ACE*card*™: A High-Performance Architecture for Run-Time Reconfiguration

Don Davis
TSI TelSys, Inc.
7100 Columbia Gateway Drive
Columbia, MD 21046
ddavis@tsi-telsys.com

Jonathan Harris
TSI TelSys, Inc.
7100 Columbia Gateway Drive
Columbia, MD 21046
jharris@tsi-telsys.com

Abstract

Recent FPGA architectures have shown an increased emphasis on run-time reconfiguration, or the ability to rapidly change the functionality of the FPGA to sequentially accommodate large processing tasks. In addition, partial reconfiguration allows for the reconfiguration of a portion of the FPGA while the remainder is running. These two features enable the use of reconfigurable computing in high-performance multi-threaded multi-user environments. However, current board designs are not optimized to provide the processing support required to maintain this run-time environment which includes management of the reconfigurable resources, interface to the host processor and data movement.

In this paper, we will describe the architecture, design and applicability of the ACEcard, a high performance reconfigurable co-processor. The ACEcard contains reconfigurable resources as well as an embedded processor to manage the runtime reconfiguration of those resources. We will provide details of the architecture of the card as well as a description of the current and future Java-based run-time environment.

1. Introduction

Much of the current work on FPGA architectures for reconfigurable computing focuses on run-time reconfiguration [1, 2, 3]. Run-time reconfiguration is the ability to rapidly change the functionality of an FPGA during the execution of the processing task. This allows an arbitrarily large algorithm that can be split up into a sequential series of sub-algorithms (that can fit into the available reconfigurable resources) to be processed by loading the first sub-algorithm, processing the data, then loading the second and so on. The performance achieved using this approach is limited by the FPGA clock rate, the FPGA reconfiguration rate and the ability to move data into and out of the part and store intermediate results.

Partial reconfiguration is the ability to change the functions of a portion of the FPGA while the remainder of the chip continues processing. When partial reconfigurability is combined with rapid run-time reconfiguration, a reconfigurable co-processor can effectively be inserted into a high-performance multi-threaded multi-user environment. In such an environment, there will be a number of processing tasks occurring simultaneously, each with a variety of resource needs, including traditional processor resources, memory resources and reconfigurable processor resources. The management, control and coordination of these resources is the function of the run-time environment.

2. The Run-Time Environment

The run-time environment provides support for multiple users and multiple threads spread across a heterogeneous environment of traditional processors and reconfigurable processors. Most reconfigurable computing devices do not include a dynamic run-time environment and their FPGAs support only functional reconfiguration (that is, they cannot be rapidly or partially reconfigured). This means that all issues such as placement and routing and data movement setup must be handled *a priori*. This approach is adequate for a single, static embedded task but does not scale well to server and multi-user applications. A dynamic run-time environment does not require the generation of configuration files off line before the system is run. Instead, partial configuration files are generated for relocatable "hardware objects." During run-time these partial configuration files are dynamically placed and routed together.

In the ACEcard co-processing environment, the host processor makes a call (via an API) to the reconfigurable co-processor. This high-level call includes, among other things, the identity of the function to perform, the location of the input data and the target location of the output data. At this point, the host processor is free to perform other tasks and make other calls to the

reconfigurable co-processor. At some later time, the host processor can retrieve the results of the computation.

The run-time environment executing on the reconfigurable co-processor interprets these high level calls and sets in motion the local processes necessary to perform the task. First, the host processor call (method invocation) is sent to the runtime environment. This method invocation is associated with a specific hardware object. A hardware object is low level functional elements that provide discrete, self-contained functionality. They may be adders, multipliers and comparators or higher order functions like filters and transforms. Hardware objects are pre-routed, relationally placed and designed to conform to a standard interface specification. The goal for designing these objects was to minimize the amount of real-time routing performed by the run-time environment. This provides two advantages. First, each object is optimally prerouted and relationally placed for the task it performs so fine grained placement and routing is not necessary. Second by designing to a standard inter-object interface specification, objects can be interconnected with relative ease.

Once a hardware object is called, the runtime environment takes the relocatable hardware object and places it in the reconfigurable resources. This consists of finding a space in the reconfigurable resources large enough to hold the hardware object. If there is no space available, the run-time environment must decide whether to wait for another object to finish or to swap out a currently running process, saving its state and restoring it when resources become available. This function is analogous to virtual memory management in a traditional microprocessor. The hardware object is then routed (inter-object routing) if necessary. This placement and inter-object routing is handled by the run-time environment. After this is complete, the data is sent to the objects, processed and sent back to the host process. The data movement functions are also handled by the run-time environment.

3. The ACE*card*

The maintenance of the run-time environment is a complex task. It must perform a number of intensive functions in real-time. If the placement, routing or data movement functions are not performed quickly, any performance gains realized by the reconfigurable resources could be lost by the overhead of the run-time environment. To address these concerns, TSI TelSys designed and built a high-performance PCI-based co-processor card which includes run-time reconfigurable FPGAs and an embedded processor - the ACEcard.

The choice of FPGA architecture was a fundamental driver of the card design. We chose Xilinx XC6264 Reconfigurable Processing Unit (RPU) parts for a number of reasons. The XC6264 parts are dynamically run-time reconfigurable and partially reconfigurable. We saw this as critical to the proper implementation of a reconfigurable co-processor architecture. The reconfiguration time is much faster than other commercially available parts from Xilinx, Altera and others. Fast reconfiguration is an important feature in a dynamic run-time environment. If reconfiguration times are large compared to reconfigurable processing times, no performance gains will be realized.

In addition, the programming of the chip is a public specification allowing us to develop our own real-time place and route routines in a very straightforward manner. This public specification also allows access to the current operating state of the entire chip. This is a necessary feature to implement hardware object swapping; that is, the ability to stop the processing of a current object, save its state and replace it with another object that is a higher priority. Other advantageous features of the XC6264 RPUs include: a 32 bit FastMAP™ interface for direct microprocessor access to all chip resources and a large array of available gates (up to 100,000) including up to 16K registers. The ACEcard includes two XC6264 chips with 41 configurable interconnects between them. This yields a total of 200,000 gates with up to 32K registers.

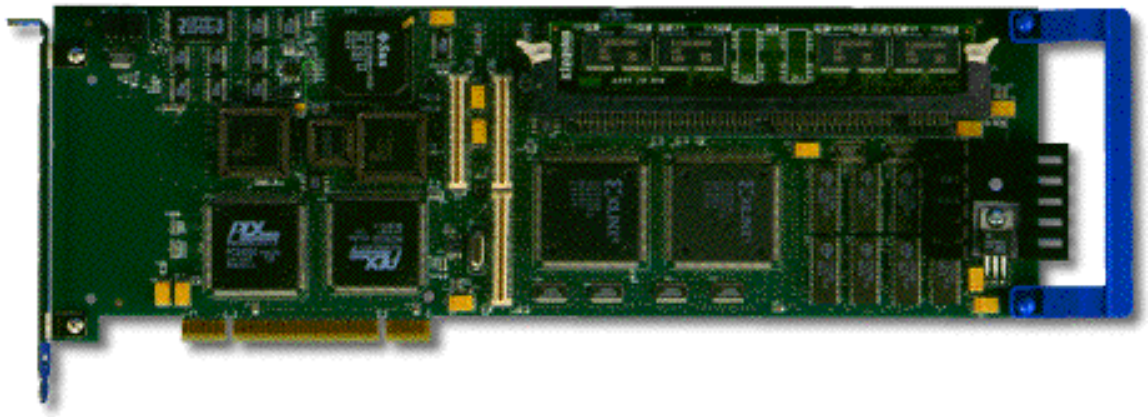

Figure 1: ACEcard

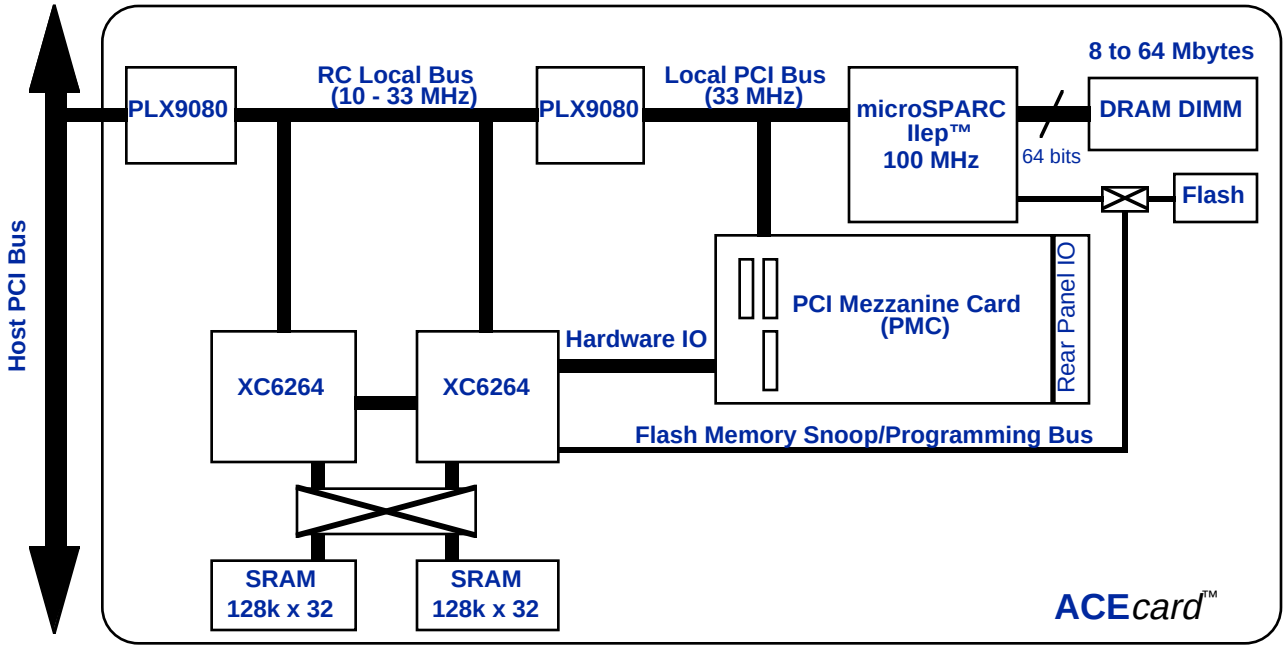

Figure 2: ACE*card* Architecture

Another important design factor was support for the run-time environment. This includes both a computing resource on which to execute the environment as well as a means to provide high speed communication and data movement between the host and co-processors. To handle these tasks, we included a high performance embedded processor on the board. We chose a 100 MHz microSPARC IIep processor. It is based on the SPARC architecture specification version 8 and includes a floating point unit, memory management unit, 16 Kbyte instruction cache, 8 Kbyte data cache, DRAM controller and a 33 MHz PCI interface. We also included up to 64 MBytes of DRAM and 1 Mbyte of flash memory. Some performance specifications are given in the table below.

Benchmark	Performance
SPECint92	65.3
SPECfp92	55.6
Dhrystone MIPS	117.7
Memory Read	152 MBytes/sec
Memory Write	177 MBytes/sec
DVMA (memory to PCI)	45 MBytes/sec
DVMA (PCI to memory)	70 MBytes/sec

Table 1: microSPARC IIep Performance

Coupled with this embedded processor are high performance PCI bridge chips which provide DMA channels for moving data at up to the full PCI bus bandwidth (33 MHz by 32 bits - 1.2 Gbps). The architecture of the ACEcard is shown in Figure 2. All data paths and memory accesses (except for the flash) on the ACE card are 32 bits wide and can run at 33 MHz. This is equal to the bandwidth of the host PCI bus and allows us full bandwidth performance across the entire system.

In addition to the RPUs and the embedded processor, we included 1 Mbyte of synchronous SRAM attached directly to the RPUs. This memory operates at 33 MHz with zero wait states. It is arranged as two banks of memory 128k by 32 bits attached to each of the RPUs. There is a crossbar switch between the memory and the RPUs to allow an RPU to access both banks of memory.

The last element on the ACEcard is an interface to a PCI Mezzanine Card (PMC). This interface is CMC/PMC compliant (IEEE P1386.1) and is a standard form-factor that supports the PCI bus electrical interface. It is used on the ACEcard to provide interface flexibility. PMC cards are available which provide network and peripheral interfaces (Ethernet, ATM, FDDI, SCSI, others) and specialized interfaces (TTL, RS-422, serial ECL). As part of this interface there is a hardware

interface bus for direct connection to the RPUs. This hardware interface bus includes 42 bits of reconfigurable input/output signals as well as a serial clock and data interface.

The combination of dynamic run-time/partially reconfigurable RPUs, a high performance embedded processor and supporting memory coupled with interface flexibility and an architecture optimized for system level performance provides a powerful platform for reconfigurable co-processing applications.

4. ACE*runtime*™ - A Java-based Runtime Environment

Current work is focused on implementing a robust run-time environment for reconfigurable co-processors. Two system platforms have been targeted. The first is a Sun UltraAX motherboard that is based on a 250 MHz UltraSPARC processor running the Solaris 2.x Operating System. The second system platform is a Windows NT 4.0 platform with a 266 MHz Pentium Pro processor. Drivers have been written for both platforms while the run-time environment was developed in the Java programming language. This has allowed us to easily support both system platforms due to the high degree of portability inherent in Java code.

ACEruntime, a Java-based run-time environment abstracts the most basic construct in the RPU, a cell, as an object and the functionality and routing of that cell are manipulated via method calls. For the instantiation of hardware objects, the run-time environment fetches the appropriate relocatable hardware object and modifies the file to place it at a physical location in the part. Since the relocatable hardware object is relationally place and routed, it must be assigned an actual location before instantiation. After the hardware object is instantiated, any additional routing necessary to connect it to other objects is performed by the run-time environment. If necessary, currently executing hardware objects may first be stopped while their state is copied to temporary storage in order to make room for the new object. The appropriate data is then sent to the object via the high-speed DMA channels on the ACEcard. These DMAs are set up, controlled and monitored by the run-time environment. The resulting data is sent back to the host via a similar DMA process.

For an application level programmer, ACEruntime has defined a Java class library that contains some basic reconfigurable processing elements. A programmer can write Java code using objects from this class library. No special consideration need be given to the fact that the class library is referencing reconfigurable hardware. Method calls are identical to standard software method calls. When the code is run, all aspects of handling the reconfigurable resources are performed by the run-time environment with no intervention from the user necessary.

5. The Future

TSI TelSys, Inc. will make many improvements to the basic architecture described here based on the evolution of the underlying technologies. FPGA chip architectures will evolve and provide more gates with faster reconfiguration. In addition, we expect to see dedicated support for run-time environments appearing on the FPGA chips as well. At first this may be in the form of hybrid chips that contain both a traditional processor and reconfigurable resources. There are a number of research efforts in this area already. [4] Support for atomic hardware objects could also be built into the FPGA providing a framework in which to plug-in the objects. Another optimization is to provide a closer coupling between the reconfigurable processor and the host processor, perhaps even on the same chip, and include the run-time environment for reconfigurable in the host operating system.

6. Conclusion

The ACEcard combined with ACEruntime provide a high-performance run-time reconfigurable architecture and Java-based run-time environment for managing the reconfigurable resources. This ACEcard architecture was designed from the ground up to provide the resources necessary to effectively implement a run-time environment for reconfigurable computing. It features a powerful embedded processor, fast DMA channels for data movement and advanced dynamic run-time reconfiguration architected to provide maximum system-level performance.

References

[1] W. Luk and N. Shirazi. Modelling and Optimising Run-Time Reconfigurable Systems. *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 167-176, April 1996.

[2] H. Schmit. Incremental Reconfiguration for Pipelined Applications. *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pp 47-55, April 1997.

[3] J. Burns, A. Donlin, J. Hogg, S. Singh, M. de Wit. A Dynamic Reconfiguration Run-Time System. *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 66-75, April 1997.

[4] J. R. Hauser and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 12-28, April 1997.