Title	FPGA-based floating-point datapath design for geometry processing
Author(s)	Xing, S; Yu, WWH
Citation	Configurable Computing: Technology and Applications, Boston, Massachusetts, USA, 2-3 November 1998, v. 3526, p. 212-217
Issued Date	1998
URL	http://hdl.handle.net/10722/46584
Rights	S P I E - the International Society for Optical Proceedings. Copyright © S P I E - International Society for Optical Engineering.

FPGA-Based Floating-Point Datapath Design for Geometry Processing

Shanzhen Xing and William Wing Hong Yu

Department of Industrial and Manufacturing Systems Engineering

University of Hong Kong

ABSTRACT

Geometry processing comprises of a great many computationally intensive floating-point operations. Real-time graphics systems generally use application-specific custom designed parallel hardware to provide the high performance computation power. When designing a graphics engine on a FPGA-based configurable computing system, cost-effectiveness is important. This paper investigates and proposes a cost-effective FPAG-based floating-point datapath for geometry process. It is designed to be a basic building block for FPGA-based geometry processors. The implemented datapath operates at a frequency of 6.25 Mhz and has an average floating-point operation time of 10.2 microseconds.

Keywords: FPGA, floating-point, datapath, geometry processing

1. INTRODUCTION

Graphics systems[1] generally use specifically designed geometry processors to transform object coordinates to screen coordinates for displaying the object on the screen. To display objects in the desired forms, the object data undergo a sequence of mathematical transformations down a typical processing pipeline as shown in figure 1. The geometry processor is responsible for the most computationally intensive floating-point mathematical operations. These mathematical processes transform object coordinates to screen coordinates, clip the transformed object to the boundaries of the view window, scale the object to the view port of the drawing device, relocate and orientate objects, and other such operations. The geometry processor then output the transformed data to the rasterization processor which organizes the visible data for display and output them to the frame buffer. The frame buffer data are scanned and sent to the display device.

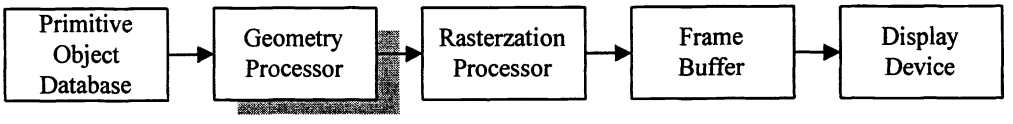

Figure 1 Computer graphics processing pipeline

The geometry processors in real-time graphics systems usually have parallel architectures with special-purpose hardware[3-6]. The custom designed hardware in the geometry processors execute floating-point mathematical operations much faster than the general-purpose processors at higher costs and with reduced flexibility. FPGA technology offers both the flexibility of configurability and custom designed hardware when designing high-performance computing systems. The special-purpose circuitry in an FPAG-based system designed for optimal performance for one application can be reconfigured and designed for another application and optimised for performance. The advantage of FPGA's reconfigurability prompts the interests of many researchers in the design and development of FPGA-based reconfigurable processors[7-9,11]. They have shown the promising potential of FPGA technology to bring application-specific performance to general-purpose computing systems. However, their work are on the design of fixed-point processors.

Part of the SPIE Conference on Configurable Computing: Technology and Applications
Boston, Massachusetts • November 1998
SPIE Vol. 3526 • 0277-786X/98/$10.00

Singh and Bellec[10] examined the implementation of rendering algorithms on FPGA devices. Their work did not apply to designing a geometry processor. Geometry processors must have the ability to operate on floating-point data. This paper presents the design and implementation of a Xilinx FPGA-based reconfigurable floating-point datapath which will be the basic building block of an reconfigurable geometry processor of a computer graphics system. Several researchers have investigated different schemes of implementing floating-point arithmetic on FPGA devices[12-13]. These papers discuss schemes such as architectures for serial and parallel operations and precision formats. No considerations have been given to designing the floating-point datapath.

2. COMPUTATION REQUIREMENTS OF DATAPATH

The geometric operations that are standard for graphics are transformation, clipping and scaling. Transformation operations are usually multiplications with 4×4 homogeneous transformation matrices. By multiplying object point vectors with an appropriate matrix effectively translates, rotates, scales or projects the object. Additions and multiplications are involved in these transformation operations. After transformations, the object is clipped according to the viewing window. The visible segments are retained and the invisible discarded. The clipping operations comprise of comparisons, additions and subtractions. The clipped object is then scaled or undergo perspective transformation to obtain the integer coordinates of the output devices prior to rasterization. The mathematical operations required are addition, multiplication and division. Thus, altogether the geometry processor datapath needs to able to perform five floating-point arithmetic operations: comparison, addition, subtraction, multiplication and division.

Before designing the datapth, the floating-point format representing the object data must be determined. The data format chosen for the design is the 32-binary floating-point format shown in figure 2. The chosen format is of the same length as the ones used in some reported custom designed processors[3-4] and has adequate range and precision for floating-point graphics data in general. The data format has a 1-bit significand sign bit *s* and 8-bit exponent. The significand is normalized 24-bit long with the leading 1-bit being implicit and having a value of 1. The 8-bit exponent adopts the biased notation, where the bias is the value subtracted from the normal unsigned representation of the data. The 32-bit floating-point format represents the data value of $(-1)^S \times (1 + significand) \times 2^{(exponent - bias)}$.

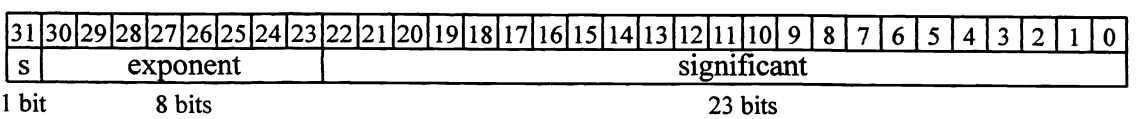

Figure2. Adopted floating-point format

The operation execution procedures dictate the design of the control unit of the datapath. The floating-point addition first compares the exponents of the two numbers and adjust the significand of the smaller number so that its exponent becomes the same as the larger number. The second step is to obtain sum of the significands of the two numbers. The final step is to normalized the sum. The floating-point subtraction is the same as the addition process except for the second step. The second step subtracts one significand from the other.

The floating-point multiplication first adds the biased exponents of the two numbers and subtracts the bias from the sum to obtain the new biased exponent. Step two is the multiplication of the two significands in a series of addition and shift operations. The next step normalizes the product and set the sign bit. Similar to the multiplication process, the floating-point division first obtains the difference of the two exponents. Step two is the significand division process which is a series of subtract and shift operations. The final step is the normalization of the quotient. Step one and step two of the multiplication operation can be executed in parallel. Likewise are the steps one and two of the division operation.

The floating-point comparison is simply subtracting both the exponent and the significand of one number from those the other.

3. ARCHITECTURE OF DATAPATH

To satisfy the computational requirements, the datapath must comprise of three major processing elements and a control unit. The three processing elements are the *exponent manipulator*, the *significand manipulator* and the *significand arithmetic unit*. The simplified organization of the these elements and the control unit is illustrated in figure 3.

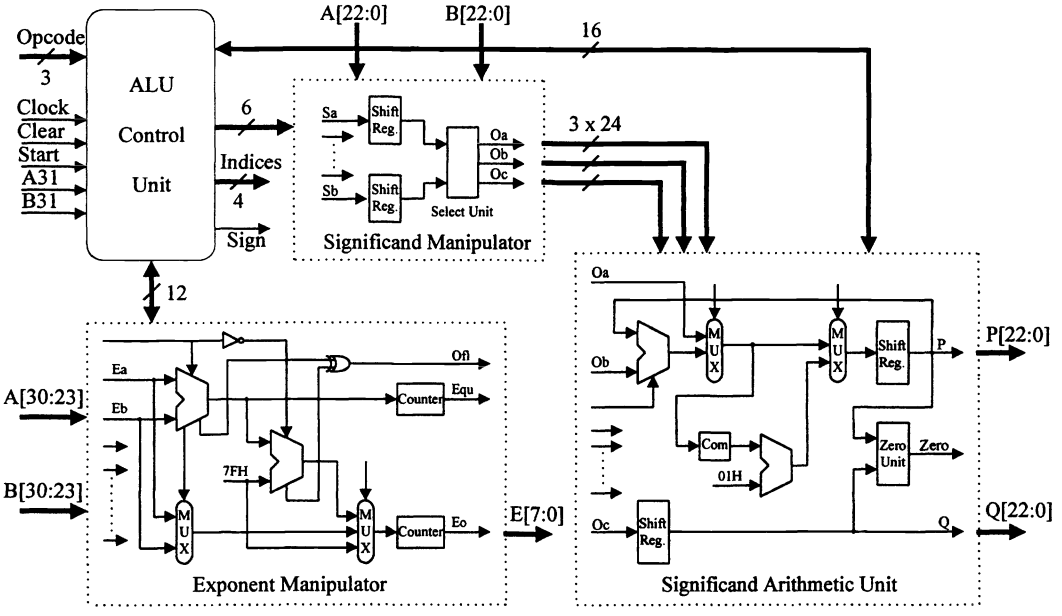

Figure 3. Simplified organization of datapath

In the cases of addition and subtraction, the exponent manipulator selects the larger exponent and signals the significand manipulator to adjust the significand of the smaller operand accordingly, so that the exponents of both numbers become equal. After the significand manipulator has adjusted the significand, it sends both significands to the significand arithmetic unit for addition or subtraction. If the result is a negative number, the arithmetic unit converts the number into a positive number and set the sign bit accordingly.

For multiplication, the exponent manipulator adds the two biased exponents and subtracts the bias from the sum to produce the correct biased sum. For division, the exponent manipulator calculates the difference of the two biased exponents and adds the bias to the difference to produce the correct biased difference. The significand arithmetic unit computes the product or the quotient of the significands. In the case of division, the remainder is loaded onto the remainder register. The exponent manipulator and the significand arithmetic unit operate in parallel.

When normalization of the result produced by the significand arithmetic unit is required, a signal is sent to the exponent manipulator to adjust the exponent accordingly.

For the comparison operation, both the exponent manipulator and the significand arithmetic unit will receive an instruction from the control unit to calculate the differences of the exponents and the significands.

Overflow and Zero may occur during operations. The overflow signal is generated by the exponent manipulator when the exponent sum or difference, during the multiplication or division operations, exceeds the 8-bit range. The overflow signal is also generated when the significand needs to be normalized while the exponent value is FFH or 00H. The overflow signal will cause the operations to be suspended.

The addition, subtraction or division operation may produce a *zero*. When that occurs, the significand arithmetic unit signals the exponent manipulator which will set the exponent to 7FH, the bias value.

4. CONTROL UNIT DESIGN AND IMPLEMENTATION

The control unit manages the passing of operands and synchronizes the events taken place in the processing elements in the datapath. In order to properly design the control unit, it is necessary to break up each operation into a sequence of execution steps or clock cycles. Breaking up the operations into execution steps allows the balancing of work done in each cycle and the shortest clock cycle to be determined. The clock cycle must be long enough to allow the execution of the longest operation step and each cycle should contain at the most one arithmetic operation step or register operation.

A high-level finite state machine shown in figure 4 is used to describe the operation of the control unit for the datapath. For simplicity, the circles in the state machine diagram may represent one to several states and contain the descriptions of the operations which generate the states. Due to the limited space for the paper, the detailed sequences of operation steps and control signals are not described in this paper. Add, Sub, Mul, Div and Com are control signals generated as a result of the addition, subtraction, multiplication, division and comparison operations. Consequential to these signals, the corresponding operation steps are executed as shown in figure 4. When the operation in the significand arithmetic unit produces a 'zero', the exponent manipulator sets the exponent to the bias value, or E = 7FH.

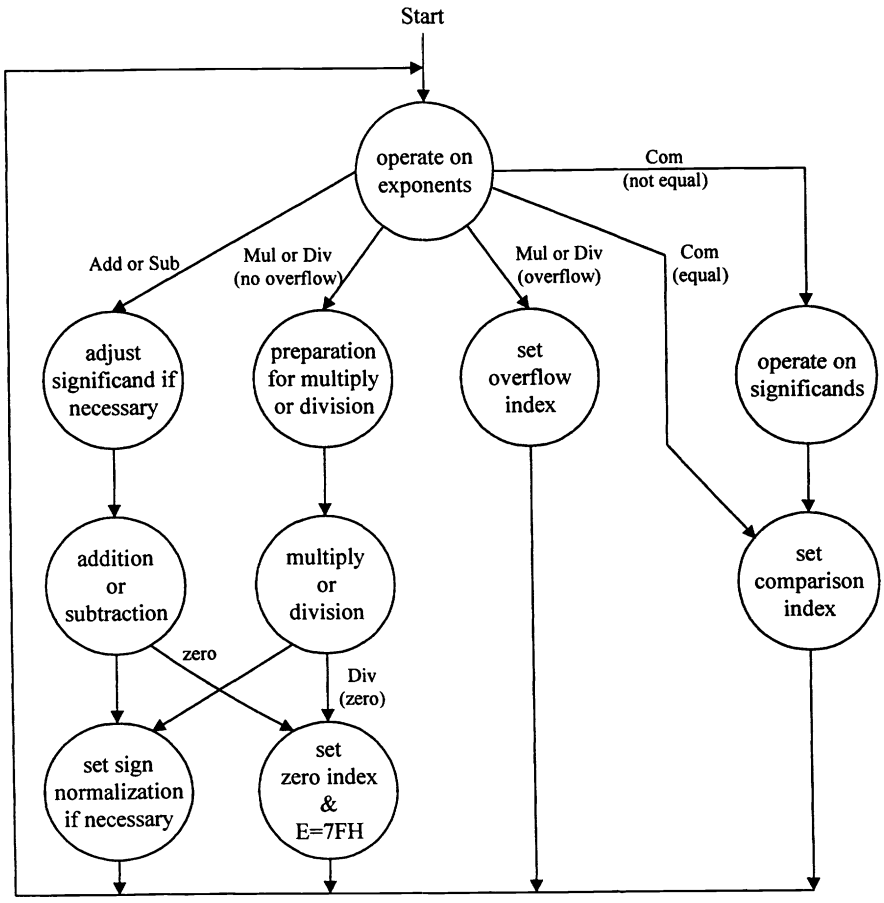

Figure 4. High-level finite state machine of control unit

The control unit as a finite state machine computes the next state and generate the appropriate control signals to activate the next operation step. One scheme for implementing the finite state machine shown in figure 4 is to develop function-specific logic circuits to explicitly determine all the next states. This scheme requires a considerable amount of hardware resources. Considering the sequential nature of many state events specified by the state machine, an alternative and more cost-effective scheme is to use a state counter to specify the next state. The scheme uses a state counter which specify the next states sequentially until a branch operation is encountered during an operation step. For the non-sequential

states, explicit function-specific logic circuits are develop to specify the next states. The control unit presented in this paper adopts the latter scheme to reduce the hardware required for the implementation of the function-specific next-state logic. Figure 5 shows the block-diagram of the control unit.

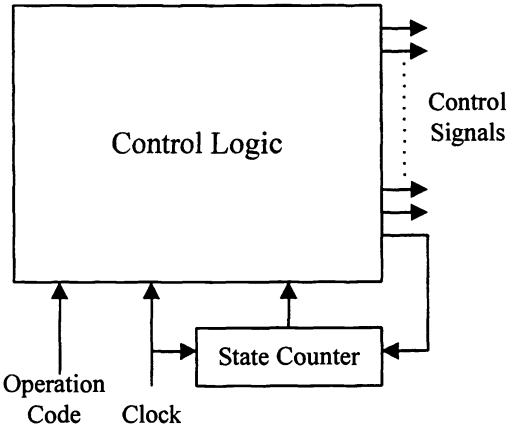

Figure 5. Control unit block diagram

To further simplify the control unit design, the multiplication and division processes which require a sequence of shift and Add / Subtract operations are separately controlled by a sub-control unit. A sub-state counter run under sub-divided clock cycles specifies the next states internal to the sub-control unit. Therefore, although the multiplication and division processes are multiple-state operations and executed in multiple clock cycles to the sub-control unit, they are considered as single-state operations by the state counter of the main control unit. When the main control unit generates the states to start the multiplication or division, the sub-control unit receives signal to start its internal sub-state counter. The control unit state counter will not generate the next states until the multiplication or division process has been completed. This proposed sub-control scheme has greatly simplified the design of the control unit and economized the cost of hardware resources.

4. DISCUSSION AND CONCLUSION

The proposed datapath has been successfully implemented on a single XC4010PQ160-5 device with a cost of 342 CLBs. The implemented datapth operates at a frequency of 6.25 Mhz and takes an average of 10.2 microseconds to execute a floating-point operation. The performance measured is device dependent. The operation speed of the proposed datapath is expected to increase with the use of a faster device. The operation speed compares favourably to those implemented on custom designed geometry processors[3,4].

However, the aim of the paper is not to design a faster geometry processor, but to underline the importance of a well-designed and cost-effective datapath to the performance of an FPGA-based geometry processor. The paper discusses the schemes used for the design of a cost-effective control unit for the datapath and the datapath organization. The high performance obtained from the implemented result suggests that the proposed datapath used as a building block for a FPGA-based geometry processor will undoubtedly contribute positively to the overall performance of the processor.

The datapath is designed for FPGA-based geometry processors, but not limited to geometry processors. It is useful as a building block for any FPGA-based 32-bit floating-point processors. The control scheme and the organisation of the 32-bit datapath can be extended to a larger, such as a 64-bit datapath. However, the costs and performance of the larger datapath will need further investigation. It is hoped that the paper will contribute to the development and the advance of FPGA-based configurable computing systems.

REFERENCES

1. J.D.Foley, A.van Dam, S.K.Feiner, and J.F.Hughes, ***Computer Graphics: Principles and Practice***, 2nd Edition, Addison-Wesley, Massachusetts, 1990.
2. D.Patterson and J.Hennessy, ***Computer Organization & Design: The Hardware/Software Interface***, Morgan Kaufmann, California, 1994.
3. J.Clark, "A VLSI Geometry Processor for Graphics," ***Computer***, July 1980, pp.59-68.
4. J.Clark, "The Geometry Engine: A VLSI Geometry System for Graphics," ***Computer Graphics***, Vol.16, No.3, July 1982, pp.127-133.
5. H.Fuchs, J.Poulton, J.Eyles, T.Greer, J.Goldfeather, D.Ellsworth, S.Molnar, G.Turk, B.Tebbs, and L.Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," ***Computer Graphics***, Vol.23, No.3, July 1989, pp.79-88.
6. G.Dunnett, M.White, P.Lister, R.Grimsdale, and F.Glemot, "The Image Chip for High Performance 3D Rendering," ***IEEE Computer Graphics and Applications***, Vol.12, No.6, November, 1992, pp.41-52.
7. C.Iseli and E.Sanchez, "Spyder: A Reconfigurable VLIW Processor Using FPGAs," ***Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines***, April 1993, pp.17-24.
8. P.French and R.Taylor, "A Self-Reconfigurable Processor," ***Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines***, April 1993, pp.50-59.
9. M.Wirthlin, B.Hutchings, and K.Gilson, "The Nano Processor: A Low Resource Reconfigurable Processor," ***Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines***, April 1994, pp.23-30.
10. S.Singh and P.Bellec, "Virtual Hardware for Graphics Applications Using FPGAs," ***Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines***, April 1994, pp.49-58.
11. N.Bergmann and J.Mudge, " An Analysis of FPGA-Based Custom Computers for DSP Applications," ***Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing***, Vol.2, 1994, pp.513-516.
12. N.Shirazi, A.Walters, and P.Athanas, "Quantitative Analysis of Floating-Point Arithmetic on FPGA-Based Custom Computing Machines," ***Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines***, April 1995, pp.155-162.
13. A.Tangtrakul, B.Yeung and T.Cook, "Signed-Digit On-Line Floating-Point Arithmetic for FPGAs," ***Proceedings of SPIE, High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic***, Vol.2914, November 1996, pp.2-13.