the four state branch prediction schemes [1].

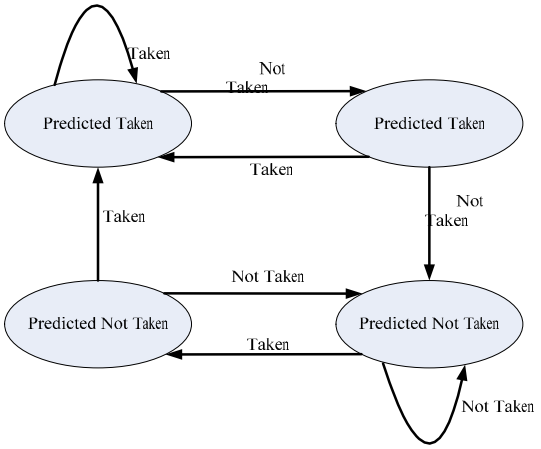

Figure 2. Four state branch prediction scheme

C. Branch Target Buffer

Branch Target Buffer (BTB) is a cache like memory with each location storing instruction address, corresponding predicted address for the next instruction after branch and a prediction bit. First, the current PC value is matched against a set of instruction addresses stored in BTB. If any of the address matches with PC value, the corresponding branch target address and the prediction bit are sent out. During the decode stage, when the instruction is known to be a branch, the prediction bit is checked. If the branch is taken (when the prediction bit is set) the instruction will continue the execution without stalls. If the branch is not taken (when the prediction bit is cleared) it will proceed to another step in the same stage. If the branch target address is less than the PC value, it is obvious that the branch is a backward branch. In general, backward branches are normally used to implement loops. Thus we will consider backward branches as taken, even if the prediction bit is de-asserted. The prediction bit of a branch instruction is cleared in BTB when branch is untaken. If a branch is mis-predicted, microprocessor will suffer a penalty of two clock cycles [6].

D. Other Techniques

Out-of-order execution is one technique employed by compiler to achieve independent instructions in a separate group and those with dependencies in one group, thus increasing the performance as a whole. *Operand forwarding* is also used to reduce data dependency. In operand forwarding the data in an instruction is fed to the previous block of later instruction. Introducing a comparator and an adder in the decoding stage of the pipeline enhances the throughput by reducing the CPI when a branch occurs. This comparator will compare the register values read during this stage and the adder gives the destination address for branching and hence reducing one clock cycle for execution stage. Thus this reduces the power consumed by the

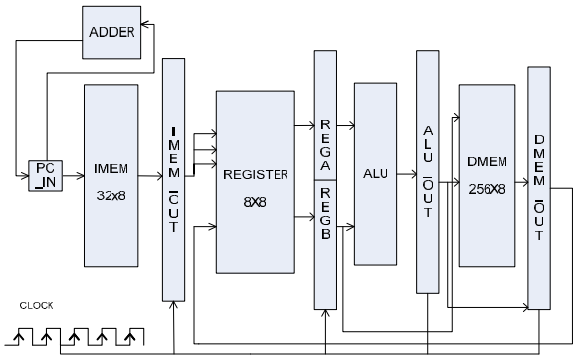

Figure 3. Implementation of datapath using single edge triggered clock

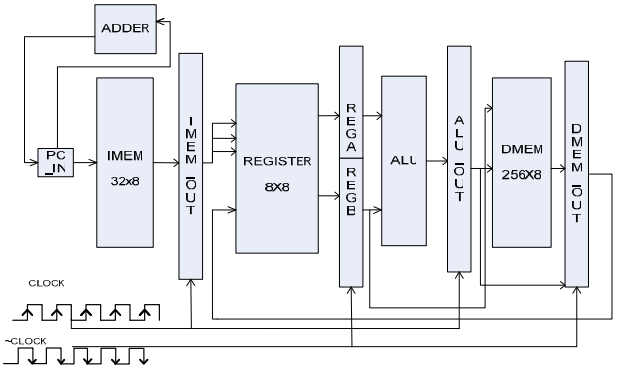

Figure 4. Implementation of datapath using dual edge triggered clock

architecture. Efficient Pipelining could be achieved by eliminating output dependencies and anti-dependencies. Output dependencies and anti-dependencies can be eliminated by automatically allocating new registers to values, when such a dependency has been detected. This technique is called *register renaming* [7].

Other technique used for better performance is *significant compression.* Instructions, addresses and data can be compressed by considering the significant bytes of the word. Some two or three extension bits are also taken into account in order to indicate the significant byte positions. Number of gates or registers used is a major factor that determines the power consumption of the system. Thus we focus on minimum usage of these two. In the case of significant compression, low order bytes and extension bits are accessed first and then the additional bytes are accessed if necessary. Thus power consumption is considerably reduced. This method can be efficiently implemented with pipelining [9].

The above mentioned techniques can be used to reduce the CPI and hence the power consumed by the architecture.

IV. RISC PIPELINING

A general RISC pipeline architecture mainly consists of five stages namely Instruction Fetch (IF), Instruction Decode(ID), Execute(EX), Memory read or write(MEM) and register write back (WB). Hence the pipeline depth is five. Every instruction (except conditional or unconditional branches) should pass through all the stages.

There are two memories- Instruction memory (for storing instructions) and Data memory (for storing data). The only memory instructions in RISC architecture are load (Loading from memory to register) and store (Storing from register to memory). Thus memory access is very less in RISC pipeline when compared to conventional microprocessors.

Each stage will process an instruction in one clock cycle. The maximum delay path will determine the clock frequency. At an average one instruction will be executed in each clock cycle. In the IF stage instruction is read from the memory and simultaneously the program counter (PC) is incremented. In ID stage the operands are read from the memory. Generation of control signals also occur in the ID stage using state machines. In the EX stage ALU operations takes place. In the MEM stage, either data is stored to memory (STORE) or data is loaded from memory to register (LOAD). In the WB stage the data is written back to register.

There are pipeline registers in between for storing the intermediate values. These registers are placed between two stages. The write back addresses, intermediate data values and control signals should pass through these registers [1].

V. IMPLEMENTATION OF DATAPATH

A simplified RISC pipelined datapath was used for analysis and implementation [1]. It comprises four pipeline registers which are used for holding the outputs of each segment, five segments that are used for Instruction Fetching (IF), Instruction Decoding (ID), Execution (EX), Memory Operation (MEM) and Write Back (WB). IF segment in our datapath include an Instruction Memory (IMEM) of 32X18 in size for holding the instructions, one adder used for incrementing the Program Counter (PC) and the output of this segment is fed into Instruction Register (IMEM_OUT) as shown in Fig. 3. IMEM_OUT holds the instruction that is to be given to the next segment. Next segment that follows IF stage is the ID segment where we have one Register block of 8X8 size with three address ports (2 for read address and one for write address), one data input signal, two control inputs (one read control and one write control) and two output signals which gives the read data from the register. This ID stage is followed by a pipeline register called REGA and REGB which stores the read register data. Segment that follows the ID stage is EX, which consists of ALU to perform the operation mentioned in the instruction on operands. ALU has two inputs and one output and this is fed to other pipeline register ALU_OUT. This segment is followed by the Data Memory (MEM) which is 256X8 in size with read and writes controls. But as we are dealing with R type instruction, we will not use this block for execution. Thus the ALU_OUT value is passed to the next pipeline register DMEM_OUT (output of MEM stage) and this is used for next segment WB. WB segment is used for writing back the data to the register. Thus data input is provided by DMEM_OUT in WB stage. The datapath of our design is shown in Fig. 3.

VI. RTL AND SYNTHESIS

We have implemented the above mentioned architecture in Verilog HDL and synthesized the design using Xilinx ISE 10.1. Each stage is implemented as a module. Instruction set format is fixed as explained below.

A. Instruction Set

In our architecture the instruction is 16 bits wide. There are 4 fields in the instruction namely opcode, Register operand 1 and Register Operand 2 represent the two operand read addresses. The fourth field corresponds to the write back register address. Each field is allotted 4 bits. The instruction format is shown in Table I.

TABLE I. REPRESENTATION OF INSTRUCTION FIELDS.

opcode (4 bits)	register address of operand 1 (4 bits)	register address of operand 2 (4 bits)	register address of operand 3 (4 bits)

In the fetch cycle, instructions will be read from the Instruction memory. The instruction stored in the location corresponding to PC_IN is read as shown in Fig. 3. In the next clock cycle the value is written into IMEM_OUT (Pipeline register) and operands are read from the register based on address fields specified by the instruction. In third clock cycle the operand data A is written to REGA (Pipeline register) and operand data B is written to REGB (Pipeline Register). Simultaneously, ALU output is also produced as in Fig. 3. In the fourth clock cycle, ALU output is written to ALU_OUT (Pipeline register). Data Memory operations if any, takes place in the same clock cycle. There are only two memory operations- Storing data to memory and loading data from memory to register. In Register type instructions there is no memory operation. Hence we need not have to bother about Data Memory stage. In the fifth clock cycle data will be written to DMEM_OUT register (pipeline register) and DMEM_OUT's output will be available as input data to the register.

One of the blocks in our design called wb_ctrl (write back control) helps in asserting the write enable to 8x8 register block as shown in Fig. 3. The write back address of an instruction should also be propagated via special registers so as to write back the data after five stages. This has to be done since every register is overwritten in each clock cycle by the data and addresses of instructions below the first instruction.

B. Dual Edge Triggered Clock

Fig. 5 shows the ordinary positive edge triggered clock which is usually used for the working of pipelined architectures. Each segment and the pipeline registers use

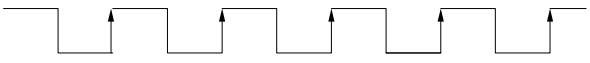

Figure 5. Single edge triggered clock.

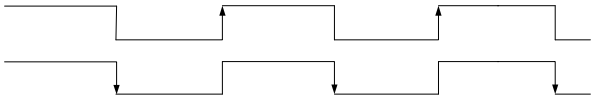

Figure 6. Dual edge triggered clock where triggering occurs in both positive and negative edges.

this clock for latching the result and for further execution of instruction. Pipeline registers work on this clock which stores the result or the output of each segment and forward to the next segment. The above mentioned clock is used for further execution in ordinary condition. In this paper we are discussing about the dual edge triggered clock which gives the same result as when we use ordinary clock. Fig. 6 shows the dual edge triggered clock which has both negative edge and positive edge and both edges are used by pipeline registers.

Here the frequency of the ordinary clock is twice as that of the dual edge triggered clock. Each edge in the dual edge triggered clock is used for execution in pipelined architecture. As shown in Fig. 4, IMEM_OUT and ALU_OUT work in positive edge of the clock whereas REGA, REGB and DMEM_OUT work in negative edge of the clock. From the equation mentioned in section II.B, power is proportional to the operating frequency. Since the frequency of the dual edge triggered clock is half compared to the ordinary clock, power consumed by the architecture is reduced. In the implemented architecture we have used both ordinary clock and dual edge triggered clock. Both schemes take same time to give the output but we got a reduction in power consumed by the architecture with dual edge triggered clock implementation.

VII. EXPERIMENTAL RESULTS

We have experimentally proved that the dual edge triggered technology decreases the power consumption of the CPU considerably. A simple pipelined datapath was designed with the help of Verilog HDL. Another one with dual edge triggered clock was also designed in Verilog HDL and these two designs were successfully simulated using ModelSim. We observed that the output (Timing diagram) of both the designs were similar as expected. From this observation, we can infer that the throughput of the system is never affected by changing the clocking scheme from single edge clock triggering to dual edge triggering. These designs were then synthesized using Xilinx ISE 10.1. The inbuilt power analyzing tool was used to analyze the power consumption by the designs. A comparative study of the power consumption by both the designs, one with the conventional technology and the other with the proposed technology was made. From Fig. 7 it can be noted that the power consumption has reduced almost by half. We analyzed and plotted the power consumption of the two technologies against different clock frequencies. Fig. 7 shows the power consumption versus frequency graph for both the designs. In Fig. 7, Design 1 refers to the datapath using

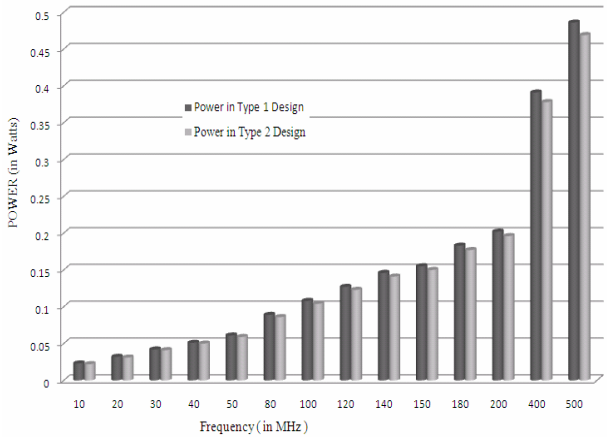

Figure 7. Graph showing the comparative power analysis with power consumption plotted against clock frequency.

single edge triggered clock and Design 2 refers to the datapath using dual edge triggered clock. We can clearly observe the decrease in power consumption using the proposed technology. For example, the power consumption by Design 1 at 200 MHz is 0.202 Watts. Since the frequency of the new scheme has halved, the corresponding power consumption of the new scheme is at 100 MHz, which is 0.104 Watts. In this case 48% power has been reduced.

From the graph it can be seen that as the frequency increases, power reduction percentage also increases. Let us consider a frequency in higher range, power consumption by design 1 at 400 MHz is 0.391 Watts. The corresponding power consumption of the new scheme at 200 MHz is 0.196Watts. In this case 49.5% power has been reduced which strongly supports our statement.

Fig. 8 shows the frequency versus power consumption graph for input signal frequency half that of clock frequency and 100% toggling. The power consumption by Design 1 at 80 MHz is 0.296 Watts. Since the frequency of clocking of the new scheme has halved, the corresponding power consumption of the new scheme at 40 MHz is 0.154 Watts. In this case 49% power has been reduced.

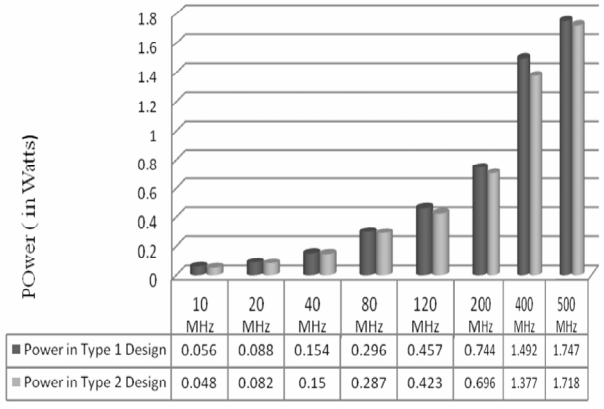

Figure 8. Graph showing the comparative power analysis with power consumption plotted against clock frequency for the input signal frequencyis half that of clock frequency and signal is toggled 100%

VIII. CONCLUSION

In this paper, we used two clocking schemes for this pipelined architecture and analyzed the power for each using simulation and power analysis tools. The first scheme was a single positive edge triggered clock and the second was using a dual edge triggered positive-negative clock. Both datapath designs were simulated in ModelSim and we observed that both schemes provide the same output timings as expected. These two designs were synthesized and power analysis was done in Xilinx ISE 10.1. Power consumption was observed for varying clock frequencies and signal toggling in both the designs and frequency-power graphs were plotted. It was observed that the power consumed in the dual edge triggered clock was less than the single edge triggered clock; nearly 50% power reduction. So the implementation of the dual edge triggered clock in the RISC pipelined architecture will decrease the power consumption without sacrificing throughput.

ACKNOWLEDGMENT

We gratefully acknowledge the Almighty GOD who gave us strength and health to successfully complete this venture. The authors wish to thank Amrita Vishwa Vidyapeetham, in particular the Digital library, for access to their research facilities.

REFERENCE

[1] John.L.Hennessy, David.A.Patterson, “Computer Architecture – A quantitative approach” 4th edition.

[2] Ziad Youssfi, Michael Shanblatt, “A New Technique to Exploit Instruction-Level Parallelism for Reducing Microprocessor Power Consumption”.

[3] Hidehiko TANAKA “Toward More Advanced Usage of Instruction Level Parallelism by a Very Large Data Path Processor Architecture”.

[4] Donald Alpet, Dror Avnon “Architecture of the Pentium Microprocessor”.

[5] Shan Atukorala “Branch Prediction Methods Used in Modern Superscalar Processors”. 0-7803-3676-3/97, International Conference onInformation, Communications and Signal Processing ICICS ’97, Singapore, 9-12 September 1997.

[6] J.E.Smith, “A study of branch prediction strategies”, Proc. 8th Int. Sym. on Computer Architecture, pp. 135-148, Minneapolis, Minnesota,May 1981.

[7] G.S.Tyson, “The effects of Predicated execution on branch prediction”, Proc. 27th Int. Sym. on Micro architecture, pp. 196-206, San Jose, Calif., Dec. 1994.

[8] S. McFarling and J.Henessy, “Reducing the cost of branches”, Proc.13th Int. Sym. on Comp. Arch., Tokyo, Japan, pp. 396-403, June 1986.

[9] Ramon Canal, Antonio Gonzalez, James E.Smith “Very Low Power Pipelines using Significance Compression”, 0-7695-0924-X/00.

[10] P.Chang, E. Hao, T. Yeh and Y.Patt, “Branch classification: a new

[11] “Mechanism for improving branch predictor performance”, Proc. 27th Int. Sym. on Micro architecture, pp. 22-31, San Jose, Calif., Dec. 1994.

[12] Unnikrishnan R. Nair, Donna J. Quammen, Daniel Tabak “Superscalar Extension for the Multris Processor”, 1089-6503/97.

[13] Vishwanadh Tirumalashetty and Hamid Mahmoodi,”Clock Gating and Negative Edge Triggering for Energy Recovery Clock”, 1-4244-0921-7/07.