2002

NASA FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

EVALUATION OF ADVANCED COMPUTING TECHNIQUES AND TECHNOLOGIES: RECONFIGURABLE COMPUTING

Prepared By:	B. Earl Wells
Academic Rank:	Associate Professor
Institution and Department:	University of Alabama in Huntsville Electrical and Computer Engineering
NASA/MSFC Directorate:	Engineering
MSFC Colleague:	David Gwaltney

Background

It is common practice to divide a digital system into software and hardware components. The greatest functionality and performance occurs when functions are placed in dedicated system hardware that is highly parallel and is optimized to perform the intended operations. More often than not it is far too expensive in time and money to create such dedicated hardware so the available hardware resources are used to implement a conventional processor which executes programmable instructions in a highly sequential manner. While this is very flexible such conventional processors are inefficient -- a large percentage of hardware resources (up to 99% of the logic gates that make up its hardware) are idle most of the time. Commercially-off-the-Shelf digital and analog hardware now exists that can change the "circuit" that is being implemented long after the device(s) are fabricated. This means that high speed hardware can now share much of the flexibility normally reserved for software-only systems.

Not all applications need the added speed and other possible features of reconfigurable systems such as fault tolerance and power management techniques, but some applications that could greatly benefit from this technology include, Genetic Algorithms, Real time processing, Data encryption/decryption ,RSA cryptography, Data Compression, Image and Video processing, String matching, Heat and Laplace equations, Newton's mechanics, Continuous System Simulation, Evolutionary Computing, Fault Tolerant Computing, Binary 2D convolution, Boltzmann machine, 3D graphic Accelerators, Discrete Cosine Transforms, Irregular Mathematical operations. For this reason, investigation into emerging techniques and technologies in reconfigurable systems is an active area of research which directly supports the goals and objectives of NASA.

Project Goals

The focus of this project was to survey the technology of reconfigurable computing determine its level of maturity and suitability for NASA applications. To better understand and assess the effectiveness of the reconfigurable design paradigm that is utilized within the HAL-15 reconfigurable computer system. This system was made available to NASA MSFC for this purpose, from Star Bridge Systems, Inc. To implement on at least one application that would benefit from the performance levels that are possible with reconfigurable hardware. It was originally proposed that experiments in fault tolerance and dynamically reconfigurability would be perform but time constraints mandated that these be pursued as future research.

Targeted Reconfigurable Computing Platform

The targeted reconfigurable computer system that was used in this research was one that was developed by Star Bridge Systems, Inc. to utilizes devices which are commonly called Field Programmable Gate Arrays, FPGAs. FPGAs were introduced to the hardware design community in the mid 1980s as a means to incorporate so-called random logic into a minimal number of integrated circuits. Since that time these chips have grown drastically in size and function where they have had a major effect in the areas of rapid prototyping of low volume application specific designs, digital logic emulation, and have served greatly to improve the time-to-market for new product designs. Star Bridge Systems and other companies are trying to capitalize not as a rapid prototyping device but as advanced computing devices. It should be noted that current FPGA technology is not optimized for this use.

Targeted Reconfigurable Platform

The reconfigurable system that served in this evaluation was the HAL-15 Hypercomputer system from Star Bridge Systems, Inc. It has a peak performance estimate of 40 billion fixed point 16-bit multiply accumulates per second operations, and 15 billion 32-bit floating point operations per second. It utilizes a total of 10 Xilinx 4062 800 picosecond FPGAs where each 4062 FPGA has 2304 Combinational Logic Blocks which translates into somewhere between 62,000-130,000 logic gate equivalents. It is PC hosted on a PCI 32 bit 33 Mhz daughter board. Systems can contain multiple daughter boards. The FPGAs are interconnected with the PC and each other using a proprietary bus structure. There is a total of 288 Mbytes of SDRAM and a 200 Mhz system clock.

The development environment is the VIVA ™ Graphical User Interface. This environment supports a structural object oriented design philosophy that allows for the incorporation of certain high level behavioral attributes some of which are normally found in textural-based hardware description languages such as VHDL, Verilog and Handle C ™. These include the concept of polymorphism where objects are created to support data types of multiple precession, operator overloading, where object with the same name can be utilized to handle different operations, and recursion where objects are defined in terms of themselves (note method of representation for hardware is novel). The major synchronization technique that is used closely follows the data flow paradigm where objects that are to be processed sequentially are designed in a pipelined manner where they must wait for the objects the precede it in the pipe to complete before it can begin operations. To accomplish this objects have special signals that performs the necessary handshaking protocol. The tool comes with a large library of high-end objects such as floating-point multipliers, dividers, adders, transcendental functions, etc. In many ways this environment falls somewhere between hardware description languages and schematic capture packages in terms of features and function.

Application 1: Implementation of a Genetic Algorithm

The first application that was implemented was a Genetic Algorithm that minimized a simple numeric objective function. Genetic Algorithms represent a set of search techniques, which are modeled after the evolutionary view of biological systems. They incorporate the ideas that an algorithm can be represented as a pattern that can be matted with other patters via the cross-over operations. A fitness function can then be applied to the resulting children to determine which of the children and parents continue over to the next generation. There is also a small probability of that the pattern will mutate thereby changing one or more of its elements.

Figure 1 shows the high-level Viva diagram for the genetic algorithm. The design is hierarchical with this being the top level -- there are a total of 44 user created objects in this design. Here we see the general structure of the design with two main storage elements. The first is the next generation storage where the children elements are stored and the associate memory which performs the combined operations of next generation storage and applying the fitness function. In this case the population size was set at a value of 16 members. Upon receiving the Go pulse the Generate_Init_Population module randomly generates X, Y data and applies the objective function for all members of the population. It then stores the results in the next generation storage. When it is complete it signals the for loop object to begin operations. This object pulses

the next generation fitness controller, which moves the children elements into the population using the objective function as a key. The associative memory only maintains a copy of the best 16 elements presented to it any elements that are worst than its worst element do no get stored and therefore do not make it to the next generation. After this is done the cross over mutate module gets pulsed. This will create 16 new children by randomly mating two elements from the population in the associative memory and then storing them in the next generation storage. When this is complete the outer level controlling for loop is signaled that the iteration is complete. When all iterations are complete the associative memory is signaled to output the best member of the population.

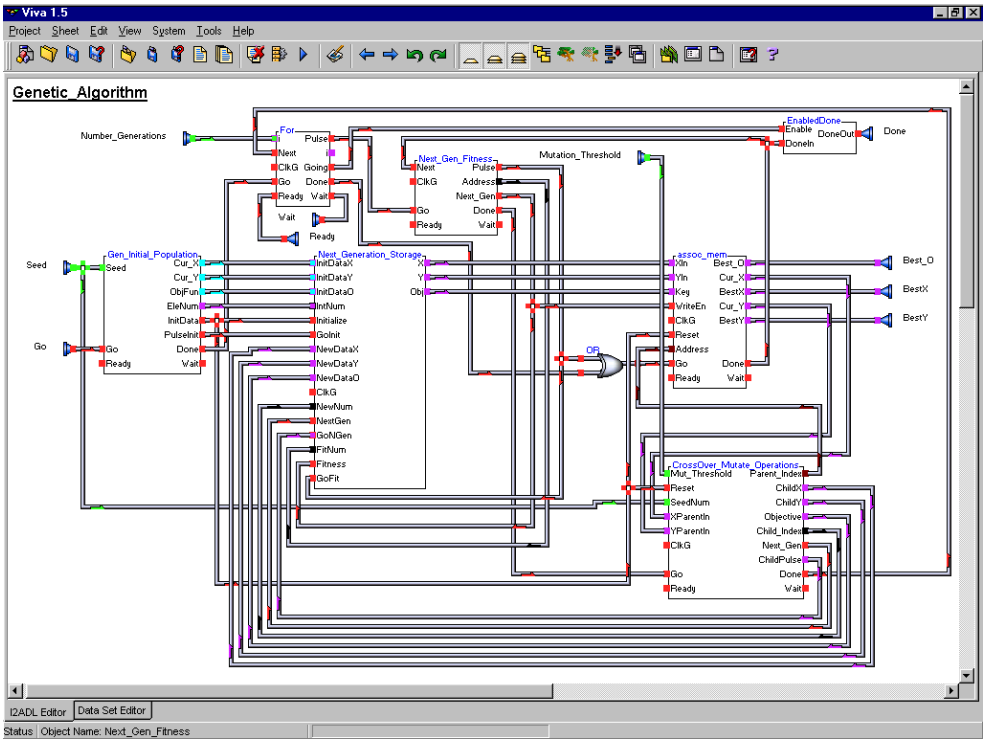

Figure 1: High-Level Viva Representation of the Genetic Algorithm

After the design methodology was fully explored the genetic algorithm was implemented in a couple of weeks. The algorithm occupied less than one of the 10 FPGAs and produced a result, which was an order of magnitude faster than an equivalent software implementation. The implementation utilized only integer arithmetic but an equivalent fixed or floating-point representation could be made by modifying the objective function and a small number of objects.

Application 2: Continuous System Simulation

The second application that was applied was a continuous simulation of a small system of differential equations that form the classical initial value problem. This utilized floating-point arithmetic. A simple Euler integrator module was created that performed the multiply accumulate operation. Figure 2 shows how these were linked to implement the sin function using a simple second order ODE.

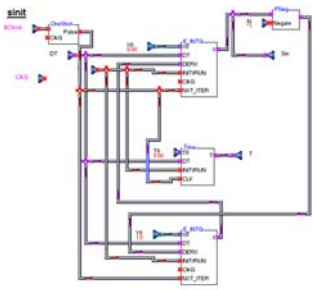

Figure 2: Continuous Simulation of a second order ODE that generates the sin(t)

Continuous simulation seems especially suitable for reconfigurable hardware implementation that has the flexibility to support high-speed operations for arbitrary structured dynamic systems. It directly supports the low level of granularity that is present in most continuous simulations and allows for efficient interpretation communication between object modules --something that is too costly with conventional parallel processing technology. There are still major space time trade-offs need to be considered. How to model nonlinear constructs such as large lookup tables and transcendental are also a major consideration.

Conclusion

This project successfully demonstrated how two types of applications can be implemented on the HAL-15 reconfigurable computer using the Viva design entry language. Viva is competing with schematic capture traditional HDL and other high level languages that are being modified to create hardware. There is a high learning curve associated with these languages. Viva is designed to minimize this learning curve. This investigator is probably biased since he has already learned these tools -- his evaluation is that Viva is closer to schematic capture than HDLs or high-level language derivatives. He also feels that Viva will not replace existing digital design methodologies since they offer a wider amount of control and functionality which is sometime needed. The use of Viva may fill an important need where algorithm developers who are not digital designers can evaluate the effectiveness of their designs or accelerate their calculations without the high learning curve associated with other techniques. This is analogous to the use of specialized graphical object oriented software in domains ranging from discrete event simulation, system architecture description languages to probabilistic analysis. This is just initial research -- more applications will have to be explored before definitive conclusions are reached. Such future research should include more advance Genetic Algorithms, Continuous Simulation, Neural Networks. Also investigations into dynamic reconfiguration techniques, fault tolerance, evolutionary computing and intelligent power management should be explored.

References

[1] Uher, Bill,. (2001), “Computing Faster Than Engineers Can Think” NewsRelease NASA Langley Research Center, Hamption, Virginia, Release No. 01-.

[2] Michell, M. (1998), *An Introduction to Genetic*, MIT Press, ISBN: 0262631857.

[3] Gear, C. W., (1971) *Numerical Initial-Value Problems in Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, N.J.