

Application of Power-Management Techniques for Low Power Processor Design

Sivaram Gopalakrishnan, Chris Condrat, Elaine Ly

Department of Electrical and Computer Engineering, University of Utah, UT 84112
(sgopalak@ece.utah.edu, chris@g6net.com, elaine.ly@utah.edu)

ECE 6770, May 06, 2006

Abstract—The growing demand for mobile and hand-held devices requires efficient low-power designs and methodologies providing acceptable levels of performance. This project studies the effects of power-management techniques, specifically clock-gating, for a general-purpose, MIPS-architecture processor. The implementation begins as a behavioral HDL model, and is converted to a synthesized model, for which simulated power-consumption statistics are derived; this process is then be repeated, but with low-power optimizations added. Performance is judged through simulated analysis; however, the completeness of this approach is questioned, and recommendations are provided for a more qualitative analysis, given the resources. Ultimately, this project gives insight into how the proposed low-power technique performs on such a design.

I. INTRODUCTION

General-purpose microprocessor design has traditionally concentrated on maximizing performance above other design considerations such as power or even area. This effort has resulted in great advances in microprocessor designs, achieving performance that often exceeds the needs of the user. This excess performance can often be sacrificed without much loss of functionality and usability, allowing the processors to be tailored for other applications when performance is not always the most important consideration—for example mobile and hand-held devices.

This project studies the effects of a power-management technique for a general-purpose, MIPS-architecture processor [1].

II. RESEARCH FOCUS

The MIPS-architecture is well suited for this project. The architecture is relatively old, built when

performance was not a luxury. In addition, this architecture is well documented, and appropriately complex—incorporating a 32-bit pipelined architecture, with other modern processor features, but not extremely advanced where more specialized techniques are required.

The goal of this project is to incorporate power-management techniques [2] on this architecture. We briefly review conventional power-management techniques and highlight the one that we shall focus upon in this project:

- Power-down Schemes
- Low-voltage Design
- Frequency Scaling
- Multiple Supply Voltages
- Multiple Device Thresholds
- Reducing Switching Capacitance

Power-down schemes are useful for powering down logic blocks which are not in use. This technique can be applied to complex arithmetic circuits which are used infrequently, for example a floating point unit, or divider circuit.

A reduction in supply voltage results in a quadratic decrease in power [2]; however, the delay of CMOS gates increases inversely with the supply voltage. This loss of performance can be compensated, to some degree, by architectural and logic optimizations. For example using a carry-lookahead adder in place of a ripple-carry adder increases performance, but at the same time increases area and power consumption. The carry-lookahead version can operate at a much lower supply voltage, with the equivalent performance of the ripple-carry version. This in turn translates into significant power savings.

Frequency scaling enables the processor to use different operating clock frequencies. Reducing the frequency relaxes the constraints on timing and thus allows for a reduction in voltage supply. The goal of frequency scaling is to achieve an optimal power-performance level.

As an extension to low-voltage design, using multiple supply voltages for selective portions of the design can result in a decrease of power consumption, for the same performance. Paths which finish computations early can be identified and their respective supply voltages can be suitably reduced.

Many technologies offer two types of n-type and p-type transistors, each with differing thresholds. Lower threshold devices are more appropriate for time-critical paths, as they can switch faster, while the higher-threshold devices can have lower leakage. Timing-critical paths can be identified and the appropriate threshold-scaling devices can be chosen for implementation.

To reduce the switching capacitance in a circuit, and therefore decrease power consumption, techniques such as transistor-sizing and logic/architectural optimizations can be applied.

In this project, we limit our optimizations to power-down schemes—where we identify blocks of the processor that are not always utilized, and prevent their operation, appropriately. Specifically, we use a technique called “clock gating,” which prevents the clock signal from reaching certain blocks in the processor hierarchy, effectively preventing them from switching, and therefore saving power. The benefit of such a technique is that performance is not sacrificed if performance is needed, because the blocks can be dynamically turned on and off depending on computational need. How this “need” will be determined will be the subject of later sections.

III. MIPS ARCHITECTURE

While the processor is based upon a MIPS-based architecture, the design of this processor is specific to a low-power application. Therefore, some aspects of the MIPS architecture may not be fully implemented as they are inappropriate for a low-power design. For example, floating-point instructions are

often not found in many low-power applications, and if necessary, can be emulated in software.

A. Instruction Set

This processor architecture is a simplified version of the basic MIPS architecture in order to concentrate on low-power techniques, not so much the complexity of the instruction set. The instruction set architecture implements the following classes of instructions:

- Basic arithmetic and logic
- Load/Store
- Constant-manipulation
- Branch and Jump

The basic arithmetic instructions comprise of addition, subtraction and multiplication. Division can be carried out in software, or be incorporated into the multiplication block. In this case, the division unit is built into hardware.

B. Pipelining

Modern processors incorporate pipelining to improve the performance of the processor by effectively utilizing the hardware. The MIPS uses a five-stage pipeline, starting with an instruction fetch stage, followed by a decode/register-file read stage. Then there is an execution stage in the ALU, connecting to a memory-access stage. The final stage is the write-back stage, where the resulting computation is either written to a register or to memory. Fig. (1) depicts the pipeline.

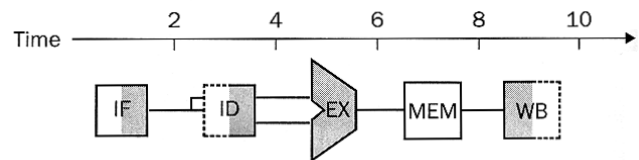


Fig. 1. Five-stage Pipeline

Multi-cycle arithmetic instructions, such as those for a multiply, are managed at the instruction/architectural level, as well as multi-cycle memory reads/writes, which also require pipeline stalling if necessary. The diagram in Fig. (2) represents the top-level block-diagram of the processor.

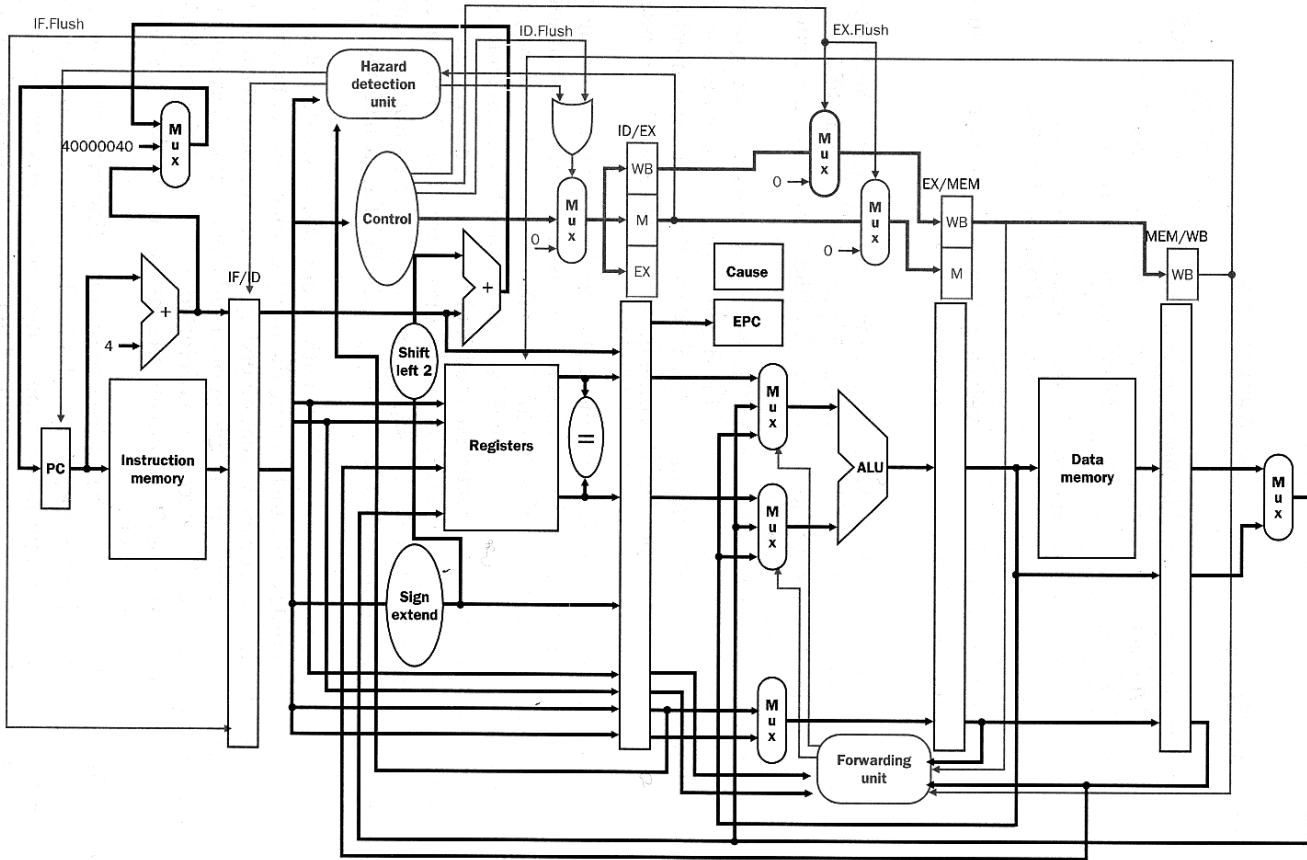


Fig. 2. Block Diagram of the processor

IV. IMPLEMENTATION

The MIPS architecture has been implemented in Verilog HDL using parametric instantiations as shown in figure 3. This implementation is a variation of the MIPS verilog core available at [3]. In [3], the core was targeting an FPGA-based implementation. We have adapted this core to a standard-cell-based implementation.

A number of modules were removed, and others required modification for use in the standard-cell implementation. Communication-interface modules such as the UART and FIFOs, were removed from the final design, as they consumed a substantial amount of area—and more importantly power. Such modules only served to distort the initial statistics unnecessarily, as they were for the FPGA-specific implementation. Other modules required modification, as they were programmed assuming an FPGA environment, specifically the presence of FPGA block-RAMs—large blocks of 1-cycle SRAM found in FPGAs—which needed to be converted into flip-

flops (registers), or emulated in a test bench.

Emulating block-RAMs was an important part of the design modification. The FPGA design incorporated the block-RAMs as the instruction cache/data memory, preloading the memory banks using memory bitmaps on compilation. This works well for an FPGA design, but not for a standard cell implementation, where an instruction cache would be implemented in a very specialized manner. Furthermore, for simulation purposes, the design required that instructions could be fed directly into the design as needed, and therefore the “instruction cache”—found in the decoder module—was removed from the design altogether, instead being emulated within the testbench.

High-level “routing” for the former instruction cache signals was implemented by passing the block-RAM signals up through the hierarchy and out the main processor modules for access by the testbench. This was required as it is not possible—in a mapped design—to directly connect to the

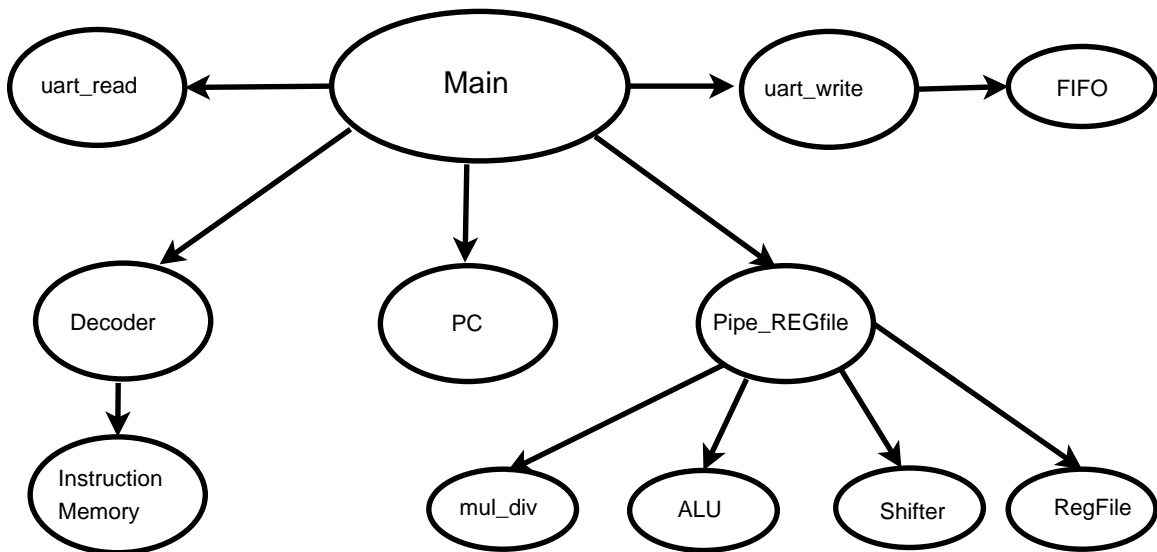


Fig. 3. Implementation of the processor

sub-module signals that provide the instructions, as can be done in structural Verilog. Providing the instructions via these routed signals was also easier using a single portmap.

The golden model design is therefore composed of the following modules:

- Mul_Div unit: This module is exercised by the multiply and divide instructions.
- ALU: This module performs the basic arithmetic and logical operations.
- Shifter: This module is used for bitwise-shifts.
- REGFile: This module contains the set of registers forming the register file of the MIPS core.

The decoder module interacts with the instructions and data and essentially implements the instruction fetch and the decode stages of the pipeline. The PC module implements the program counter. The PileREGfile module implements the other three stages of the pipeline.

Our focus primarily on the mul_div unit for implementing the optimization using the power-down scheme. Multiply and division operations are generally quite rare in many instruction streams; their probability of occurrence is often less than 10%. Despite the infrequency of the instructions, the unit used to implement them consumes a significant ratio of the total power in the processor; however, to eliminate the unit altogether and perform the operations in software would incur a severe performance penalty. This performance-power trade-off

TABLE I
POWER CONSUMPTION OF INDIVIDUAL BLOCKS

Module	Power
Pipe_REGfile	1.6647 W
MUL_DIV	549.6277 mW
ALU	247.4343 mW
Shifter	230.9180 mW

makes this unit an ideal candidate for our clock-gating optimization.

We choose to apply a clock-gating methodology to the mul_div unit, where the clock is applied to this unit only when this unit performs a non-redundant operation. We identify the bits from the instruction register that activate the operations using the mul_div unit and generate a boolean logic which can further be used for clock-gating this unit.

V. PRELIMINARY RESULTS

Some preliminary experimental results are shown in Table I. It can be seen that the module Pipe_REGfile which is mainly composed of the three other modules (mul_div, alu and Shifter) consumes as much as 1.6647 W. In this module, the mul_div module itself consumes around 550 mW. So the mul_div module consumes almost 30% of the power in the Pile.REGfile. Adding a clock gating module for the mul_div can result in significant power savings in the processor.



Fig. 4. Layout of the Golden (left) and Optimized Model (right)

TABLE II
COMPARISON OF THE GOLDEN MODEL AND THE OPTIMIZED MODEL

Design Characteristics	Golden Model	Optimized Model	Improvement
Combinational Area	3852317	3856142	-0.1%
Non-combinational Area	1988712	1988712	-
Net Interconnect Area	22376476	22442624	-0.3%
Cell Area	5841137	5844960	-0.065%
Total Area	28217504	28287478	- 0.24%
Timing	35.8	36.36	-1.5%
Power	1.4681 W	1.3223 W	+ 10%

VI. OPTIMIZATION

The `mul_div` module is utilized for the MUL and DIV instructions. Various signals are required to control the operation of the `mul_div` module, such as a function bus which determines the sign, mode and the enable signals for the `mul_div` module. In addition, there is a stop-bit which determines whether the MUL/DIV operation is completed. Only when the stop-bit is turned off, does the decoder module fetch the next instruction from the instruction memory.

An enable signal is generated to determine whether the `mul_div` module is required for the current instruction. We, therefore use this signal along with the clock to perform the clock-gating for this module.

A. Results

After performing the optimization, we ran random simulations for both the golden model and

the optimized model. The results from the random simulations are presented in table II. In this table, we present the design characteristics such as: the combinational area, non-combinational area, net interconnect area, the total areas, timing and power. It can be seen that although there is a slight increase in the areas and timing characteristics (as expected), there is significant savings in power. Hence, it can be seen that power-usage is significantly improved with negligible performance loss.

Figure 4 depict the final layout descriptions of the golden model and the optimized model, respectively.

VII. TESTING AND VERIFICATION

Figure 5 shows the structure of our test bench representing the top-level module for simulation and verification purposes. Clock and reset signals, along with block-RAM simulation-inputs and outputs, serve as the interface to the design. The block-RAM modules are simulated, in the testbench, using register-arrays. This “memory” is preloaded with the

TABLE III
POWER CONSUMPTION OF INDIVIDUAL BLOCKS

Benchmarks	Total Instructions	Mul/Div Instructions
Count	354	10/0
Calculator	460	2/1
Pi	145	5/3
Reed Solomon	1219	5/1
Dhrystone	394	1/1

data prior to simulation, via Memory Initialization File (MIF)-format files. In total, the block-RAM modules simulate four 4K-memory blocks, as were present in the original FPGA design.

Five benchmarks were chosen for simulation: a simple arithmetic calculator, incremental counter, π -calculator, Reed Solomon code generator, and a Dhrystone integer-programming benchmark. These benchmarks were provided along with the original FPGA design [3], as a series of test vectors written in assembly language. These benchmarks are then compiled into MIF files, which are then loaded into the block-RAMs for simulation.

As seen from Table III, multiplication and division operations represent an insignificant percentage of instructions. However, they do not necessarily reflect the potential effectiveness of our power optimization technique. Due to the non-linear nature of instruction streams, involving loops and jumps, instructions involving multiply and division instructions may represent a significant portion of the actual computations. Hence, these test cases are treated as generic tests to verify that instructions are handled correctly by the core. For a purpose of testing the mul-div module, we use this test cases set as a minimum test case. This proves to be a limitation in the analysis of power-usage.

VIII. LIMITATIONS AND FUTURE WORK

One of the important aspects of power calculation is to be able to determine the power consumed over some specific test-bench cases. A more qualitative power analysis can only be performed using actual simulation, as a simple analysis of sequence of instructions is not sufficient for true power-use evaluation. We can then statistically estimate the power consumption depending on the number of MUL/DIV instructions occurring in an actual

instruction *stream* as opposed to an instruction distribution as seen from simple program analysis. There are many tools available in the industry to automatically perform such estimations. However, in academia (University of Utah), resources and time are limited, and two options presented themselves as possible methods to estimate power-usage at a more qualitative level: extensive analog simulation, and pattern simulation via a dedicated tool (NanoSim).

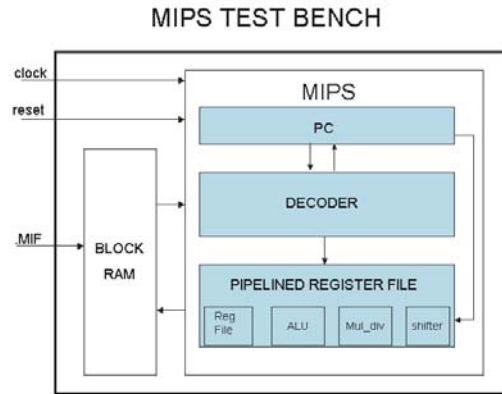


Fig. 5. Testbench platform.

A. Analog Simulation

Using the Analog Artist tool in the Cadence environment, we can run the simulation using our test-vectors and calculate power using the operating voltage, and the current drained over a period of time. However, in our case it would become an expensive operation in terms of memory and time since we have to run these simulations over a 32-bit microprocessor core. Hence, this method did not present itself as a viable option.

B. NanoSim Simulations

At the suggestion of another student, in the final design-review, we sought out a tool called NanoSim to provide power estimates. NanoSim is a tool which allows estimation of power using specific simulation patterns; however, this tool requires a lot of information about the library elements and their parameters for performing this estimation. To the best of our knowledge, the UofU_Digital_v1_1 library that is available to us—and the library used to map the design—does not have these parameter files. Also,

NanoSim accepts a vector file format (.vec), which needs to be generated from the .vcd file format of the testbench—requiring further scripts to perform the conversion. If the parameter file issue is resolved, NanoSim can ideally be used to perform the power-estimation.

C. Realistic Benchmarking

The benchmarks presented earlier do not reflect “real world” processor-usage. Most of the benchmarks are *performance* benchmarks—designed to test the arithmetic capabilities of the processor design. Benchmarks reflecting the standard use of general-purpose processor, would be useful for gauging the effectiveness of a low-power design.

IX. CONCLUSION

We have successfully demonstrated that clock-gating provides improved power-reduction performance for a realistic processor design, while negligibly affecting the performance and area characteristics of the chip. It must be noted, however, that the testing methods used to gauge the effectiveness of the power-optimization approach do not paint a full picture of the extent by which this approach has improved power-savings. One method to improve the quality of the power-savings is through simulation; however, this has limitations as well, as the type of benchmark is also important. This will be an area of future research, which due to time and resource constraints, cannot be performed for this project. However, our results do show promise, and may serve as a basis for future power-saving techniques, with more qualitative analysis on how these techniques perform in realistic environments.

REFERENCES

- [1] John L. Hennessy and David A. Patterson, *Computer Organization & Design*, Morgan Kaufmann Publishers, 1998.
- [2] Anantha Chandrakasan, William J. Bowhill, and Frank Fox, *Design of High-Performance Microprocessor Circuits*, IEEE Press, 2001.
- [3] T. Sugawara, “YACC—Yet Another CPU CPU”, <http://www.opencores.org/projects.cgi/web/yacc/overview/>.