Power and Area Efficient Implementation of Single Precision Floating Point Unit

Vishnu Mawandia, Gajendra Sujedia
Department of Electronics and Communication Engineering, Rajasthan Institute of Engineering and Technology

Abstract - The use of floating point unit has lot of application in real time embedded systems. Algorithms like Fast Fourier Transform(FFT) from the digital signal processing (DSP) domain often make extensive use of floating-point arithmetic. This paper presents the design and implementation of an efficient single precision floating-point processor in FPGA. This processor can be dynamically configured, loaded, and executed when needed by software applications. The system is binary compliant with the conventional microprocessor without interlocked pipelining (MIPS) architecture and the IEEE-754 standard. Here the hard-ware design is done in a way to optimize the area and delay. The design is coded in Verilog hardware description language at Register Transfer Level (RTL) and synthesized in Virtex 5 device with the help of Xilinx ISE tool.

I. INTRODUCTION

Floating-point arithmetic is widely used in many areas, especially in scientific computation; numerical processing and signal processing (like digital filters, FFT, image process-ing, etc.)[8]. The IEEE-754 defines the standard for single-precision and double-precision formats.The range & precision of numbers that can be represented using IEEE-754 format is higher than that of fixed point representation with the same number of bits.Implementation of arithmetic operations for IEEE floating-point standard in hardware becomes a crucial part of almost all processors. The applications are always look-ing for high-performance and area efficient implementation of floating-point arithmetic operation. Due to progression in VLSI technology nowadays we have FPGA’s with high speed, more embedded modules and more number of logic.These make them suitable for implementing complex applications and also we can go for improved implementation of application’s like floating point arithmetic. If the performance of floating point arithmetic in FPGA is improved, Then FPGA is a attractive platform for scientific and real time applications.By embedding our floating point processor we can easily improve the speed of floating point application. our goal is to create a flexible, generic embedded floating point processor, which over floating point applications will improve performance and save a significant amount of FPGA real estate when compared to implementations on current FPGAs. With this goal of flexibility in mind, our processor was designed so that it can be configured to perform several useful functions. Since multiplication and addition are two of the most commonly used arithmetic operations, these operations are included in the ALU, both in integer and floating-point mode.Our processor has 512 MB of data memory,256 KB of program memory,32 number of 32 bit register file,32 bit A and B register.32 bit ALU,32 bit PC(program counter),32 bit IR(instruction register).It has two modes of operation floating point mode and normal integer mode.In floating point mode operations like adder ,subtracter,multiplier and multiply-add are performed and it also handles 5 floating point exceptions.For effective implementation the ALU uses merged datapath for floating point addition and multiplication and efficient algorithm for multiplier and adder design. The design is implemented in verilog HDL and synthesized for Xilinx virtex-5 device.The design is synthesized using Xilinx ISE tool.

II. SINGLE PRECISION FLOATING POINT NUMBER

Single-precision floating-point format is a computer number format that is specified in the IEEE-754-2008 standard.fig:1 shows a single precision floating point format.where sign bit determines the sign of the number. It is the sign of the mantissa as well. Exponent is an 8 bit signed integer from -128 to 127 (2’s Complement) or can be an 8 bit unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 single precision definition. In this case an exponent with value 127 represents actual zero. The true mantissa includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1 unless the exponent is stored with all zeros. Thus only 23 fraction bits of the mantissa appear in the memory format but the total precision is 24 bits[6].

A. Processor Design

The first step in design is choosing an efficient instruction set architecture for our processor.Here we uses MIPS ISA(instruction set architecture).MIPS is a load-store RISC (Reduced Instruction Set Computer) instruction set with three operands .Remaining of the design is divided into two part

- Data path performs the data operations as commanded by the program instructions.

- Controller design controls the datapath, memory and I/O according to the program instructions.

B. Control unit design

The control unit of the MIPS single-cycle processor examines the instruction opcode bits [31:26] and decodes the instruction to generate 12 control signals to be used in the datapath.

The controller uses FSM to generate the control signal.Here the FSM consist of one initial state and 3 operating states. The states are start, fetch, decode, execute

START : All control signals are assigned to zero.

FETCH : Control signals are assigned in way to fetch the instruction from program memory. The control signal active in this state are: PCen, memread, Iren

DECODE: In this state the instruction is decoded and the datapath control signals prepared for next cycle.The control signal active in this state are:Iren, write, Bsel, ACCen, Ben.

EXECUTE: In this state the data from the file register is passed to ALU for the desired operation and the result is written back to the destination register.The control signal active in this state are: ALUfz, d memrd, d memwr, memtoreg, dstsel C. Datapath design Here the datapath is based on MIPS (microprocessor without interlocked pipeline stages). It also utilizes the features of the Harvard architecture (separate memory for instruction and data).In this scheme instructions are executed in multi clock cycles. The datapath consist of 512 MB of data memory,256 KB of program memory,32 number of 32 bit register file,32 bit A and B register.32 bit ALU with floating point support,32 bit PC register,32 bit IR register. To incorporate pipelining the datapath is clearly divided into three section(fetch, decode, execute). And operation of each section is controlled by the control signal generated from the controller.

C. Fetch unit

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 . The instruction fetch component contains the following logic elements that are implemented in Verilog: 16-bit program counter (PC) register, an adder to increment the PC by one, the instruction memory, and an Instruction register.

D. Fetch

Instruction decode unit: The main function of the instruction decode unit is to decode the 32-bit instruction fetched in previous state (fetch state) to index the register file and obtain the register data Values as seen in Figure:4 . This unit also sign extends instruction bits [15 - 0] to 32-bit. The logic elementsimplementedinVerilogincludemultiplexersanda32 bit register file,16 to 32 bit sign extender and A & B register.

E. Decode

Execution unit: The execution unit of the MIPS processor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUfz signal in the case of arithmetic operation. The branch address is calculated by adding the PC+1 to the sign extended immediate field shifted left 2 bits by a separate adder. And obtaining the address of data memory in case of load and store instruction. The logic elements implemented in Verilog include a multiplexer, an adder, the ALU and the ALU consist of datapath for floating point arithmetic.fig:5 shows the datapath for execute unit and the corresponding floating point ALU

F. Execute

The execution unit of the MIPS processor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUfz signal in the case of arithmetic operation. The branch address is calculated by adding the PC+1 to the sign extended immediate field shifted left 2 bits by a separate adder. And obtaining the address of data memory in case of load and store instruction. The logic elements implemented in Verilog include a multiplexer, an adder, the ALU and the ALU consist of data path for floating point arithmetic.fig:5 shows the data path for execute unit and the corresponding floating point ALU is shown in

ALU is a single precision IEEE-754 compliant integrated unit. It can handle basic floating point operations like floating point addition, subtraction, multiplication and multiply-add in floating point mode and 23 bit normal addition, subtraction & multiplication in integer mode of operation. The mode of operation can be indicated by 27th bit of instruction and if the bit is set to one then floating point operation is performed. ALU control signal from the controller select the desired ALU operation corresponding to the instruction. The input to the ALU is 32 bit value from A & B register. for floating point operation these are floating point numbers represented in IEEE-754 format and in the case of integer operation first 23 bit of these operand is used as input values to the ALU.

III. SIMULATION RESULT

A. Multiplier

A1(shown in diagram) is the accumulator and the value stored in accumulator is: 32’b00000000000000000000000000000001.B1(shown) is another general purpose register of 32 bit in which we have stored 32’b00000000000000000000000000000010.These 2 values are multiplied using the multiplier and the result(shown) is obtained after simulation is 32’b0000000000000000000000000000010 which correspondins to the required result.

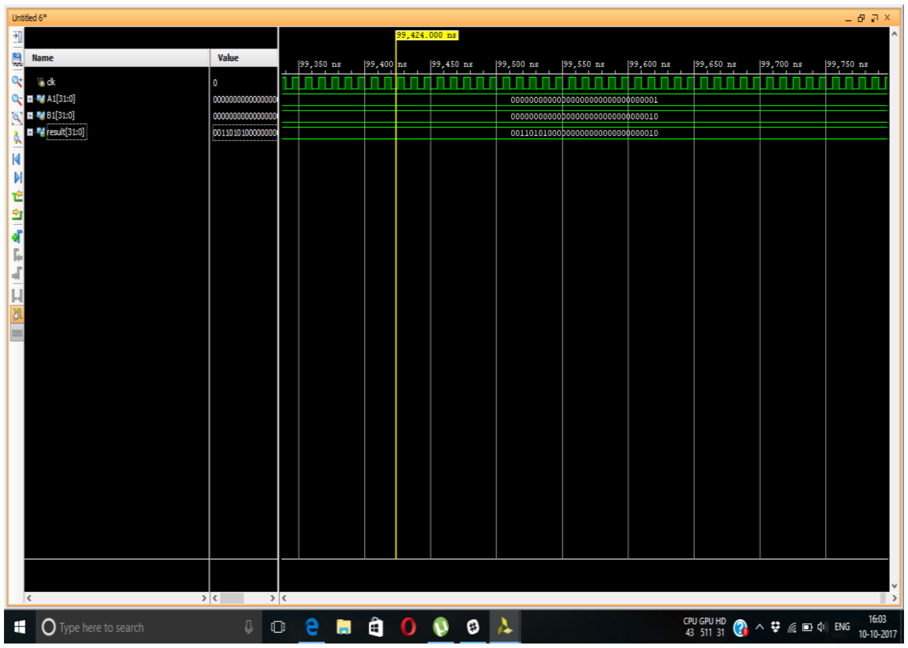

Fig 1. Simulated Waveform of Multiplier

B. Adder

The value stored in accumulator is 32’b00000000000000000000000000000001.B1(shown) is another general purpose register of 32 bit in which we have stored 32’b00000000000000000000000000000010.These 2 values are added using the adder and the result(shown) is obtained after simulation is 32’b01110101010000000000000000000000 which is obtained post normalisation and thus correspondins to the required result.

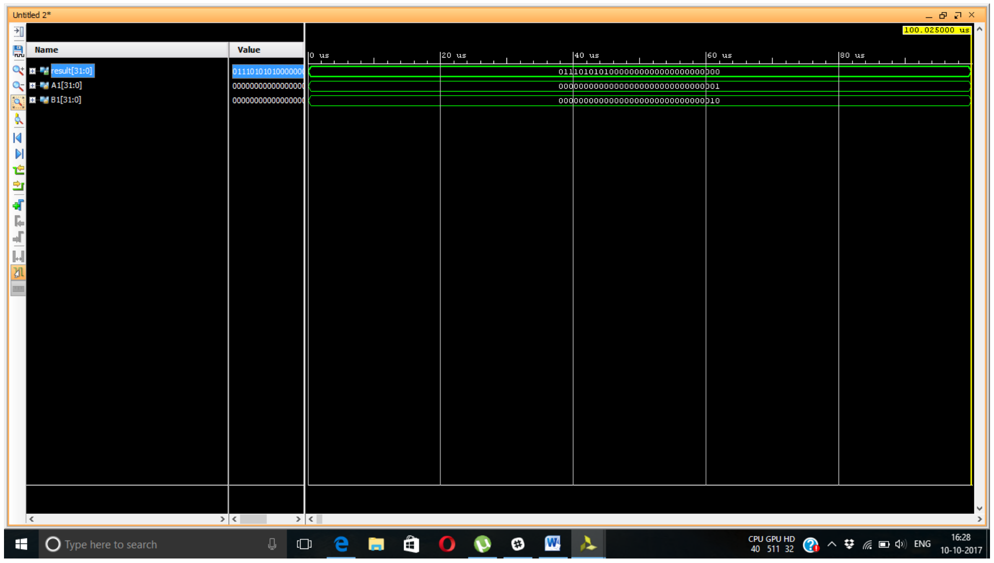

Fig 2. Simulated Waveform of Adder

C. Divider

The value stored in accumulator is 32’b00000000000000000000000000000010.B1(shown) is register of 32 bit in which we have stored 32’b00000000000000000000000000000001.These 2 values are divided using the divider and the result(shown) is obtained after simulation is 32’b11110101110000000000000000000000 which is obtained post normalisation which correspondins to the required result.

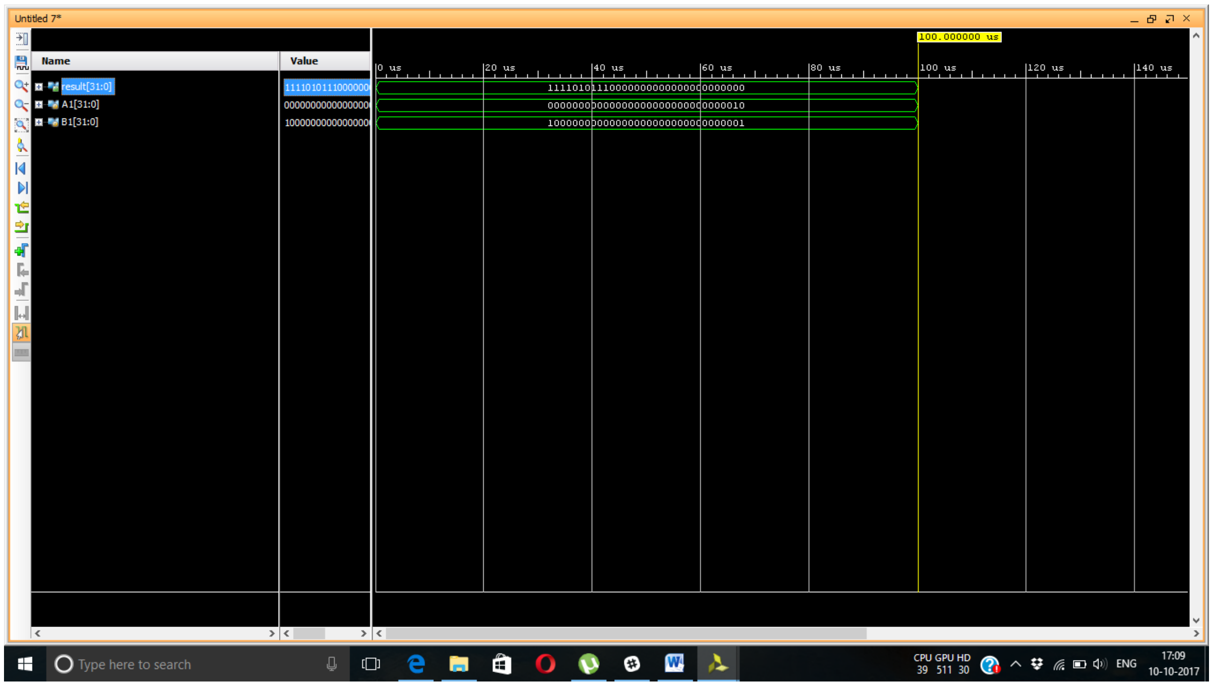

Fig 3. Simulated Waveform of Divider

Table 1: Performance comparison Table

Feature	**Earlier result**	**Optimised result**
Power	9.54 W	8.76 W
LUT used	14500	13977
Registers used	3000	2116

IV. CONCLUSION

This project deals with development of a efficient Floating Point adder, Subtractor and Multiplier for ALU in Verilog .That ALU is used to design a single precision floating point processor. Here the processor uses MIPS and Harvard based architecture. The whole design is performed with the help of Xilinx and synthesized with Xilinx tools. The experimental result shows that area and delay of the processor is reduced with the help of suitable hardware design for the datapath. Efficiency is again improved by reconfiguring the datapath for two modes of operation, integer mode and floating point mode. A simple program to add and multiply two floating point numbers is stored in program memory and corresponding floating point data is stored in data memory.

V. REFERENCES

[1]. C. H. Ho, C. W. Yu, P. H. W. Leong, W. Luk, and S. J. E. Wilton, Domainspecific hybrid FPGA: Architecture and floating point applications, in Proc. Int. Conf. Field Program. Logic Appl. (FPL), 2007,pp. 196201.
[2]. Yee Jern Chong and Sri Parameswaran, Configurable Multimode Embedded Floating-Point Units for FPGAs, IEEE Transactions 2010.
[3]. Akkas, Dual-mode quadruple precision floating-point adder, in Proc. 9th Euromicro Conf. Digit. Syst. Des. (DSD), 2006, pp. 211220. Xilinx Inc., Virtex-5 Family Overview - LX, LXT,and SXT Platforms, 2007.
[4]. P. C. Diniz and G. Govindu, Design of a field-programmable dualprecision floating-point arithmetic unit, in Proc. Int. Conf. Field Program. Logic Appl. (FPL), 2006, pp. 14. ANSIWEE std 754-1985, IEEE standard for binary Floating-point arithmetic, IEEE New York (1985)
[5]. C.W. Yu, J. Lamoureux, S.J.E. Wilton, P.H.W. Leong and W. Luk.The Coarse-Grained/Fine- Grained Logic Interface with Embedded FloatingPoint Arithmetic Units. International Journal of Reconfigurable Computing, 2008, Article ID 736203, 10 pages, 2008.

[6]. Earl E. SwartzlanderJr.,andHani H.M. Saleh, FFT Implementation with Fused Floating-Point Operations, IEEE Transactions on computers, vol.61

[7]. Steven Smith, (2003), Digital Signal Processing-A Practical guide for Engineers and Scientists, 3rd Edition, Elsevier Science, USA

[8]. John G. Proakis, Dimitris K Mano lak is,(2 003),D igitalSignaIProcessingPrincipl es,Algorithnms,Applications,4th edition, Primer, USA

[9]. "A New Common Sub expression Elimination Algorithm for Realizing low-Complexity Higher Order Digital Filters"IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 29, No.5, pp 844 - 848, May 2010

[10]. Vinay K. Ingle, John G. Proakis,(2009), DigitalSignalProcessingU singMATLAB,3e ,Cengage learning. IEEE Standards Board. "IEEE Standard for Bi- nary Floating-point Arithmetic". Technical Re- port ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronics Engineers, New York, 1985.

[11]. T. Ebisuzaki et al. "GRAPE Project: An Overviewn. Publications of the Astronomical Society of Japan, 45:361-375, 1993.

[12]. Richard I. Hartley and Keshab K. Parhi.Digit- Serial Computation.Kluwer Academic Publishers, Boston, MA, 1995.

[13]. John L. Hennessy and David A. Patterson. Com- puter Architecture A Quantitative Approach, Sec- ond Edition. Morgan Kaufmann, 1996.

[14]. NabeelShirazi, A1 Walters, and Peter Athanas. "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Ma- chines". In IEEE Symposium on FPGAs for Cus- tom Computing Machines, pages 155-162, April 1995.

[15]. Hong-Ryul Kim Todd A. Cook and LoucasLouca."Hardware Acceleration of N-Body Simulations for Galactic Dynamics". In SPIE Photonics East Con- ferences on Field Programmable Gate Arrays (FP- GAS) for Fast Board Development and ReconfigurableComputing, pages 115-126, 1995.

[16]. N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM"95), pp.155–162, 1995.

[17]. John L. Hennessy and David A. Patterson.Computer Architecture A Quantitative Approach, Second Edition. Morgan Kaufmann, 1996.

[18]. Richard I. Hartley and Keshab K. Parhi.Digit-Serial Computation. Kluwer Academic Publishers, Boston, MA, 1995

[19]. A. El Gama1 et al., "An architecture for electrically configurable gate arrays," IEEE J. Solid State Circ., vol. 24, pp. 394398, 1989.

[20]. C. Renard, "FPGA implementation of an IEEE Standard floating-point unit," Tech. Rep., Thayer School of Engineering, Dartmouth College, NH USA. Shirazi, N., Walters, A. and Athanas, P. "Quantitative analysis of floating-point arithmetic on FPGA based custom computing machines", Proc. IEEE symp.on FPGAs for Custom Comput. Machines, 1995, pp. 155-162.

[21]. Styles, H. and Luk, W. "Customising graphics applications: techniques and programming interface", Proc. IEEE Symp. on Field-Programmable Custom Computing Machines, 2000, pp. 77-87.

[22]. Ligon 111, W.B. et al. "A re-evaluation of the practicality of floating-point operations on FPGAs", Proc. IEEE Synzp.On FPGAs for Custom Comput.Machines, 1998, pp. 206-215

[23]. Louca, L. et al. "Implementation of IEEE single precision floating-point addition and multiplication on FPGAs", Proc. IEEE Symp. on FPGAs for Custom Computer Machines, 1996 Luk, W. and McKeever, S. "Pebble: a language for parametrised and reconfigurable hardware design", Field- Programmable Logic and Applications, LNCS 1482, Springer, 1998, pp. 9-18.