International Journal of Electronics and Communication Engineering (IJECE)
ISSN(P): 2278-9901; ISSN(E): 2278-991X
Vol. 4, Issue 2, Mar 2015, 13-22

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

A NOVEL LOW POWER PIPELINED DATAPATH DESIGN USING PARALLELISM HAVING CONSTANT THROUGHPUT

ANSHUMAN TEWARI[1] & HARSH GUPTA[2]

[1]R. V. College of Engineering, RVCE, Bangalore, Karnataka, India

[2]Department of ECE, Manipal University Jaipur, Rajasthan, India

ABSTRACT

In this paper, we present a novel algorithm to make Pipelined Datapath architecture to be combined with another Datapath module in a Parallel fashion known as Parallel Pipelined datapath. It has the best power saving efficiency of up to 0.1125X as compared to 2.5X, at the cost of decreased throughput from 4X to 1X, while maintaining the same chip area

KEYWORDS: A Novel Low Power Pipelined Datapath Design

INTRODUCTION

The instructions of data path design starts from instruction address to the instruction memory by use of the program counter. Register operands are used after the instruction is fetched, as specified by the field of instructions. After the Register operands have been fetched, they can be operated to compute for a memory address (load or store), to compute for an arithmetic result or to compare (as in branch) instructions. Figure 1 represents the components of a basic computer design [2].

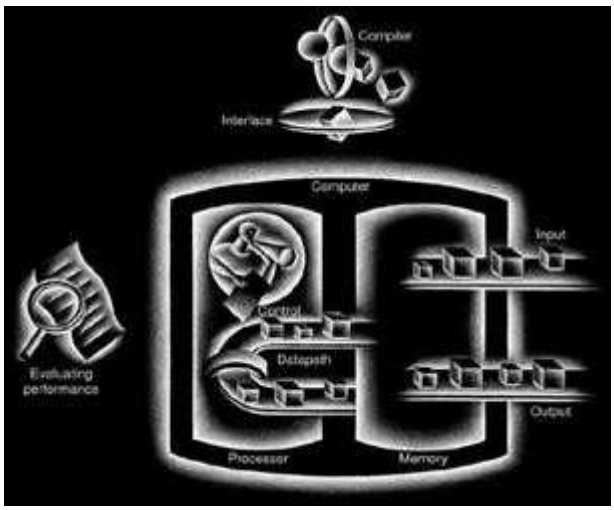

Figure 1: Components of a Computer Design

The result of an arithmetic logical instruction from ALU is written to a register file. When the operation is for a load or store, the ALU result is used as an address for store and load a value from memory into the register. At the final stage, the output of ALU and memory is written back into the register file after computation. ALU output is also used in branch instructions to determine the next instruction address coming from either the ALU (where the Program Counter (PC) and branch offset are added) or from the adder unit which increments the current PC by 4. Figure 2 represents a

simple datapath architecture where the buses (shown by thick lines) are used as interconnecting wires to join the functional units that consist of multiple signals [2]. The arrow lines are used to guide the information data flow; crossing lines are connected by the presence of a dot.

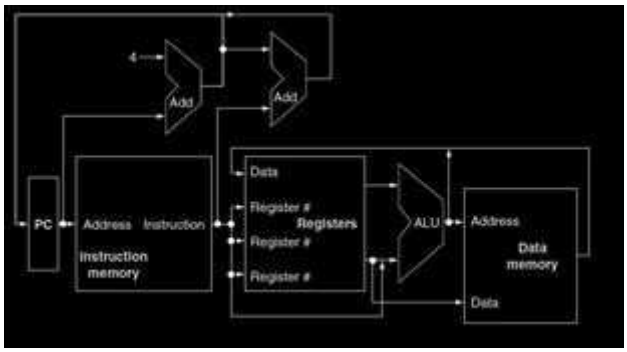

Figure 2: A Simple Datapath Design Architecture

DATAPATH ARCHITECTURE

As shown in Figure 3, the top multiplex replaces the value of PC (PC+4 or the branch destination address); the AND gate is used to control the multiplexer (select line) that together with the zero output of ALU and signal from control block (representing the branch instruction). The multiplexer unit (in middle) has the output that returns back to the register file which is used to steer the output of ALU (in case of an arithmetic logical instruction) or the output of data memory block (in case of a load for fetch). The bottommost multiplexer is used to determine whether the second ALU input is from the register file or from the offset field of instruction (an immediate operation, load or store, or a branch instruction) [2].

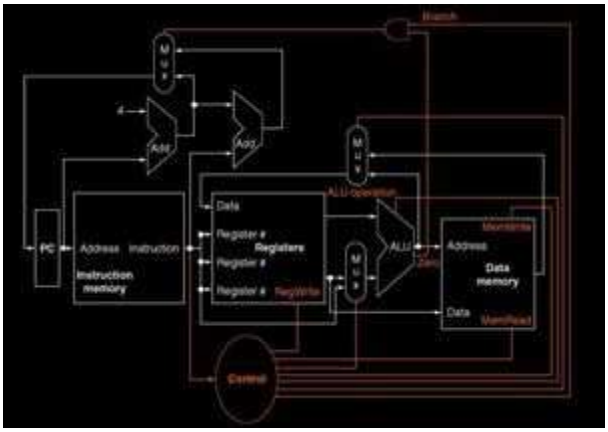

Figure 3: Basic Implementation of the MIPS Subset Including the Necessary Multiplexors and Control Lines

Control lines (shown by red color) from the control block are straightforward and determine the operation performed at ALU, such as whether the data memory should be read or write, or should the registers perform a write operation.

Figure 4 shows each datapath element explicitly: Instruction memory unit (extreme left side) to store the instructions from a program counter and supply the stored instructions at a given address. PC is used to hold the address of the current instruction. The next sequential unit is the Registers, from where the data1 and data2 are read. Following to it,

an adder unit is used to increment the PC to point to the address of next instruction. This adder, (a combinational block), can be built from an ALU and designed simply by wiring the control lines so that the control block always specifies an add operation. Another adder unit (labeled as Add) is used to indicate that it acts as a permanent adder block and cannot perform other ALU functions [2]. Last but not the least; we have the Data memory block for executing any instruction, to start fetching the instructions from the memory block. For executing the next upcoming instructions, the program counter must be incremented by 4 bits so as to point to the next instruction to be executed.

The R–format instruction reads two registers (namely, Register1 and Register2) which are later used to perform an ALU operation upon its read value, and write the results on a memory unit. The R-instruction includes *add*, *sub*, *and*, *or*, and *slt*. An instance for such as R-type instruction is *add $t1,$t2,$t3*, which reads the content of source register $t2 and $t3 and writes the result to $t1. The processor's general purpose register is used to store hierarchal structures in a register file. This register file is a collection read or writes instructions and contains the register state of the machine. An ALU is needed to operate on the values read from Registers block.

R-format instructions have 3 register operands. First of all, we read the two data words from a register file. An input to the register file specifies the register number to be read and an output correspondingly from the register file carries the value to be read. For writing a data word, two inputs are mandatory. First, specify the register number to be written with and second; supply the data to be written into that specific register location.

Register file always outputs the contents of register numbers that can be read as register inputs. A total of four inputs are needed among which three are for register numbers and one for data. Two data output are drawn from it. The input registers have five bit input each, making them 32 bit wide registers, whereas the data input and the two data output buses are each 32 bits wide. The ALU takes 32 bit inputs and produce a 32 bit outcome. One bit output for the signal, if the result is zero from an ALU [2].

Here, we considered the MIPS *load* word instruction by *lw$t1, offset_value($t2)* and the store word instruction by *sw$t1,offset_value($t2).* These instructions compute for a memory address by adding the base register address to it. If the instruction is *load*, the value read from the memory must be written into the register file and loaded in the specified register, here that is $t1.

Keeping apart the Main Control Unit design, an ALU Control block uses the function code and a 2-bit signal as its control input to get the selected output to ALU unit. Identifying the fields of instruction and the control lines for the datapath is the real challenge. In order to connect the field of instruction to the datapath, three types of instruction formats are used, namely, R-type, branch, and load/store. The instruction format is as follows:

- *Opfield*, also known as the *opcode*, is always contained in bit locations [31:26], referred as field Op[5:0].
- The two registers are specified by *rs* and *rt* fields for read positions: [25:21] and [20:16]. The same applies for branch instruction, R-type and store instruction as well.
- The base register for load and store instructions is placed in bit positions [25:21] (*rs*).
- For 16-bit offset load, branch equal and store instruction is put in positions [15:0].
- The destination register is placed in one out of the two places. For a load or fetch, it is in bit positions [20:16] (*rt*), while for an R-type instruction; it is in bit positions [15:11] (*rd*).Thus there is a need to add an additional multiplexor to select which field of the instruction is used to indicate the register number to be written [2].

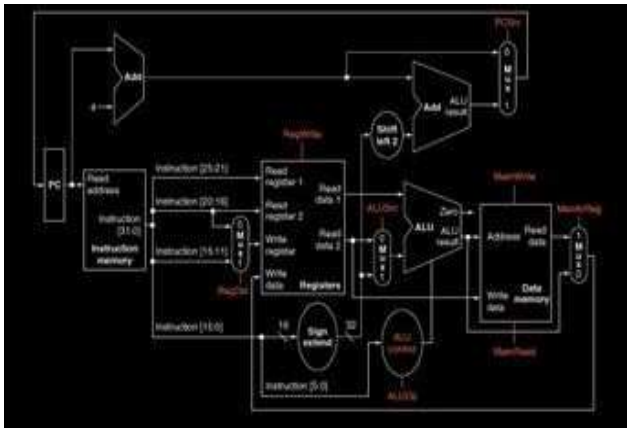

Figure 4: Datapath Design with All Necessary Multiplexors and Control Signals

Using this know-how, we add the instruction labels and an extra multiplexor (for the Write register number input of the register file) to the simple datapath shown in Figure 1. Figure 5 shows these additional blocks plus the ALU control block, the write signals for state elements, the read signal for data memory, and the control signals for multiplexors. Since all the multiplexors have two inputs, each necessities a single control line. Figure 5 also shows the seven single-bit control lines (from Control unit) plus the 2-bit ALUOp control signal to the ALU Control block

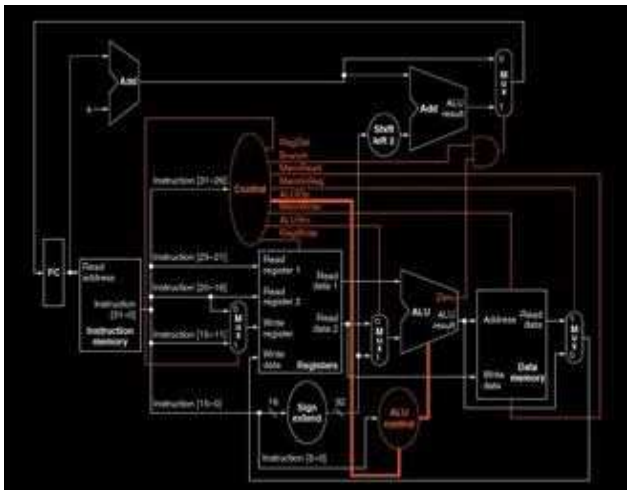

Figure 5: Simple Datapath Design with the Control Unit and All the Control Signals (Shown in Red)

PIPELINED DATAPATH IMPLEMENTATION USING PARALLELISM

Parallel processing is an important technique for reducing power consumption in CMOS circuits. The key approach is to trade area for power while maintaining the same throughput. In simple terms, if the supply voltage is reduced by half, the power is reduced by one-fourth and performance is lowered by half. The loss in performance can be compensated by parallel processing. The pipelined parallel datapath implementation is illustrated by a simple 16-bit addition mechanism where the 16-bit operand is split into 8 bits, each pipelined and combined in a parallel fashion for higher throughput at reduced power supply voltages. Figure 6 shows an example of a 16-bit adder with 16-bit inputs each (shown as A16 and B16) and resulting in a 16-bit addition output, which will be used as the basis for splitting these 16-bit inputs, each input being split into two blocks of 8-bits in a parallel fashion and then pipelined to give the resultant 16-bit output.

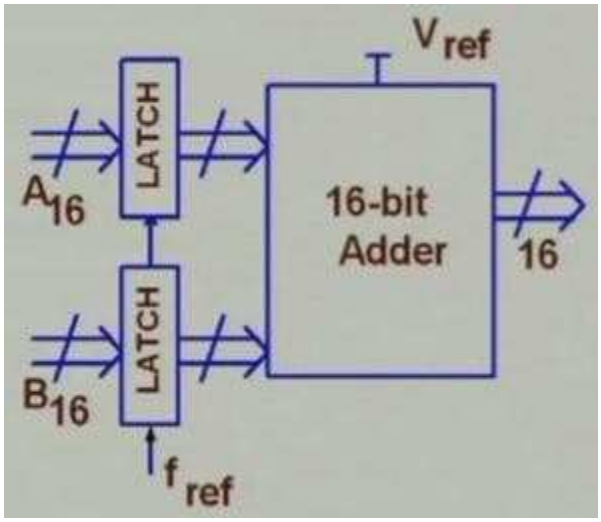

Figure 6: Example - Two 16-Bit Registers Supplies Two Operands to an Adder. Delay of the Critical Path of the Adder is 10 nsec. Operating Frequency = 100 MHz

The estimated dynamic power of the circuit:

$$Pref = Cref * (Vref)^2 * Fref \quad (1) \qquad (1)$$

- **Parallelism**

As shown in Figure 7, the adder unit has been duplicated twice, but the input registers have been clocked at half the frequency of *Fref*. This helps to reduce the supply voltage such that the critical path delay is not more than 20 nsec. Figure 8 depicts the Multi-core structure for Low power high frequency applications.

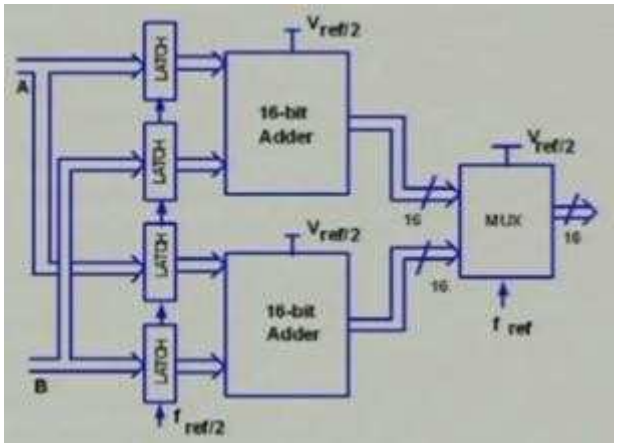

Figure 7: 16-Bit Adder Split into 8-Bit Pipelined Parallel Latch Enabled Structure

- **Impact of Parallelism**

The following equation depicts the power consumption in 'Parallel' architecture. Table 1 gives the comparison 'With' and 'Without' Vdd scaling of parallel designs [1]. The estimated dynamic power is 0.227 times *Pref*:

$$P_{par} = 2.2C_{ref}\left(\frac{V_{ref}}{2}\right)^2 \times \frac{f_{ref}}{2} \approx \frac{2.2}{8}P_{ref} \approx 0.277P_{ref}$$

Table 1: Area, Power and Throughput – 'With' Voltage Scaling and 'Without' Voltage Scaling for Parallel Designs

Parameter	Without Vdd Scaling	With Vdd Scaling
Area	2.2X	2.2X
Power	2.2X	0.227X
Throughput	2X	1X

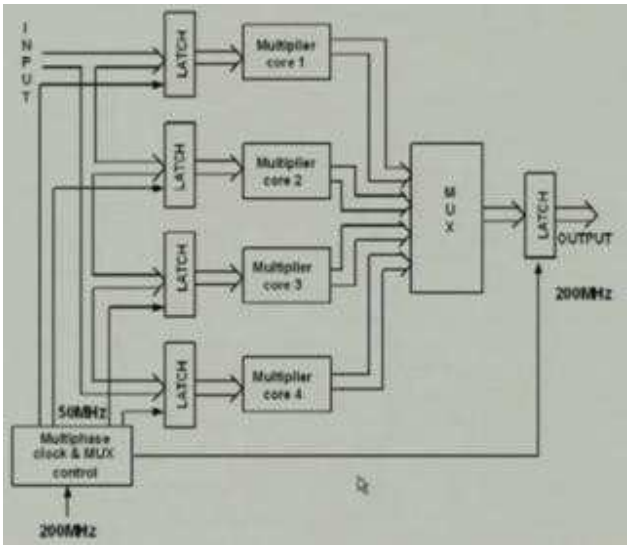

Figure 8: Example - Multi-Core Structure for Low Power

- **Pipelining**

Pipelining is an implementation technique where multiple tasks are performed in an overlapped manner. It can be implemented when a task can be divided into two or more subtasks, which can be performed independently. Figure 9, 10 and 11 depicts task division in pipelined structure.

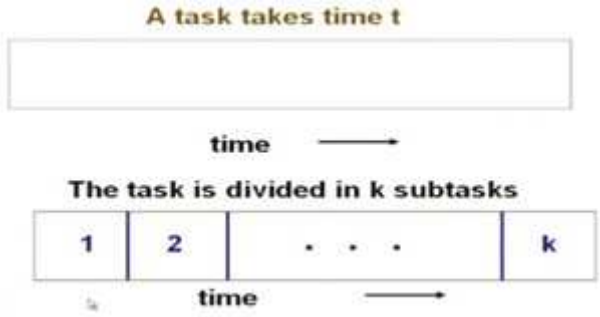

Figure 9: An Example of a Pipelined Task Divided Into Subtasks

Pipeline Implementation

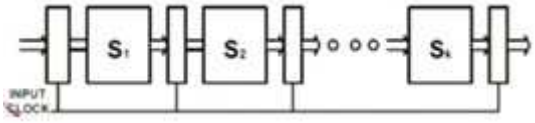

Figure 10: Different Subtasks are Performed by Different Hardware Blocks Known as Stages. Result Produced by Each Stage is Temporarily Buffered in Latches and then Passed Onto the Next Stage

More hardware resources are utilized in pipelined structure implementation [1]. Task execution in pipelined architecture is shown in Figure 11.

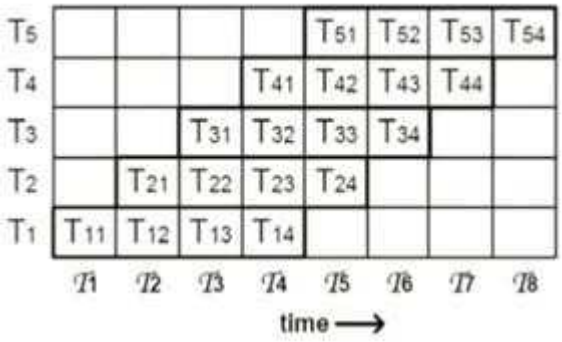

Figure 11: Task Execution in a Pipelined Implementation

Pipeline Performance Parameters

- **Clock Period**

  τ = Max { time delay of a stage }$_1^k$ + other delays

- **Frequency:** Reciprocal of the clock Period $f = 1/\tau$
- **Speedup:** k stage pipeline, n tasks

  $S_k = \frac{nk}{k+(n-1)}$ ➔ k when n >> k

- **Efficiency:** Ratio of its actual speedup to the ideal speedup [1]

  $\eta = S_k / k$

- **Throughput:** Number of tasks that can be completed per unit time

  $w = \eta / \tau$

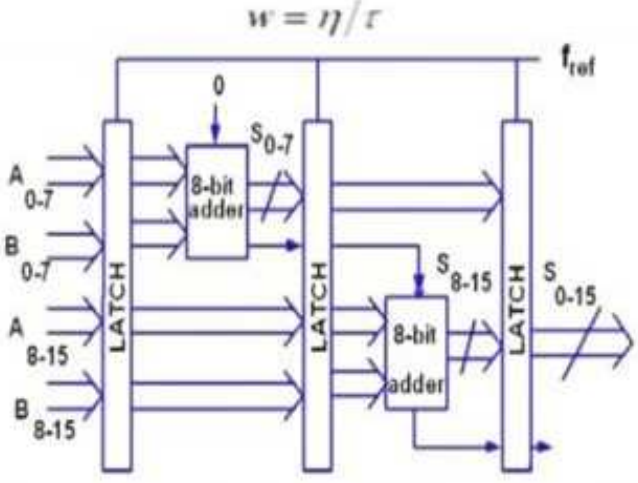

Figure 12: Pipelining for Low Power

In Figure 12 pipelined realization, instead of 16-bit addition, 8-bit addition is performed in each stage. The critical path delay through the 8-bit adder stage is about half that of 16-bit adder stage. Therefore, the 8-bit adder operates at a clock frequency of 100MHz with a reduced power supply voltage of *Vref/2*. Table 2 depicts the same.

- **Impact of Pipelining**

The estimated dynamic power is 0.28 times *Pref*:

$$P_{pipe} = C_{pipe} \cdot V_{pipe}^{2} \cdot f_{pipe} = (1.15C_{ref})\left(\frac{V_{ref}}{2}\right)^{2} \cdot f = 0.28P_{ref}.$$

Table 2: Area, Power and Throughput – 'With' Voltage Scaling and 'Without' Voltage Scaling for Pipelined Designs

Parameter	Without Vdd Scaling	With Vdd Scaling
Area	1.15X	1.15X
Power	2.3X	0.28X
Throughput	2X	1X

- **Parallel Pipelining**

Here, more than one parallel structure is used and each structure is pipelined. Both power supply and frequency of operation are reduced to achieve substantial overall reduction in power dissipation as shown in Figure 13. Table 3 depicts the Parallel Pipelined architecture strategy

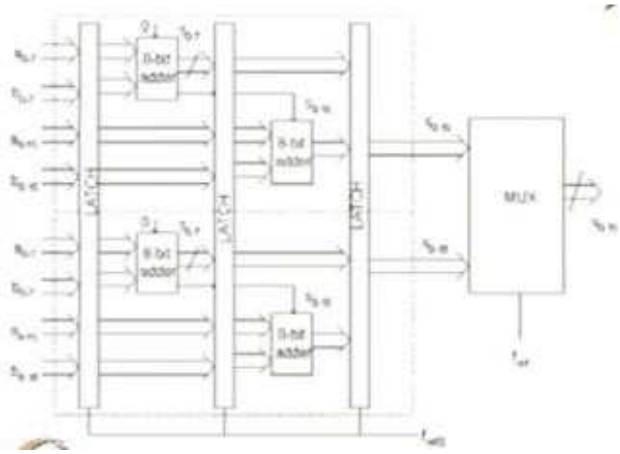

Figure 13: Combined Pipelined and Parallelism Architecture

- **Impact of Parallel Pipelined Strategy**

The estimated dynamic power is 0.1125 times *Pref*:

$$P_{parpipe} = (2.5C_{ref})(0.3V_{ref})^{2}\left(\frac{f_{ref}}{2}\right) = 0.1125P_{ref}.$$

Table 3: Area, Power and Throughput – 'With' Voltage Scaling and 'Without' Voltage Scaling for Parallel Pipelined Designs

Parameter	Without Vdd Scaling	With Vdd Scaling
Area	2.5X	2.5X
Power	2.5X	0.1125X
Throughput	4X	1X

CONCLUSIONS

The parallel datapath architecture has a power saving of 0.227X in comparison to 2.2X at the cost of reduced throughput from 2X to 1X. The pipelined datapath architecture has a power reduction to 0.28X from 2.3X with a reduced output to 1X from 2X. However, the combined datapath, which involves the combination of both pipelined and parallel architecture, has the best power saving efficiency of up to 0.1125X as compared to 2.5X, at the cost of decreased throughput from 4X to 1X, while maintaining the same chip area.

REFERENCES

1. https://www.youtube.com/playlist?list=PLTEh-62_zAfHmJE- pcjgREKiKyPSgjkxj
2. Computer Architecture, Fifth Edition: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design) Paperback – September 30, 2011 by John L. Hennessy (Author), David A. Patterson (Author), ISBN-13: 978-0123838728, Edition: 5th