DESIGN AND IMPLEMENTATION OF LOW POWER PIPELINED 64-BIT RISC PROCESSOR USING FPGA

VIJAY KUMAR JINDE¹, NAGARAJU BOYA², SWAPNA CHINTHAKUNTA³, RAMANJAPPA THOGATA⁴

¹, ³, ⁴Department of Physics, S. K. University, Anantapur, Andhra Pradesh, India
²Department of Physics, Intell Engineering College, Anantapur, Andhra Pradesh, India

ABSTRACT

This paper presents the design and implementation of a low power pipelined RISC processor using a FPGA (Field Programmable Gate Array). The processor design is based on the 4-stage pipelining & low power techniques in front-end design process, which is characterized by 64-bit architecture having four 64-bit registers. The RISC is designed using the Hardware Descriptive Language Verilog HDL. This paper presents the architecture, low power unit, control unit, arithmetic logic unit and instruction set of the 64-bit RISC processor. The module functionality and performance issues like area, power dissipation and propagation delay are analyzed using Altera DE2 board. 7-segment displays are connected to RISC I/O interface for testing purpose, Quartus II 10.1 suite is used for software development, Modelsim is used for simulations.

KEYWORDS: Clock gating, FPGA, RISC processor, Quartus II 10.1.

1. INTRODUCTION

The trend in the recent past shows the RISC processors clearly outsmarting the earlier CISC processor architecture. RISC is a type of microprocessor that has a relatively limited number of instructions. It is designed to perform a smaller number of types of computer instructions so that it can operate at a higher speed (perform more million instructions per second, or millions of instructions per second). Earlier, computers used only 20% of the instructions, making the other 80% unnecessary. One advantage of reduced instruction set computers is that they can execute their instructions very fast because the instructions are so simple [1].
RISC chips require fewer transistors, which makes them cheaper to design and produce. In a RISC machine, the instruction set contains simple, basic instructions, from which more complex instructions can be composed. Each instruction is of the same length, so that it may be fetched in a single operation. Most instructions complete in one machine cycle, which allows the processor to handle several instructions at the same time. This pipelining is a key technique used to speed up RISC machines.

This paper presents to minimize the power of RISC processor and clock gating technique is used in the architectural level, which is an efficient low power technique. The design considered as low power pipelined 64-bit high performance RISC processor. The various blocks of 4 stage pipelining includes Fetching, Decoder, Execution and Memory Read/Write is implemented in one clock cycle. In the design, power reduction is done in front end process so that low power RISC processor is designed without any complexity.

The most important feature of RISC instruction format is to decode the information. It has the ability to execute one instruction per cycle. This is done by overlapping the fetch, decode and execute phases of two or three instructions by using a procedure referred to as pipelining. Instructions are of fixed number of bytes and take fixed amount of time for execution. In this design, most instructions are of uniform length and similar structure, arithmetic operations are restricted to CPU registers and only separate load and store instructions access memory.

2. DESIGN OF 64-BIT RISC PROCESSOR

The architecture of the proposed low power pipelined 64-bit RISC Processor is a single cycle pipelined processor, small instruction set, load/store architecture, fixed length coding and hardware decoding and large register set. This is a general-purpose 64-bit RISC processor with pipelining architecture: it gets instructions on a regular basis using dedicated buses to its memory, executes all its native instructions in stages with pipelining. It can communicate with external devices with its dedicated parallel IO interface [2]. In the low power RISC design, all the arithmetic, branch, logical operations are performed and the resultant value is stored in the memory/registers and retrieved back from memory, when required[3].

The architecture consists of four stage pipelining: Instruction Fetch, Instruction Decode, Execute, Memory Read/Write Back [4]. The function of the instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction. Fetching instruction means the instruction present in the memory is fetched from the PC and stored it in the instruction register. The main function of the instruction decode unit is, opcode fetched from the memory is being decoded for the next steps and moved to appropriate registers. The purpose of the instruction execute is to perform required operation based on the opcode and store the result in immediate register. The purpose of the store unit is store the result into corresponding register or memory [5]. The proposed architecture of RISC processor is shown in Fig.1.
3. DESCRIPTION OF LOGIC BLOCKS

In the present work, the RISC processor consists of blocks namely, Instruction Fetch (Program Counter), Control Unit, Register File, Arithmetic and Logical Unit (ALU), Memory Unit and low power technique.

3.1. Instruction Fetch

This stage consists of Program Counter, which performs two operations, namely, incrementing and loading. The program counter (PC) contains the address of the instruction that will be fetched from the Instruction memory during the next clock cycle. Normally, the PC is incremented by one instruction during each clock cycle unless a branch instruction is executed. When a branch instruction is encountered, the PC is incremented by the amount indicated by the branch offset. The PC Write input serves as an enable signal. When PC Write signal is high, the contents of the PC are incremented during the next clock cycle. When it is low, the contents of the PC remain unchanged.

3.2. Control Unit

The control unit generates all the control signals needed to control the coordination among the entire component of the processor. This unit generates signals that control all the read and write operations of the register file, and the Data Memory. It is also responsible for generating signals that decide when to use the multiplier and when to use the ALU. It generates appropriate branch flags that are used by the Branch Decide unit.

3.3. Register File

This is a two port register file which can perform two simultaneous read and write operations. It contains four 64-bit general purpose registers. These register files are utilized during the execution
of arithmetic, data instructions and floating point operations. It can be addressed as both source and destination using a 2-bit identifier. The registers are named R0 through R3. The load instruction is used to load the values into the registers and store instruction is used to hold the address of the corresponding memory locations. When the Reg_Write signal is high, a write operation is performed to the register.

### 3.4. Arithmetic Logic Unit

The ALU is responsible for all arithmetic and logic operations that take place within the processor. These operations can have one operand or two, these values coming from either the register file or from the immediate value from the instruction directly. The operations supported by the ALU include add, subtract, compare, AND, OR, NOT, Increment, NAND and NOR. The output of the ALU goes either to the data memory or through a multiplexer back to the register file. The multiplier is designed to execute in a single cycle instructions. All operations will be done according to the control signal coming from ALU control unit.

Control unit is responsible for providing signals to the ALU that indicates the operation that the ALU will perform. The input to this unit is the 5-bit opcode and the 2-bit function field of the instruction word. It uses these bits to decide the correct ALU operation for the current instruction cycle. This unit also provides another set of output that is used to gate the signals to the parts of the ALU that it will not be using for the current operation. This stage consists of some control circuitry that forwards the appropriate data, generated by the ALU or read from the Data Memory, to the register files to be written into the designated register.

### 3.4 Memory Unit

The Load and Store instructions are used to access this module. Finally, the Memory Access stage is where, if necessary, system memory is accessed for data. Also, if a write to data memory is required by the instruction it is done in this stage. In order to avoid additional complications it is assumed that a single read or write is accomplished within a single CPU clock cycle. The architecture uses dynamic branch prediction as it reduces branch penalties under hardware control.

### 3.5 Instruction Set

A common misunderstanding of the phrase "reduced instruction set computer" is the mistaken idea that instructions are simply eliminated, resulting in a smaller set of instructions. An instruction set or instruction set architecture (ISA) is the part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O. An ISA includes a specification of the set of opcode (machine language), and the native commands implemented by a particular processor.

The instruction set used in this architecture consists of arithmetic, logical, memory and branch instructions. It will have short (8-bit) and long (16-bit) instructions shown in Table1. For all arithmetic and logical operations, 8-bit instructions are used. For all memory transactions and jump instructions, 16-bit instructions are used. It will also have special instructions to access external ports. The architecture will also have internal 64-bit general purpose registers that can be used in all operations. For all the jump instruction, the processor architecture, will automatically flushes the data in to pipeline, so as to avoid any misbehavior.
Short Instruction Format:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Long Instruction Format:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>00</td>
<td>??</td>
</tr>
<tr>
<td>Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>11</td>
<td>01</td>
</tr>
</tbody>
</table>

Table 1: Instructions Set of RISC processor

3.6 Low power Technique

There are several different RTL and gate-level design strategies for reducing power. In the present work, Clock Gating design is used for reducing dynamic power. In this method, clock is applied to only the modules that are working at that instant [6]. Clock gating is a dynamic power reduction method in which the clock signals are stopped for selected register banks during the time when the stored logic values are not changing. The clock pulse for low power technique is shown in Fig.2. The input to low power unit is global clock and its output is gated clock, since the module will block the main clock in the following conditions.

- When instruction is halt.
- When there is a continuous Nop operation.
- When program counter fails to increment.

![Fig. 2: Clock Pulses of Low Power Unit](image)

4. SIMULATION RESULTS

The simulation results have been verified by using Modelsim. The Fig.7 shows simulation results of how to reduce the dynamic power by using low power technique. The Fig.8 shows the simulation results of instruction fetch unit. The Fig.9 shows the simulation results of instruction decode unit. The Fig.10 shows the simulation results of execution unit which performs all arithmetic, branch and logical operations. The Fig.11 shows the simulation results of pipelined RISC processor using low power technique. The RTL schematic architecture shows as shown in Fig.12.
Fig. 3: Simulation results of Low Power Unit

Fig. 4: Simulation results of Instruction Fetch Unit

Fig. 5: Simulation results of Instruction Decode Unit
Fig. 6: Simulation results of Execution Unit

Fig. 7: Simulation results of pipelined RISC processor
5. FLOW CHART OF THE PROCESSOR

Start

Set initial Program Counter (PC)

Fetch instruction from instruction

Increment Program Counter (PC)

Decode from instruction register

Based on opcode instruction, executes ALU operations and Floating point unit

Stored into memory unit

Fig. 9: Flow Chart of Processor
6. CONCLUSION

FPGA based low Power Pipelined 64-bit RISC processor is designed. Modelsim is used to verify the simulation results. The design is implemented on Altera DE2 FPGA on which Arithmetic operations, Branch operations and Logical functions are verified. The proposed architecture is able to prevent pipelining to multiple executions with a single instruction. When the processor is idle, CLOCK is switched off through sleep mode by using low power technique. This design can be used for low power applications to enhance the battery life of the devices. The proposed architecture is able to prevent pipelining from flushing when branch instruction occurs and able to provide halt support.

REFERENCES

2. http://elearning.vtu.ac.in/12/enotes/Adv_Com_Arch/Pipeline/Unit2-KGM.pdf