Implementation of Efficient Fixed Point ALU with 32 Bit Processing Capability

Neelesh Kumar Kachhwaha[1], Prof. Sunil Shah[2]

[1]*M. Tech (VLSI Design) Gyan Ganga Institute of Technology and Sciences Jabalpur, MP India*
[2]*Dept. of ECE Gyan Ganga Institute of Technology and Sciences Jabalpur, MP India*

***Abstract*— Exploiting computational precision can improve performance significantly without losing accuracy in many applications. To enable this, we propose an innovative arithmetic logic unit (ALU) architecture that supports true dynamic precision operations on the fly. The proposed architecture targets fixed-point ALUs. In this paper we focus mainly on the precision controlling mechanism and the corresponding implementations for fixed-point adders and multipliers. We implemented the architecture on Xilinx Virtex-5 XC5VLX110T FPGAs, and the results show that the area and latency overheads are *1% ~ 24%* depending on the structure and configuration. This implies the overhead can be minimized if the ALU structure and configuration are chosen carefully for specific applications.**

The VHDL coded synthesizable RTL code of the Fixed Point Arithmetic core has a complexity. We verified the functions of the Fixed Point Arithmetic by a simulation with a single instruction test as the first step and implemented the Fixed Point Arithmetic with the FPGA.

***Keywords*—32-Processor, 32-bit fixed point Arithmetic, fixed point Processor *RISC; VHDL* ;)**

I. INTRODUCTION

The Arithmetic Logic Unit is one of the essential components of a computer. It performs arithmetic operations such as addition, subtraction, multiplication, division and various logical functions. In this paper ALU is simulated and analyzed on various parameters such as speed, power and number of logical blocks used by that ALU. The Arithmetic operations such as addition, subtraction, multiplication, division and the logical operations are realized using VHDL. Xilinx 8.1i software is used for writing the VHDL codes and the simulation is carried out with ModelSim 5.5f simulator.

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs. The inputs to an ALU are the data to be operated on operands and a code indicating the operation to be performed and status information from a previous operation; the ALU's output is the result of the performed operation. In many designs, the ALU also exchanges additional information with a status register, which relates to the result of the current or previous operations, because ALUs can be built in so many ways with wide specifications. The main objective of the project is to have a working ALU that performs different arithmetic and logic functions for all possible combinations of the inputs. The speed of ALU was not an issue and we wanted it to run at low power.

II. CIRCUIT DESIGN

This chapter gives an overview of the Hierarchy of the 32-bit ALU and its design. First, we will introduce all the different types of logic gates that has been used in the design. Then, we will give an overview of the 32-bit ALU. Finally, we will discuss the top level of the design.

Operations that can be performed:

- add (2 cycles)
- sub (2 cycles)
- mul (2-3 cycles depending on multi cycle constraint for higher speed)
- reciprocal (31 cycles but less LEs than divider)
- divider (31 cycles)
- from Int / to Int (1 cycle)
  to be extended... :)

III. PIPELINED ARCHITECTURE

Pipelining is a powerful way of improving the throughput of digital systems. The single-cycle processor is upgraded to pipelined processor by subdividing the single-cycle processor into five pipeline stages. Thus five instructions are executed simultaneously, one in each stage. Ideally, the clock frequency is almost five times faster because each stage has only one-fifth of the entire logic. Since reading, writing the memory, register file, and using the ALU typically constitutes the biggest delays in processor, the pipeline stages are chosen so that each stage involves exactly one of these slow steps.

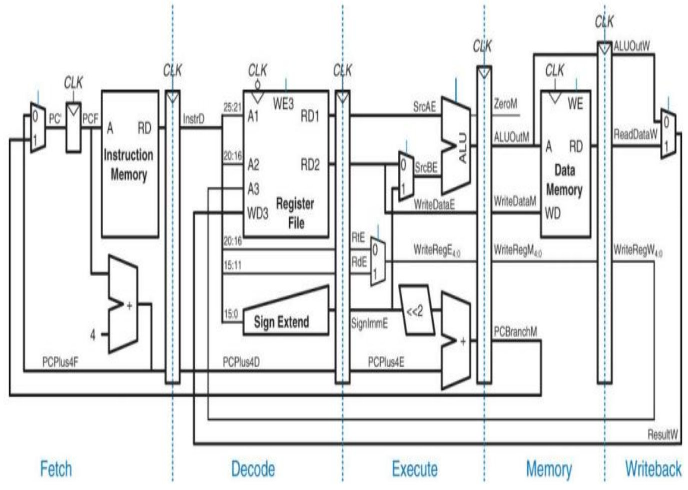
The five pipelined stages can be described as follows: Fetch: the processor reads the instruction from instruction memory.

- Decode: processor reads the source operands from the register file and decodes
- The instruction to produce the control signal. Execute: performs the computation with ALU.
- Memory: processor reads from or writes into the data memory.
- Write back: processor writes the result to the register file when applicable.

Each instruction is thus broken up into a series of steps, and several steps of different instructions are executed simultaneously, improving the throughput significantly.

IV. PIPELINED DATA PATH

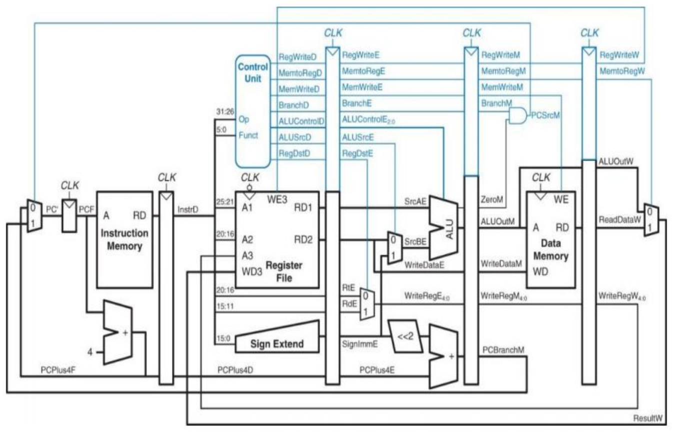
The single-cycle processor is converted into the pipelined processor by adding registers. Figure 2.6 shows the pipelined data path formed by inserting four pipeline registers to separate the data path into five stages. In pipelining, all signals associated with a particular instruction must advance through the pipeline in unison. We observe that the write back to the register file gets the data from Result W and hence, the address signal Write Reg has to be pipelined along through the memory to remain in sync.



Pipelined data path

Pipelined Control Unit Control signals for pipelined processor are same as the single-cycle processor and hence, control unit is the same. The op-code and function fields of the instruction are examined in the decode stage by the control unit to produce the control signals. They must be pipelined along with the data to remain synchronized with instruction. Figure 2.7 shows the control and data unit for pipelined architecture.

All the programs running on the MIPS use the same instruction set. Instructions indicate both the operations to perform and the operands to use. The operands may be read from memory, from registers, or from the instruction itself. Representation of the instructions in a symbolic format is called assembly language.



Pipelined processor with control.

Instruction operates on operands and these operands can be stored in registers, memory, or they can be constants stored in the instruction itself. Registers are used for quick access to operand but they hold relatively a small amount of data. Additional data can be stored in a large data memory, which can be relatively slow. MIPS is a 32-bit architecture because operands are 32-bit data.

V. SIMULATIONS AND RESULTS

In this chapter we are looking into the performance results using Xilinx ISE and XST Synthesis tools. The register transfer level (RTL) description of the micro-architecture is designed and simulated in VHDL using Xilinx ISE design suit and basic functionality is verified using the assembly codes.

VI. CONCLUSION AND FUTURE WORK

In this paper we demonstrate quad fixed point arithmetic processor with 32 bit data processing capability is implemented. Quad Fixed Point 32-bit Arithmetic Core implements a full customizable arithmetic core using the Quad Fixed Point 32-bit. Available arithmetic operations are easily configured by an generic flag. Benefits are much less area requirements lesser pipeline depth and higher speed compared to an FPU at the cost that the number range is limited from + - 2^(-24)to2^29.

The processor for this paper is built from the pipelined MIPS processor micro-architecture and is initially designed in VHDL and verified. Since the real number representation on the processor is fixed-point, the VHDL simulations are further modified with fixed-point library. The required optimization in the MIPS pipelined processor to support the wireless communication applications are studied in detail. The MIPS processor ALU is enhanced to support real numbers using fixed point arithmetic. Addition, subtraction, multiplications, and inversion are the listed operations to achieve ALU algorithms. Block wise method of implementation is employed for addition, subtraction, multiplication. Performance of Design are compared with fixed-point other simulation results. Fixed-point ALU using Newton-Raphson division and block wise analytical inversion algorithms achieve precession error in the range 10-5. The design is further synthesized and results indicate the max frequency of 101 MHz. Load word (lw) instruction is used to fetch the data into register file, which is the slowest instruction. Loading the back-to-back data from concurrent memory locations into the register file using a single new instruction is another suggested scope for improvement.

REFERENCES

[1]. Accuracy-aware processor customisation for fixed-point arithmetic Shervin Vakili ✉, J.M. Pierre Langlois, Guy Bois IET Comput. Digit. Tech., 2016, Vol. 10, Iss. 1, pp. 1–11

[2]. J. Kurzak and J. Dongarra, "Implementation of mixed precision in solving systems of linear equations on the Cell processor: Research Articles," Concurr. Comput. : Pract. Exper., vol. 19, pp. 1371-1385, 2007.

[3]. J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra, "Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems)," presented at the Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida, 2006.

[4]. J. Lee and G. D. Peterson, "Iterative Refinement on FPGAs,"in Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on, 2011, pp. 8-13.

[5]. A. R. Lopes, A. Shahzad, G. A. Constantinides, and E. C.Kerrigan, "More flops or more precision? Accuracy parameterizable linear equation solvers for model predictive control," in IEEE Symposium on Field Programmable Custom Computing Machines, Napa, California, 2009.

[6]. J. Sun, G. D. Peterson, and O. O. Storaasli, "High-Performance Mixed-Precision Linear Solver for FPGAs,"IEEE Trans. Comput., vol. 57, pp. 1614-1623, 2008.

[7]. Yiannacouras, P., Steffan, J.G., Rose, J.: 'Exploration and customization of FPGA-based soft processors', IEEE Trans. Comput.-Aided Design Int. Circuits Syst., 2007, 26, (2), pp. 266–277

[8]. Mishra, P., Dutt, N.: 'Architecture description languages for programmable embedded systems', IEE Proc. Comput. Digit. Tech., 2005, 152, (3), pp. 285–297

[9]. Lee, D.U., Gaffar, A.A., Cheung, R.C.C., Mencer, O., Luk, W., Constantinides, G. A.: 'Accuracy-guaranteed bit-width optimization', IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., 2006, 25, (10), pp. 1990–2000

[10]. Yu, P., Radecka, K., Zilic, Z.: 'An efficient method to perform range analysis for DSP circuits'. Int. Conf. on Electronics, Circuits, and Systems (ICECS), December 2010, pp. 855–858

[11]. Vakili, S., Langlois, J.M.P., Bois, G.: 'Customised soft processor design: a compromise between architecture description languages and parameterisable processors', IET Comput. Digit. Tech., 2013, 7, (3), pp. 122–131

[12]. Cong, J., Gururaj, K., Liu, B., et al.: 'Evaluation of static analysis techniques for fixed-point precision optimization'. IEEE Symp. on Field Programmable Custom Computing Machines, 2009, pp. 231–234.

[13]. Le Gal, B., Casseau, E.: 'Word-length aware DSP hardware design flow based on high-level synthesis', J. Signal Process. Syst., 2011, 62, (3), pp. 341–357.

[14]. Menard, D., Herve, N., Sentieys, O., Nguyen, H.N.: 'High-Level synthesis under fixed-point accuracy constraint', J. Electr. Comput. Eng., 2012, pp.

[15]. Vakili, S., Langlois, J.M.P., Bois, G.: 'Finite-precision error modeling using affine arithmetic'. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), May 2013, pp. 2591–2595.