

High-Performance FPGA-Based General Reduction Methods

Gerald R. Morris, Ling Zhuo, and Viktor K. Prasanna
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-2562
{grm, lzhuo, prasanna}@usc.edu

Abstract

FPGA-based floating-point kernels must exploit algorithmic parallelism and use deeply pipelined cores to gain a performance advantage over general-purpose processors. Inability to hide the latency of lengthy pipelines can significantly reduce the performance or impose unrealistic buffer requirements. Designs requiring reduction operations such as accumulation are particularly susceptible. In this paper we introduce two high-performance FPGA-based methods for reducing multiple sets of sequentially delivered floating-point values in optimal time without stalling the pipeline.

1 Introduction

Reductions crop up in a number of situations such as dot product and vector norm. Designers must translate such large parallel cases into a series of smaller cases and reduce the sequence of values that are subsequently produced. In some floating-point kernels, the values to be reduced are calculated sequentially. Either way, we have p sets of sequentially delivered inputs, with set i containing n_i floating-point values ($0 \leq i \leq p-1$) that must be reduced to a single value, s_i .

One of the seminal papers associated with this problem is [1]. This technique will not work for consecutive sets of floating-point values without causing a stall or a buffer overflow. Our earlier work [2] only handled the case when the input sets were a power of 2. In this paper we describe two general methods for designing high-performance FPGA-based reduction circuit from pipelined floating-point cores without introducing stalls or imposing unrealistic buffer requirements.

2 Serial Reduction Method

Conceptually, the serial method is a time-division multiplexed implementation of a full binary reduction tree. It is an extension of our earlier work [2].

This method works for arbitrary length sets, n_s , up to a design-specific maximum ($n_s \leq n$), and performs each reduction in optimal $\Theta(n_s)$ time without stalling the pipeline.

As shown in Figure 1, there is some control circuitry, two α -stage pipelined adders, and $\lg(n)$ buffer levels having three words each. The buffer read and write schedule guarantees collision-free use of the pipelines and ensures that the buffers will never overflow.

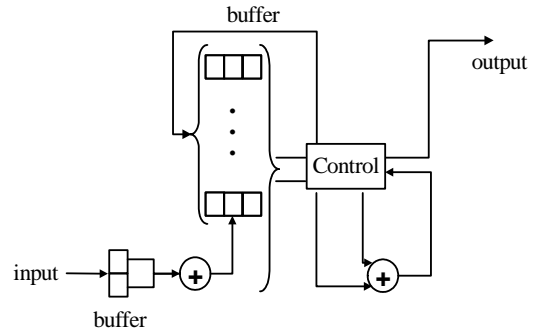


Figure 1. Serial Reduction Architecture

3 Parallel Reduction Method

The parallel method uses two α -stage adders. One adder reduces all the values for a given set. After the last value in a set arrives, there are multiple partial reductions still in the pipeline, so the other adder begins reducing the new set while the first adder finishes reducing (coalescing) the previous set. If all adders are busy, new values are buffered until an adder becomes available.

This method works for arbitrary length sets, n_s , and performs each reduction in optimal $\Theta(n_s)$ time without stalling the pipeline. Designs based on this method require $\alpha \lceil \lg(\alpha) + 1 \rceil$ buffer space and two pipelined functional units.

The reduction circuit operates in one of four modes as shown in Figure 2.

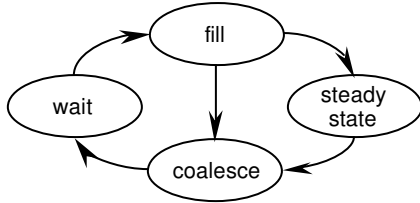


Figure 2. Modes of Operation

Wait Mode: The adder is waiting for a new set of values to arrive.

Fill Mode: The adder pipeline is filling up but has not yet produced any outputs.

Steady State Mode: The pipeline is full and a single value is in the FIFO. The pipeline output value and FIFO value are inserted into the pipeline.

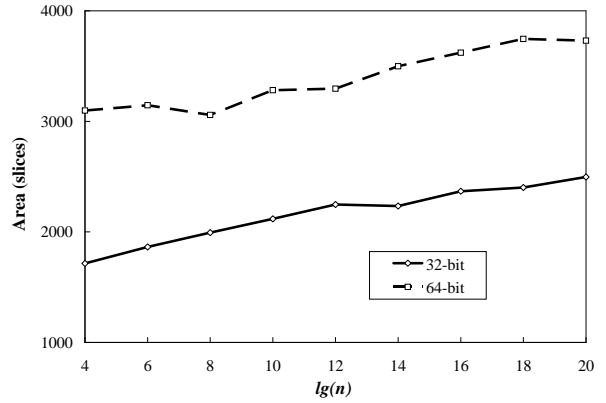
Coalesce Mode: The *coalesce* mode begins whenever the last value in a set is read from the FIFO. After coalescing the pipeline contents down to a single value, the circuit transitions back to the *wait* mode.

4 Experimental Results

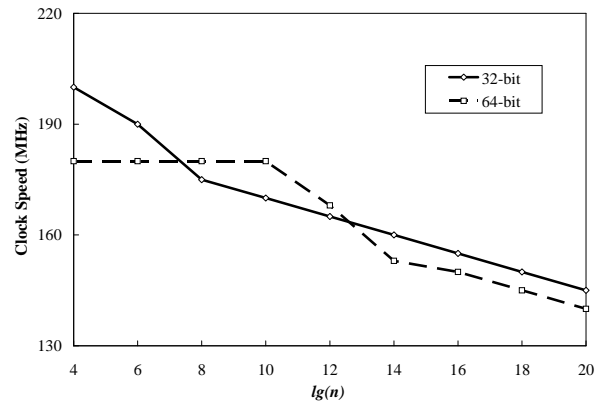
Using the Xilinx ISE 6.2i and Mentor Graphics ModelSim 5.7 development tools with a Xilinx Virtex-II Pro XC2VP7 as target device, we implement several designs based on the serial method.

In our experiments, we use IEEE-format floating-point adders for both the single precision (32-bit) and double precision (64-bit) implementations. Both of the adders are pipelined and achieve high clock speed.

We show how the area and the speed of the serial reduction method vary with n . As n increases from 2^4 to 2^{20} , the area of a 32-bit design increases by about 45%, and the area of a 64-bit design increases by about 20%. From Figure 3b we see that the degradation in speed for a 32-bit design is about 25%, and the degradation in the speed for a 64-bit design is no more than 25%. Clearly our method is scalable across a broad range of input sizes. Furthermore the actual number of slices used, even for the largest design gives our method an exceptionally small footprint, which should easily fit into most design situations.



(a) 32-bit



(b) 64-bit

Figure 3. Reduction Circuit Characteristics

5 Conclusion

We have presented two high-performance FPGA-based general methods for reducing multiple sets of sequentially delivered floating-point values.

The diminutive footprint and high clock-rate make these methods ideal candidates for a number of applications where multiple sets of sequentially delivered floating-point values must be reduced.

References

- [1] L. M. Ni and K. Hwang. Vector reduction methods for arithmetic pipelines. In *Proceedings of the 6th International Symposium on Computer Arithmetic*, June 1983.
- [2] L. Zhuo, G. R. Morris, and V. K. Prasanna. Designing scalable FPGA-based reduction circuits using pipelined floating-point cores. In *Proceedings of the 12th Reconfigurable Architectures Workshop*, Denver, CO, April 2005.