Using FPGAs to Prototype a Self-Timed Floating Point Co-Processor

Joe H. Novak Erik Brunvand*

Dept. of Computer Science, University of Utah
Salt Lake City, UT 84112

Abstract

Self-timed circuits offer advantages over their synchronously clocked counterparts in a number of situations. However, self-timed design techniques are not widely used at present for a variety of reasons. One reason for the lack of experimentation with self-timed systems is the lack of commercially available parts to support this style of design. Field programmable gate arrays (FPGAs) offer an excellent alternative for the rapid development of novel system designs provided suitable circuit structures can be implemented. This paper describes a self-timed floating point co-processor built using a combination of Actel Field Programmable Gate Arrays (FPGAs) and semi-custom CMOS chips. This co-processor implements IEEE standard single-precision floating point operations on 32-bit values. The control is completely self-timed. Data moves between parts of the circuit according to local constraints only: there is no global clock or global control circuit.

1 Introduction

Self-timed circuits are distinguished from clocked synchronous circuits by the absence of a global synchronizing clock signal. Instead, circuit elements synchronize locally using a handshaking protocol. This protocol requires that a circuit begin operation upon receipt of a request signal and produce an acknowledge signal when its operation is complete. Self-timed circuit techniques are beginning to attract attention as designers confront the problems associated with the speed and scale of modern VLSI technology [1]. Many of the problems associated with large VLSI systems are related to distributing the global clock to all parts of the system. In addition to avoiding these clock distribution problems, self-timed circuits can be faster, more robust, and easier to design than their globally clocked counterparts.

As part of our ongoing investigation into the suitability of self-timed design in a variety of application domains, we have designed a completely self-timed IEEE single precision floating point unit (FPU) [2] using a combination of FPGAs and semi-custom CMOS. The FPGAs' quick turn around time was essential to timely completion of the project, and the self-timed circuit style allowed the FPGA and CMOS circuits to cooperate without concern for the relative speeds of the different technologies. The floating point unit consists of an adder, fabricated in semi-custom CMOS, and a multiplier and divider, implemented with Actel FPGAs. Because we are interested in exploring the benefits of self-timed control, simple algorithms are used for the floating point arithmetic operations. However, also due to the self-timed nature of the circuits, more sophisticated algorithms could easily be used to upgrade the FPU after the prototype is evaluated. For testing and performance evaluation, the self-timed floating point unit was interfaced to a commercial computer, the Atari ST. This particular computer was chosen not for its speed, but because its cartridge port allows convenient asynchronous communication between the CPU and an external device, the FPU in this case [3]. Because the FPU is self-timed, we are able to use even a slow (8 Mhz) personal computer as a test platform for the prototype.

*This work is supported in part by NSF award MIP-9111793

2 Self Timed Circuits

A self-timed, or asynchronous, circuit does not have a globally distributed clock signal to synchronize events. Instead, events are initiated locally between parts of the circuit using handshaking signals. Synchronization of events is controlled by these local handshaking protocols. Self-timed protocols are often defined in terms of a pair of signals: one that requests or initiates an action, and another that acknowledges that the requested action has been completed. One module, the sender, sends a request event (*Req*) to another module, the receiver. Once the receiver has completed the requested action, it sends an acknowledge event (*Ack*) back to the sender to complete the transaction.

Although self-timed circuits can be designed to implement their communication protocols in a variety of ways, the circuits used to build the FPU use two-phase transition signalling for control and a bundled protocol for data paths [1]. Using two-phase signalling [4], also known as transition signalling or event logic, every transition on a control line, either from low to high or from high to low, causes an action to be taken or an event to occur. Bundled data paths are a compromise to complete self-timing in the data path. Associated with each set of data wires is a pair of transition control wires encoding a *Req* and *Ack* signal. The bundled protocol requires that the data be valid at the receiver before the *Req* transition is seen by the receiver. This single-sided local timing constraint is similar to, but weaker than, the equipotential constraint described by Seitz [4].

A variety of control circuits have been designed to be used with the Actel FPGAs [5, 6]. They include:

Merge The "OR" function for transitions, implemented by an XOR gate. The merge's output makes a transition in response to a transition on either of its inputs.

Join The "AND" function for transitions, implemented by a C-element. The C-element's output changes state only after both of its inputs change state. It is useful for synchronizing events.

Call A module that acts as a hardware subroutine call allowing multiple access to a shared resource. The Call module routes the *Req* signal from a client to the subroutine, and after the subroutine acknowledges, routes the *Ack* back to the appropriate client. The requests must be mutually exclusive.

Select A module that steers an input transition to one of two outputs based on the value of a Boolean *select* signal. The *select* signal is a bundled data signal with respect to the input transition.

Q-select A module like a select, except the *select* signal is not bundled and may be changing even when the Q-select is sampling that signal. Thus, it requires some way of sampling the *select* signal reliably. Because sampling changing signals reliably requires analog circuits, this module is approximated in the FPGA implementation using metastability information found in the Actel databook [7, 5]. A Q-select can be used in loops to act as an arbiter for concurrent events.

Toggle A module that routes input transitions alternately between two outputs.

Latch A module that latches bundled data signals upon receipt of transition control signals.

Carry Completion Adder A form of adder that reports when the addition is complete by sensing when the carry chain is complete.

Using these modules, algorithms can be implemented directly in hardware. This allows for an intuitive design methodology somewhat different than standard state machine design. Rather than designing a global control state machine, the algorithm is mapped directly to the circuits that implement it. Control, in the form of signal transitions, is passed from one part of the circuit to the next in accordance with the desired sequence of events. Each step in the algorithm starts with receipt of a *Req* signal, takes as long or as short a time as it needs to complete its task, and then acknowledges with an *Ack* signal to start the next phase of the algorithm.

Because of the self-timed organization, the functionality of the individual circuits is separated from their performance. Thus, circuit pieces in a prototype system may be constructed from technologies that have different performance characteristics. In our FPU, for example, although the multiplier and divider are implemented using Actel FPGAs, the floating point adder is implemented using a similar cell set in a MOSIS CMOS technology. If a faster overall system is desired, a new algorithm or a faster technology can easily be substituted for the existing implementation [8]. For example, to speed up the FPU, the FPGAs could be replaced by CMOS chips without retiming the system.

3 Adder

The adder is a semi-custom CMOS chip. It was fabricated in 2.0 micron CMOS through the MOSIS service and was designed using the PPL integrated circuit design software developed at the University of Utah [9]. There are six steps in the addition algorithm used in the FPU, shown in Figure 1. This is also the block diagram of the adder because the algorithm is implemented directly in hardware. The direct implementation was made possible because of the self-timed design paradigm.

The first step is to unpack the 32-bit input words to extract the signs, exponents, and mantissas from the IEEE format operands. Second, the mantissas are aligned for addition. Third, the addition is performed on the mantissa and the exponents are adjusted according to the result. Fourth, the result is normalized according to the IEEE format. Fifth, the result is rounded. All four rounding modes as defined in the IEEE standard are implemented in this circuit [2]. Finally, the result is packed into IEEE format and delivered to the output of the adder. This organization is clearly suitable for pipelining to improve the performance of the adder; although, for this first prototype, pipelineing was not used. Because the adder circuits are self-timed, adding pipelining is as simple as adding pipeline latches between each of the stages shown in Figure 1.

The adder is packaged in a 65-pin PGA. It consists of 15090 transistors in a die area of 400 mil^2. A total of 49 user I/O pins are used. The adder was tested for functionality by applying a large number of test patters to the simulation, and then using the same patterns on the fabricated chips. To measure performance a test platform was constructed that sends data to the chip as quickly as the chip can respond. Because the interface is self-timed, a simple asynchronous

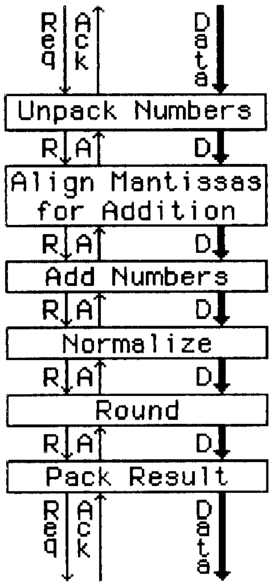

Figure 1: Adder Block Diagram

state machine can send data to the chip in response to the acknowledge from the previous data. Using this platform, this unpipelined version of the adder chip was measured at 256.4 KFLOPS.

4 Multiplier and Divider

A serial-parallel or "pencil and paper" algorithm is used for multiplication and division. This simple algorithm was used for two primary reasons. First, it is consistent with the FPU's principle of operation. That is, the project is a first attempt at fully self-timed floating point and its design should be kept simple to maximize the probability of correct operation. Second, it allows the multiplier and divider to share hardware, saving space on the FPGAs.

Figure 2 shows the block diagram of the multiply/divide unit's data path. After unpacking the mantissas and exponents from the floating point arguments, the shared adder is used to perform all required operations. In the case of multiplication, the mantissas are multiplied by shifting and adding through the P and A registers shown in Figure 2. The exponents are then added using the same adder, the result is normalized, rounded, and packed back into IEEE format. The division uses the same hardware to shift and subtract the mantissas during division, subtract the exponents, and then repack the result. Because of the sharing of hardware, this implementation is less suitable for pipelining than the adder. The multiply/divide unit works on a single operation

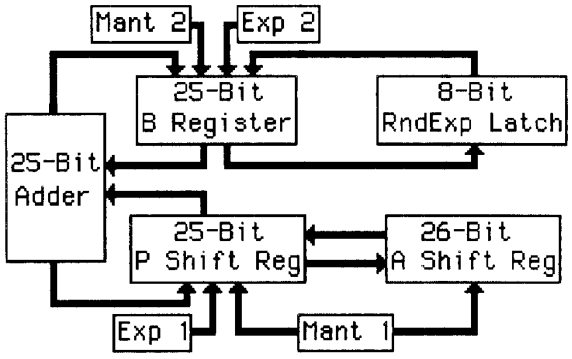

Figure 2: Multiplier/Divider Block Diagram

at a time. Future versions could use the same interface with a different implementation to improve performance. In this case, the self-timed nature of the circuit ensures that the FPU would continue to function properly, only the performance is affected.

The use of FPGAs in the multiply and divide circuitry allowed greater experimentation in self-timed design. For example, a carry completion sensing adder is used for integer arithmetic. This type of adder computes its sum in response to a *Req* signal. By sensing that every bit of the adder has correctly computed its sum and carry, the adder also generates an *Ack* signal when the sum is correct. Because the carry chain of any particular addition is likely to be short, this adder exhibits extremely fast operation on average, only slowing down in the rare cases when the carry chain must propagate along many bit of the adder.

The circuitry is contained in two Actel 1280 FPGAs. The 1280s are 1.2 micron CMOS devices in 160-pin PQFPs [7]. The first 1280 contains interface circuitry, control modules, and rounding logic. It utilizes 93% of the available logic modules (1148/1232) and 69% of the available I/O pins (97/140). The second 1280 contains the data path shown in Figure 2. It utilizes 100% of the available modules (1232/1232) and 42% of the I/O pins (59/140). As with the adder, a large number of test patterns were used during the simulation of the multiply/divide unit, and these same patterns were then used on the completed FPGAs. The test platform used for the adder to measure merformance was also used on the multiply/divide unit. This implementation using the Actel FPGAs was measured at 20.0 KFLOPS average for multiplication and 15.4 KFLOPS average for division. Note that these performance numbers are average case performance using our current data sets. The data-dependent completion time of the adder used in the multiply/divide unit means that individual operations may take slightly more or slightly less time depending on the length of the internal carry chain required by the various additions.

5 Controller

The system controller is the interface between the host computer and the FPU. It interprets the computer's instructions and directs the FPU to operate accordingly. All operands and results are transferred to and from the host computer through the controller in 16-bit words to match the cartridge interface port of the Atari ST. Also, the Atari cartridge port uses four-phase return-to-zero signalling. This is translated by the controller to the two-phase transition signalling used by the FPU. Data passed from the host computer to the FPU includes an operation code (add, multiply, or divide) and 32-bit IEEE format data passed in 16-bit words. Results from the FPU are also passed back to the host computer in 16-bit words.

The self-timed FPU's addition hardware is independent of its multiplication and division hardware. The system controller takes advantage of this fact by allowing parallel instruction scheduling. Out of order instruction completion is also allowed. An extra bit in the result lets the computer determine what type of operation completes first. The controller circuitry is also built as a self-timed circuit using the same set of circuit modules used in the FPU, and is contained in one Actel 1020A FPGA. It is a 1.2 micron CMOS chip packaged in an 84-pin PLCC. The control chip utilizes 78% of the available logic modules (436/547) and 97% of the I/O pins (67/69).

6 System Performance

Performance of the FPU as a system was measured by connecting the completed FPU to the Atari ST through its cartridge port. The Atari ST is based on an MC68000 running at 8Mhz and is thus a rather slow, albiet convenient, platform. When interfaced to an Atari ST, the adder is capable of 13.6 KFLOPS. This is considerably slower than the measured rate of 256.4 KFLOPS for the adder alone and is due to the overhead of the Atari cartridge port and the software overhead of using the FPU. The multiplier and divider operate at 10.7 KFLOPS when interfaced to the Atari ST. This is compared to 20.0 KFLOPS and 15.4 KFLOPS for multiplication and division respectively without the system overhead.

The linpack benchmark program running on the Atari was used to determine the overall speed of the FPU running floating point code. It rated the FPU at 8 KFLOPS without using parallel instruction scheduling. Using parallel instruction scheduling and accounting for I/O overhead, the FPU could theoretically run at 60 KFLOPS.

7 Conclusions

Self-timed circuits constitute an interesting design domain. Many small scale examples of self-timed circuits can readily be found; however, realistic self-timed circuits are rare. We have built, using a blend of FPGAs and custom CMOS chips, a self-timed IEEE single precision floating point processor. The FPU has been interfaced to a commercial computer system for testing and evaluation. FPGAs are well suited to experimenting with and developing novel systems like self-timed systems. The quick turn around time of the FPGA coupled with the ease with which self-timed circuits can be interchanged allow the designer to experiment with and fabricate different implementations of the same circuit in a timely manner.

References

[1] I. Sutherland, "Micropipelines," *CACM*, vol. 32, no. 6, 1989.

[2] "IEEE standard for binary floating point arithmetic," August 1985. ANSI/IEEE Std 754-1985.

[3] R. Constan, "A 16-bit cartridge port interface," *ST Log*, January 1989.

[4] C. L. Seitz, "System timing," in *Mead and Conway, Introduction to VLSI Systems*, ch. 7, Addison-Wesley, 1980.

[5] E. Brunvand, "Using FPGAs to implement self-timed systems," *Journal of VLSI Signal Processing*, vol. 6, 1993. Special issue on field programmable logic.

[6] E. Brunvand, "A cell set for self-timed design using actel FPGAs," Technical Report UUCS–91–013, University of Utah, 1991.

[7] Actel Corporation, *ACT Family Field Programmable Gate Array Databook*, March 1991.

[8] E. Brunvand, N. Michell, and K. Smith, "A comparison of self-timed design using FPGA, CMOS, and GaAs technologies," in *International Conference on Computer Design*, (Cambridge, Mass.), October 1992.

[9] J. Gu and K. F. Smith, "A structured approach for VLSI circuit design," *Computer*, November 1989.