Optimizing the FPGA Memory Design for a Sobel Edge Detector

Craig Moore, Harald Devos and Dirk Stroobandt
Hardware and Embedded Systems Group
Electronics and Information Systems Department
Ghent University, Belgium
craig.moore, harald.devos, dirk.stroobandt@elis.ugent.be

***Index Terms*—Buffers, Field programmable gate arrays, Memories, Memory architecture**

***Abstract*—This paper explores different memory systems by investigating the trade-offs involved with choosing one memory system over another on an FPGA. As an example, we use a Sobel edge detector to look at the trade-offs for different memory components. We demonstrate how each type of memory affects I/O performance and area. By exploiting these trade-offs in performance and area a designer should be able to find an optimum on-chip memory system for a given application.**

I. Introduction

Hardware designers enjoy exponential gains in computation speed, compared to software designers, by taking advantage of the highly parallel nature of their architecture. They have the freedom to utilize their hardware in any way they choose, but this makes the size of their problem space much larger. Software designers only need to optimize their design on one particular fixed architecture, thus simplifying the number of considerations they need to make. Therefore, hardware designers have both great opportunities and great challenges when it comes to designing an application to run on much more flexible field programmable gate array (FPGA) architectures.

We claim that much of the design effort should be focused on the memory system for FPGAs since they can be a major bottleneck in system performance. Hardware compilers should be able to take advantage of the flexibility of FPGAs to configure their memory system efficiently, but to do this they will have to balance the various performance metrics for the different memory components available on the FPGA chip. This paper explores several memory systems to show the trade-offs of using one memory component over another. By exploiting these trade-offs a designer can find the optimum on-chip memory system for a given application.

II. Memory System Metrics

There are three important metrics when it comes to memory: bandwidth, latency, and size. In terms of I/O performance the two critical components are bandwidth and latency [1]. Using a traditional pipeline analogy, bandwidth would be the size (flow rate) of the pipe or the number of pipes used. Latency can be described as the length of the pipe(s). The longer the pipe, the longer it takes for the data to reach its destination and the slower the overall system. Obviously, the closer the source is to its destination the shorter the pipeline length. If items can be stored near their destination, then they will not need to be retrieved further upstream which reduces the overall execution time. This also adds size as an important metric in terms of I/O performance. Since the size of the memory located closest to the functional units is limited, new values will have to be stored/retrieved from a larger external memory. If there is insufficient space in the memory, then old values will have to be replaced. A well designed memory system should only replace those values that are no longer needed in order to prevent having to retrieve the same value again. Therefore, the goal should be to minimize (or eliminate) multiple calls to external memory for the same value to reduce the overall execution time.

Memory systems in FPGAs have many of the same components as memory systems for traditional Von Neumann processor architectures. The difference between the two is that FPGAs can be reconfigured and optimized for an individual application. Components can also be configured so that each element stored can be accessed in parallel rather than sequentially. Besides bandwidth, latency, and size, consideration should also be given to how the selected memory system will affect the footprint of the design. There are typically three different types of memory available: registers, block memory (RAM), and external memory. Each type has its own benefits and drawbacks, as seen in Table I.

TABLE I
Proposed taxonomy of FPGA memory trade-offs.

Type	Latency	Bandwidth	Size	Area
registers	++	++	±	– –
block memory	+	+	±	+
external memory	–	– –	++	++

± Depends on FPGA architecture selected

III. Exploration of Memory Systems

With these metrics in mind, we have evaluated four different memory designs shown in Table I. We did this exploration using the Sobel edge detector because it offers a number of opportunities for data reuse, prefetching, and parallel access.

TABLE II
TABLE OF SYNTHESIS RESULTS FOR EACH DESIGN.

Design	Frequency	Cycles[1]	Time[1,2]	Area[1,3]	Registers[1]	RAM[1,4]	Accesses[1,5]	Bandwidth[1,6]
1	142.82 MHz	810,275	5.67 ms	187	91	0 bits	808,992	142.93 B/s
2	41.31 MHz	102,403	2.48 ms	19,079	8,386	0 bits	102,400	41.29 B/s
3	123.20 MHz	104,000	0.84 ms	459	252	7,584 bits	102,400	121.30 B/s
4	134.73 MHz	103,683	0.77 ms	492	260	5,056 bits	102,400	133.06 B/s

1. Values when used with a 320×320 pixel, 8-bit encoded bitmap input image
2. Execution time $= cycles/frequency$
3. The total number of logic elements utilized by the design
4. Bits of on-chip RAM memory blocks utilized
5. Number of accesses to external memory made by the design
6. Required external memory bandwidth = $\big(memory\ accesses \cdot (bytes/access)\big)/time$

The Sobel edge detector example is also similar to a number of other applications that use windowing operations, such as a FIR filter and wavelet transforms [2]. Sobel edge detectors are used in image processing to identify and isolate areas on an image with strong intensity changes from one pixel to the next.

The four designs summarized in Table II each have a different type of memory system. The first design uses only external memory with no data reuse. It has a limited number of registers that are used to cache values read from memory. This design has the smallest footprint, fastest clock speed, largest required bandwidth, and the greatest number of required clock cycles. Its limited storage capacity means it must call external memory multiple times for the same value, which is why the design requires 8 times more clock cycles than the other designs. The second design is composed of a large number of registers so that values can be stored and reused. This design has by far the largest footprint, slowest clock speed, smallest required bandwidth, and also the fewest number of clock cycles. The third design uses only block memory where values are shifted into place. The design has a moderate footprint, clock speed, and bandwidth with an average number of required clock cycles. The fourth design is a combination of 9 shift registers and two memory blocks where values are also shifted into place. This design is much like the third design, but with a faster clock speed, fewer clock cycles, and smaller block ram utilization.

After simulating each design, we verified that it produces the same output images as the C code which we based our designs on [3] and synthesized it using Altera's Quartus II Version 8.0 targeting a Stratix EP1S25F1020C5 board. The results are shown in Table II.

IV. CONCLUSIONS AND FUTURE WORK

The fourth design which is a combination of registers and block memory worked best because it had the shortest pipeline and most efficient memory utilization. It contains a type of 'smart' buffer that shifts values into the sliding window, without having to read values in from memory multiple times. We believe this design could be used with a high-level compiler and is an improvement over the memory systems developed in [2], [4].

Our exploration of different memory systems for a Sobel edge detector has helped us to develop a memory taxonomy where trade-offs exist in the different types of memory available as shown in Table I and Table II. Many of these trade-offs are due to the location and density of the components. Registers are distributed evenly across the area of the FPGA so building a large memory block from registers means longer path lengths in order to connect them together. Memory blocks have a tighter density but longer latency and less bandwidth than registers. They also have fixed locations so functional units must be built close by in order to reduce the path length between them. If there is insufficient space around these fixed memories, functional units must be placed farther away, leading to longer path lengths and routing congestion. It is only by exploiting these trade-offs that a designer can find the optimum on-chip memory system for a given application.

For a Sobel edge detector, we found that combining registers with block memory works best. For any other application, a similar exploration may result in a different memory system which is best suited for it. Further study into more applications should give us enough information to construct a basic set of memory templates that can be used with a high-level compiler, which is the goal of our future work.

V. ACKNOWLEDGMENTS

This research is supported by the I.W.T. grant 060068. Ghent University is a member of the HiPEAC Network of Excellence. The authors would like to thank Karel Bruneel and Wim Meeus for their valuable suggestions.

REFERENCES

[1] J. Hennessy and D. Patterson, *Computer Architecture - A Quantitative Approach*, 3rd ed., D. Penrose, Ed. Morgan Kaufmann, 2003.

[2] Z. Guo, B. Buyukkurt, and W. Najjar, "Input data reuse in compiling window operations onto reconfigurable hardware," in *LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, Compilers, and Tools for Embedded Systems*. ACM Press, July 2004, pp. 249–256.

[3] B. Green, "Edge detection tutorial," http://www.pages.drexel.edu/~weg22/edge.html, 2002, last Accessed 2009.02.24.

[4] Y. Dong, Y. Dou, and J. Zhou, "Optimized generation of memory structure in compiling window operations onto reconfigurable hardware," in *International Workshop on Applied Reconfigurable Computing (ARC)*, ser. Lecture Notes in Computer Science, vol. 4419, 2007, pp. 110–121.