

Feature Article

FPGA Lessons Learned

Introduction

The Alabama Supercomputer Authority (ASA) fills a number of roles in the state of Alabama. These roles include providing high performance computing facilities for educational use, providing network connectivity, hosting computer systems for other institutions, and being a technology leader in high performance computing. In this last role as a technology leader, ASA has twice had the first supercomputer of a new model installed in the country, and has provided a number of systems with new technological advances, such as hypercube architectures, vector processors, and rapid array interconnects. In late 2006, ASA continued this trend by installing FPGA chips into the Cray XD1 supercomputer at the Alabama Supercomputer Center (ASC).

This article is a discussion of FPGA technology, written for people that are new to this chip architecture. The following sections will discuss what FPGA chips are, why they can make software run faster, when they can't, and how much work is involved in making programs run on FPGA chips. We won't attempt to go into all of the technical details, but rather intend to give the flavor of what is involved in an FPGA development project. However, individuals considering embarking on their first FPGA development project should be well served by considering the information presented here before they begin.

What is an FPGA?

FPGA stands for Field Programmable Gate Array. An FPGA is a computer chip, but it is significantly different from most other chips. Even very complex chips, like the processor in your desktop computer, have a set of functions that are burned into the chip and don't change. When you run different programs, different data is run through the processor, and it executes instructions in a different order, but the chip itself doesn't change. An FPGA chip is designed with programmable logic blocks, so that in essence the actual circuits on the chip can change. Every program that uses an FPGA chip can flash a new circuit design onto the chip,

tailored specifically for the needs of that program. This scheme of being able to set up a special set of circuits to execute each program is called "reconfigurable computing".

FPGA chips are used in a number of consumer electronic devices, such as cell phones. This is done for cost saving reasons. If chips had to be custom designed for each cell phone, then manufacturers would need to manufacture a hundred chip models in order to offer a hundred cell phone models. It is much less expensive to have one manufacturing plant make FPGA chips and flash a different set of circuits onto the chip for each model of cell phone. Using reconfigurable chips also makes it possible to update the design to fix problems.

FPGA chips can also be used to make computer programs run much faster. Consider a case in which a computer program spends a very large amount of time calling one specific function with different pieces of data. A circuit could be designed to perform that function. When the program starts a hundred copies of that circuit could be flashed onto the FPGA chip. When the program runs, it will use the FPGA to perform that critical function, and run as fast as it would run on a hundred computers at once. In some cases, the performance improvement from utilizing FPGAs can be a thousand fold or more.

What applications can run on FPGA chips?

The example of a program running a hundred times faster in the previous paragraph is very encouraging. Unfortunately, not all programs can be adapted to run on FPGAs. The limits that determine what software can or can't run on FPGAs shift as new FPGA chip designs are released, but there are always limits. These limits are based on three critical issues; chip real estate, parallelizability, and programming labor cost.

Chip designers refer to a computer chip as having a certain amount of "real estate". There is a limited amount of space on a chip, which limits the number of transistors, gates and circuits that can be put on the chip. If the functions the FPGA chip is to perform would require too much space on the chip, then it won't be possible to use the FPGA. Remember that great performance increases primarily come from putting many copies of the circuit onto the FPGA chip, so it is necessary to have enough chip real estate to fit perhaps 100 or more copies of the circuit. To most people, it isn't obvious from looking at a program source code

how big the circuit to do that task would be. At the time this article was written, the following rule of thumb was recommended;

If your program spends 90% of the time executing 200 lines of code or less, it might be a good candidate for FPGAs.

This article already touched on the idea of running many copies of the circuit in parallel. This is because the FPGA chips actually run significantly slower than the typical computer processor. There is also an additional overhead of flashing the FPGA circuits at the beginning of the job. There may be some bottlenecks moving data onto and off of the FPGA chip as well. Of course, the FPGA chip itself costs money. Lastly, the biggest cost involved in getting a piece of software to use FPGAs is the labor required to reprogram the software. The metric of wanting to fit 100 copies or more of a function on the FPGA chip is a rough metric for estimating whether the performance gain will be enough to offset all of these other issues.

Programming FPGAs

As FPGAs aren't processors, you cannot compile existing code to run on FPGAs or link in a FPGA library. The process instead consists of designing circuits, and testing the behavior of those circuits. There are some utilities for FPGA programming that give C language like interface for convenience, but an intimate knowledge of the chip architecture and circuit design is still required. At the present time, no Fortran options are available. As such, most people that do this type of work have a strong background in electrical engineering or computer engineering. Even with that background, this isn't a trivial task. People who wish to become FPGA programmers should plan on taking a class in reconfigurable computing, then devoting a couple years of work to getting their code to run on FPGA chips.

One of the primary means for programming FPGAs is VHDL (VHSIC hardware description language). VHDL is a language that looks similar to ADA in syntax. However, it does not define how a program works. It defines the behavior of the circuits on a computer chip. Once VHDL has been generated and tested, a program can be used to do the "place and route" to figure out the exact physical placement of those circuits on the chip. Place and route is generally a fairly automated process, although it

may take hours to complete. There are also other hardware description languages, such as Verilog, which is nearly as popular as VHDL.

There are simulator programs that can use the VHDL to run a simulation of how the chip will behave. Unfortunately, the simulators aren't 100% reliable. The fact that the chip seems to run in the simulator doesn't guarantee that it will run on the physical chip. In some cases, these problems can be alleviated by using only 80% of the real estate on the FPGA chip to leave a margin for error. One way to improve chip utilization is to use an analysis system (analogous to testing every junction on the chip with an oscilloscope) to identify and adjust for problems. Most FPGA chips also have some pre-made circuits that can be utilized to do common tasks.

The step past VHDL in terms of usability are some more advanced programming tools that are available. Examples are Impulse C, Starbridge, and the Mitrion Software Development Kit. These tools allow the FPGA developer to work in a more C like environment, or even from graphic interfaces. However, they are not complete C compilers. Thus you can't just recompile C programs, or link against C libraries. These tools can be very useful for FPGA development, but the person using them still needs to have a solid understanding of the FPGA architecture and circuit design.

There are various models of FPGA chip. VHDL and the other programming utilities go a long way towards having a portable code. However, there can still be issues associated with moving code to different chip models. If the code has been written to utilize features of a specific model FPGA chip, it may give better performance, but be harder to port to other chips. Because the FPGA code is so intimately tied to the hardware, there tend to be more FPGA portability issues (even between chips from the same company) than are seen with conventional computer programs.

Conclusions

FPGAs can give incredible improvements in program performance. However, they aren't appropriate for all applications. Over time enhancements in chip capability are likely to make FPGAs viable for a wider range of applications. This article isn't intended to discourage the use of FPGAs, but rather to give would be, first time, FPGA developers a

realistic view of what is involved in the FPGA development process. FPGAs can be very powerful tools, in the hands of engineers that have the skill to use them effectively.

References

Wikipedia article on VHDL <http://en.wikipedia.org/wiki/VHDL>

Impulse Accelerated Technologies <http://www.impulsec.com/>

Mitrion <http://www.mitrion.com>

Starbridge <http://www.starbridgesystems.com/>

Presentations given at a workshop on FPGA programming are at <http://www.asc.edu/seminars/index.shtml>