Low-Power Pipelined MIPS Processor Design

Gautham P, Parthasarathy R, Karthi Balasubramanian
Department of Electronics and Communication, Amrita School of Engineering,
Amrita Vishwavidyapeetham
Kollam, India
gauthampp@yahoo.co.in, partha_parthu@yahoo.co.in, karthib@amritapuri.amrita.edu

***Abstract*—This paper presents the design and implementation of a low power five-stage parallel pipelined structure of a MIPS-32 compatible CPU. The various blocks include the data-path, control logic, data and program memories. Hazard detection and data forwarding units have been included for efficient implementation of the pipeline. A modified architecture is proposed that leads to significant power reduction by reducing unwanted transitions. Verilog design followed by synthesis on to Xilinx spartan-3E FPGA was done. On-chip distributed memory of Spartan-3E was used for the data and the program memory implementations.**

***Index Terms*—MIPS Processor, Parallel Pipeline, Low Power Architecture.**

I. Introduction

MIPS processor design is based on the RISC design principle that emphasizes on a load/store architecture. Due to the difference in time taken to access a register as compared to a memory location, it is much faster to perform operations in on-chip registers rather than in memory. To eliminate the latency of memory operations, MIPS processor uses the load/store architecture where the access to memory is only through load and store instructions. The processor has many registers and operations are performed in data present in those registers [1]. The basic architecture of a MIPS processor is shown in Fig. 1.

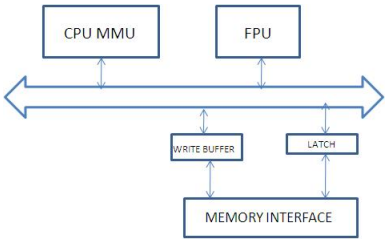

Fig. 1. MIPS Top Level Structure

When talking about the characteristics of MIPS processors, it is very important to differentiate between the terms 'architecture' and 'hardware implemenation'. Architecture refers to the instruction set, registers, address layouts etc. while hardware implementation refers to the manner in which different processors use the architecture to build their own model. The architecture remains the same for all MIPS based processors while the implementations may differ. Different models like single cycle, multi-cycle and pipelined designs have been used for designing the same. The pipelining can further be implemented as a Parallel pipeline, Superpipeline or as a Superscalar pipeline [1] .

Lot of literature is available on MIPS processor design. Patterson and Hennessy [2] have discussed the basic architecture of the above mentioned three models. Reaz et al. [3] have detailed the hardware implementation of a single cycle based design. In our paper, the design and implementation of the processor based on parallel pipelining method has been explored. Modifications have been introduced in the datapath to reduce dynamic power dissipation. Kamran et al. [4] have talked about reducing power during pipeline stalls. During these stall periods, NOPs are introduced which do not do useful work but power may be dissipated if data transitions take place. For power reduction, they have proposed to feed-back the previous data to the pipeline during these periods and prevent toggling of data, thus reducing dynamic power dissipation. This method does acheive power reduction but only during pipeline stall stages, which may or maynot occur frequently depending on the compiler efficiency. In our design power reduction is acheived through by-passing pipeline stages that cause unneccessary switching activity. One of the influencing factors of Dynamic power dissipation is switching activity and dynamic power is given by the equation,

$$P = 0.5C(V_{dd})^2 E(sw) f_{clk} \quad (1)$$

Thereby decreasing switching activity (E(sw)) results in reduced dynamic power consumption.

The top level structure along with the supported instruction set and the implementations of the different modules are discussed in section II. Results are presented in section III and the paper concludes in Section IV.

II. Architecture

The instruction set of MIPS consists of three types namely Register, Immediate and Jump. Patterson and Hennessey [2] gives the complete list of instructions belonging to these types. A subset of the instruction set has been implemented in our design, the list of which is given in Fig. 2

The pipelined architecture [2] of the processor is given in Fig. 3 where the different stages of the pipeline are seperated by pipeline registers.

The major building blocks of the design include

1) Memory and Register Blocks
2) Datapath
3) Control Logic

INSTRUCTION TYPE	INSTRUCTION
R TYPE	AND,OR,XOR,ADD,SUB,NOR,SLT
I TYPE	ANDI,ORI,XORI,ADDI,SLTI, SHIFT LEFT, SHIFT RIGHT
	LW,SW
	BRANCH
J TYPE	JUMP

Fig. 2. Implemented Instructions

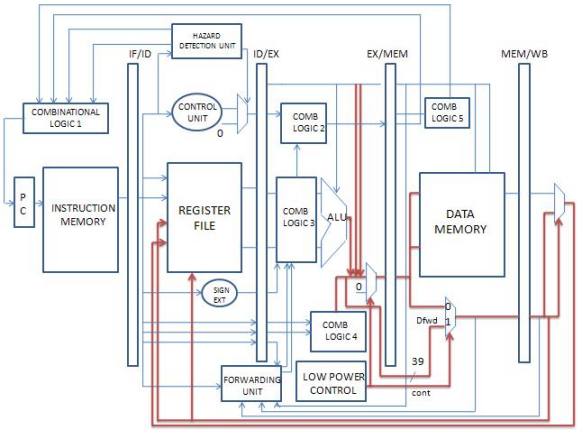

Fig. 3. Top Level Pipelined Architecture (adapted from [2])

4) Data Forwarding Unit
5) Hazard Detection Unit
6) Power Reduction Unit.

Memory and Register Blocks

The data and instruction memories are 32 by 256 bytes wides and 32 by 1024 bytes wide respectively while the register block is of size 32 by 32 bytes. The instruction memory is implemented as a single port on-chip distributed ROM while the data memory is implemented as a single port on-chip block RAM inside the FPGA.

Datapath

The pipeline stages for different type of instructions mentioned above are shown in the Fig. 4

In the fetch stage, instructions are fetched at every cycle from the instruction memory whose address is pointed by the program counter (PC). During the decode stage, the registers are read from the register file and the opcode is passed to the control unit which asserts the required control signals. Sign extension is also done for the calculation of effective address. In the execute stage, for Register type instructions, the ALU operations are performed according to the ALU operation control signals and for load and store instructions, effective address calculation is done. The load and store instructions write to and read from the data memory in the memory stage while the ALU results and the data read from the data memory are written in to the register file by the register type and load

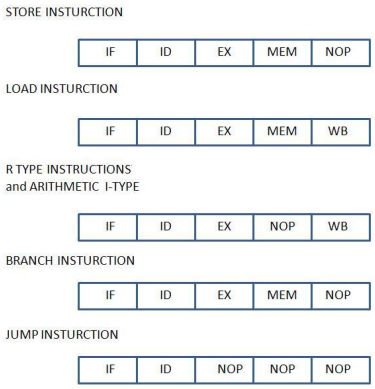

Fig. 4. Pipelined Representation of Instructions

instructions respectively in the write-back stage.

The data-path contains pipeline registers in-between each of the stages. These registers are used to carry-over results from the previous to the following stages.

Control Logic

The pipeline is controlled by setting control values during each pipeline stage. Each control signal is active only during a single pipeline stage and hence the the control lines can be divided according to the five pipelined stages. These signals will be forwarded to the adjacent stage through the pipeline registers.

Data Forwarding

Data hazards arise from the dependence of one instruction on an earlier one in the pipeline. To avoid these hazards, data needs to be forwarded which is done by the forwarding unit. Fig. 5 shows the graphical representation of forwarding [2].

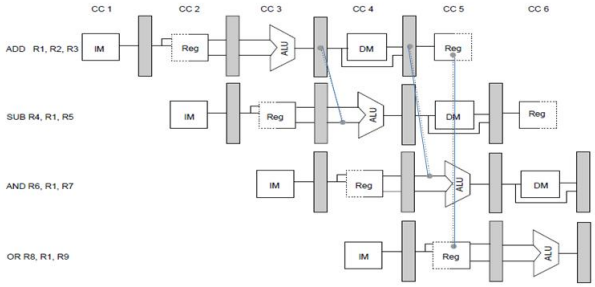

Fig. 5. Graphical Representation of Data Forwarding (taken from [2])

Hazard Detection

Data forwarding cannot prevent all pipeline stalls. When a data requested by a load instruction has not yet become available it leads to load-use hazards. To resolve this hazard, the pipeline is stalled by the unit for one stage and then continues with the forwarding of data. Fig. 6 illustrates the load-use hazard situation [2].

Power Reduction Unit

One of the key concern in any microprocessor based system is power consumption. Power dissipation is either static or dynamic. Static power dissipation is caused due to leakage and short circuit currents while dynamic power dissipation is due

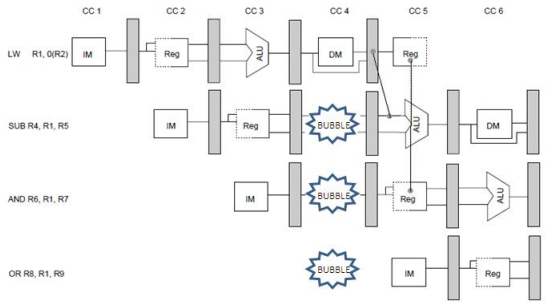

Fig. 6. Data Forwarding with a Stalled Pipeline stage for a 'Load-Use' Hazard (taken from [2])

to switching activity of the various transistors in the circuit. Dynamic power forms the major chunk of power dissipation in CMOS circuits and hence require a lot of attention. The proposed power reduction circuit intends to reduce dynamic power by reducing unwanted transitions.

Fig. 7 shows the normal pipeline stages. It can be seen that the data memory stage of the pipeline is not used by any of the arithmetic instructions. Transitions during this unused state cause extra power dissipation. To avoid this wastage, the pipeline is reconfigured to bypass this stage for these set of instructions. Hence data obtained from the execution stage is forwarded directly to the write back stage. During this time, the EX/MEM pipeline registers are maintained at zero value thus ensuring that no transitions take place and power dissipation is reduced.

This is possible for all arithmetic instructions that occur after a store/ branch/jump instruction. Arithmetic instructions have a 'NOP' stage in the mem-stage while there is a 'NOP' during the write-back stage for the store/ branch/jump instructions. Hence the write-back stage of the arithmetic instructions can be moved to the mem-stage without causing any resource conflicts. This is illustrated by arrows in Fig. 7.

This bypassing of data can be continued till a Load instruction is encountered. A Load instruction uses all five stages of the pipeline and hence a resource conflict will arise as shown by crossed-arrows in Fig. 7. So, data has to pass through the regular pipelined structure till it encounters a store/branch/jump instruction after which the reconfigured pipeline can be again brought in.

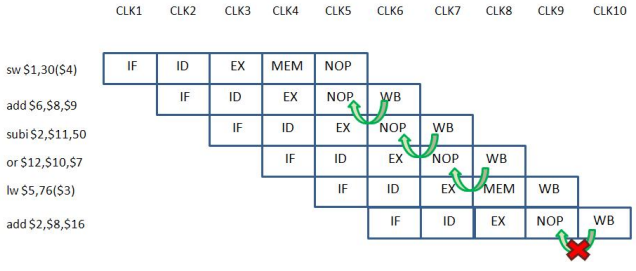

Fig. 7. Pipeline Showing Scope for Reduction of One Stage

Fig. 8 shows the reconfigured pipeline structure. Here it is seen that the three arithmetic instructions following the store instructions use the reconfigured pipeline structure. After that a load instruction comes thus reverting back to the original pipeline structure. The modified structure will again be used after the next store/branch/jump instruction.

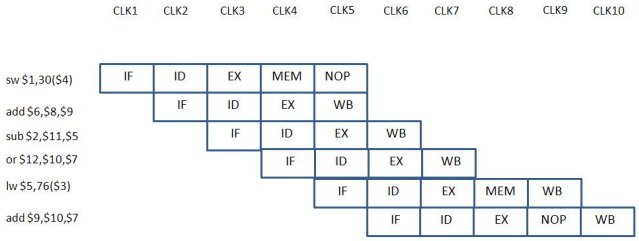

Fig. 8. Reconfigured Pipeline with Bypassing

Fig. 9 shows the design of the Power Reduction unit.

In the EX stage, all the data which are required to pass through EX/MEM pipeline register are concatenated. The concatenated data (Dfwd) includes the 32-bit output value of the ALU, 5-bit destination write address value and the two control signals that are needed in the write-back stage. This 39 bit data is given as one input to the multiplexer (mux1) that chooses between this and a 'zero' value according to the control signal received from the 'Low Power Control' unit and passes it on to the pipeline register. Mux2 receives inputs from this pipeline register and also the data that needs to be bypassed. The control signal decides which one to be sent to the next stage.

The control block generates a control signal (Cont) which acts as select line for the two multiplexers. When the control unit detects a store/branch/jump it asserts the control signal (cont) high and keeps it asserted till a load instruction is detected. During this above mentioned period, the write-back stage gets the forwarded data and the memory stage gets a constant zero value thus preventing any further transitions and reducing the amount of dynamic power dissipated. When the control signal is de-asserted, then data passes through the standard pipeline structure. The assertion and de-assertion of the control signal depends on the previous and the current instructions as mentioned above.

With a single write-port Register file, only arithmetic instructions after a store instruction can be dealt this way. With the use of a two write-port register file, this can be extended to all arithmetic instruction even when a load instruction is encountered. This is possible since two write-backs can occur simulataneously.

III. Results

Fig. 10 shows the results obtained after successful synthesis.

Power analysis using Xpower Analyzer was done for the portion of the design between the EX/MEM and MEM/WB stages. This was performed for both the standard and our modified pipelined architectures. Fig. 11 describes the results of power analysis for different clock frequencies in which data toggles at every clock cycle. In the modified pipeline

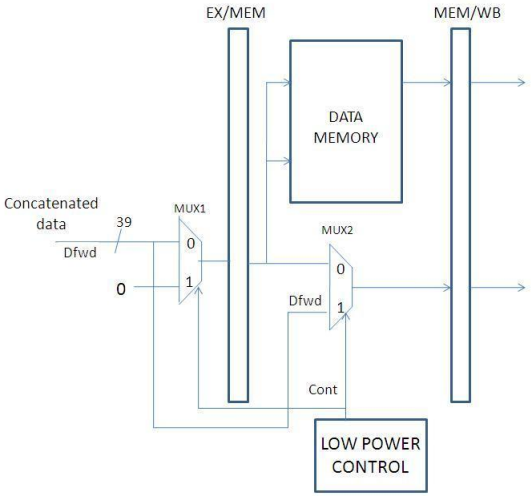

Fig. 9. Power Reduction Unit

Device	SPARTAN 3E FPGA
No. of 4 input LUT's	1890
No. of Registers	397
Maximum Operable Frequency	205.7 MHz

Fig. 10. Synthesis Results

design, power is analyzed for different toggling frequencies of the control signal. This frequency depends on the occurrence of arithmetic instructions after a store/branch/jump instruction and before a load instruction. The average power consumption over the range of measured control frequency for a particular clock frequency is also shown.

Clock (in MHz)	Data Toggle (in MHz)	Power Consumption in Normal Pipeline (in W)	Control Toggle (in MHz)	Power Consumption in Modified Pipeline (in W)	Average power consumption in Modified Pipeline (in W)
200	100	1.359	100	1.167	1.139
			75	1.155	
			50	1.143	
			25	1.130	
			10	1.123	
			1	1.119	
150	75	1.022	75	0.878	0.857
			50	0.866	
			25	0.854	
			10	0.847	
			1	0.843	
100	50	0.686	50	0.590	0.577
			25	0.581	
			10	0.571	
			1	0.566	

Fig. 11. Power Consumption Comparison Table

Fig. 12 shows the plot of power consumption versus the different clock frequencies for both the standard and the modified pipeline design. It can be noted that as the clock frequency increases, the power consumed by both the design increases. But the average power of the modified structure is less than the standard one for all the clock frequencies.

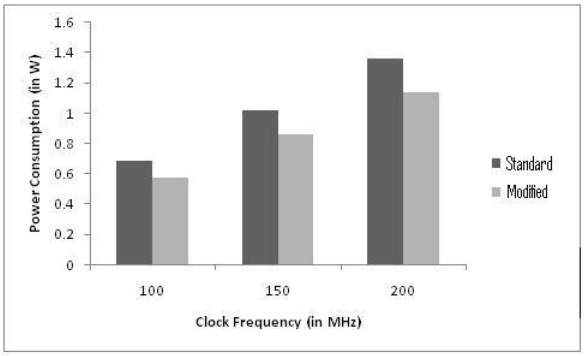

Fig. 12. Power Consumption Comparison Graph

IV. Conclusions

We have presented an architecture of MIPS processor which is capable of power reduction. The processor was successfully designed in verilog HDL, simulated with ModelSim and synthezied on to a Xilinx Spartan-3E FPGA. The design utilized 1890 four-input LUTs and 397 flip-flops and had a maximum frequency of operation of 205.7 MHz. Modifications were effected in the datapath of the design to reduce dynamic power consumption as shown graphically in the previous section.

V. Future Work

The paper presents a comparative power analysis study, but the actual power dissipation can be calculated more effectively using ASIC design tools. It is envisaged that the authors would be implementing the design using Synopsys tools for further study.

Acknowledgements

We thank Mr. Rajesh Kannan for guiding us, Aravind for providing us with resources, Parents for their constant support, Ram Kumar for inspiring us with this topic, Varrun Ramani and Vishnu for constant motivation, Shakti Prasad and Murali Krishnan for having faith in us, Vivek for his valuable comments and all KD guys and well wishers for their support. Finally we bow down to the god almighty for being the driving force behind us.

References

[1] *MIPS32 architecture for Programmers. Vol 1: Introduction to MIPS architecture*.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, The hardware/Software Interface*. Morgan Kaufmann, 2005.

[3] M. S. I. Mamun Bin Ibne Reaz and M. S. Sulaiman, "A single cycle mips risc processor design using vhdl," in *Proceedings of IEEE International Conference on Semiconductor Electronics*, Dec 2002, pp. 199–203.

[4] A. A. S. Pejman Lotfi Kamran, Amir Mohammad Rahmani and A. A. Kusha, "Stall power reduction in pipelined architecture processors," in *Proceedings of the 21st International Conference on VLSI Design*, 2008, pp. 541–546.