Chapter 1. General Introduction

In the mid 1980s a new technology for implementing digital logic was introduced, the field-programmable gate array (FPGA). These devices could either be viewed as small, slow gate arrays (MPGAs) or large, expensive programmable logic devices (PLDs). FPGAs were capable of implementing significantly more logic than PLDs, especially because they could implement multi-level logic, while most PLDs were optimized for two-level logic. While they did not have the capacity of MPGAs, they also did not have to be custom fabricated, greatly lowering the costs for low-volume parts, and avoiding long fabrication delays. While many of the FPGAs were configured by static RAM cells in the array (SRAM), this was generally viewed as a liability by potential customers who worried over the chip's volatility. Antifuse-based FPGAs also were developed, and for many applications were much more attractive, both because they tended to be smaller and faster due to less programming overhead, and also because there was no volatility to the configuration.

In the late 1980's and early 1990's there was a growing realization that the volatility of SRAM-based FPGAs was not a liability, but was in fact the key to many new types of applications. Since the programming of such an FPGA could be changed by a completely electrical process, much as a standard processor can be configured to run many programs, SRAM-based FPGAs have become the workhorse of many new reprogrammable applications. Some uses of reprogrammability are simple extensions of the standard logic implementation tasks for which the FPGAs were originally designed. An FPGA plus several different configurations stored in ROM could be used for multi-mode hardware, with the functionality on the chip changed in reaction to the current demands. Also, boards constructed purely from FPGAs, microcontrollers, and other reprogrammable parts could be truly generic hardware, allowing a single board to be reprogrammed to serve many different applications.

Some of the most exciting new uses of FPGAs move beyond the implementation of digital logic, and instead harness large numbers of FPGAs as a general-purpose computation medium. The circuit mapped onto the FPGAs need not be standard hardware equations, but can even be operations from algorithms and general computations. While these FPGA-based custom-computing machines may not challenge the performance of microprocessors for many applications, for computations of the right form an FPGA-based machine can offer extremely high performance, surpassing any other programmable solution. While a custom hardware implementation will be able to beat the power of any generic programmable system, and thus there must always be a faster solution than a multi-FPGA system, the fact is that few applications will ever merit the expense of creating application-specific solutions. An FPGA-based computing machine, which can be reprogrammed like a standard workstation, offers the highest realizable performance for many different applications. In a sense it is a hardware Supercomputer, surpassing traditional machine architectures for certain applications. This potential has been realized by many different research machines. The Splash system [Gokhale90] has provided performance on genetic string matching that is almost 200 times greater than all other Supercomputer implementations. The DECPeRLe-1 system [Vuillemin95] has demonstrated world-record performance for many other applications, including RSA cryptography.

One of the applications of multi-FPGA systems with the greatest potential is logic emulation. The designers of a custom chip need to verify that the circuit they have designed actually behaves as desired. Software simulation and prototyping have been the traditional solution to this problem. However, as chip designs become more complex, software simulation is only able to test an ever decreasing portion of the chip's functionality, and it is quite expensive in time and money to debug by repeated prototype fabrications. The solution is logic emulation, the mapping of the circuit under test onto a multi-FPGA system. Since the logic is implemented in the FPGAs in the system, the emulation can run at near real-time, yielding test cycles several orders of magnitude faster than software simulation, yet with none of the time delays and inflexibility of prototype fabrications. These benefits have led many of the advanced microprocessor manufacturers to include logic emulation in their validation methodologies.

While multi-FPGA systems have great potential to provide high-performance solutions to many applications, there are several problems that hold back current systems from achieving their full promise. Their hardware structures tend to have much too few I/O connections for their logic capacity to be adequately used. This leads to very low logic utilization, with only 10% to 20% of the available capacity usable. While some of these problems may be unavoidable, since we are using FPGAs targeted to single-chip applications to build multi-chip systems, the routing topologies of multi-FPGA systems need to be improved to compensate for this. Also, while there are logic emulation systems that meet the needs of single-chip ASIC prototyping, the domain of system-level or board-level prototyping has largely been ignored. These emulation opportunities are quite different from single chip systems, since multi-chip designs have a heavy reliance on premade components, and have resource demands that vary much more widely than that of a single ASIC. Not only does board-level prototyping demand a flexible and extensible emulation hardware system, one that allows the inclusion of arbitrary premade components, but it also brings into sharper focus the need to support external interfaces. Even in single-chip systems, one of the major benefits of an emulation is that it may be capable of being put into its target environment, thus encountering the true input-output demands of the final implementation. However, an emulation of a circuit will almost always run slower than the final implementation, meaning that it will no longer meet the timing requirements of the external signaling protocols. Current emulation systems have no answer to this problem, forcing the user to develop ad hoc solutions. For board-level prototyping, where a system will be required to meet several signaling protocols at once, this problem becomes an even greater concern. Thus, a general-purpose solution to the logic emulator interface problem is critical to the success of board-level prototyping, and can improve emulations of individual ASICs as well.

The problems with current multi-FPGA systems are not solely with the hardware. Automatic mapping software is an important part of any multi-FPGA system that hopes to achieve widespread utility, and is an absolute requirement for any successful logic emulation system. While there are niches for multi-FPGA systems with hand-mapped solutions, very few people are capable and willing to expend the effort to hand-map to a multi-FPGA system, and without software these systems will only realize a small portion of their promise. Unfortunately, today's mapping software is inadequate for most potential applications. They deliver quite poor mappings, and can take huge amounts of time to complete. Creating a mapping with current tools can take almost an entire day to finish. Obviously, this is a tremendous disadvantage, since any time savings that could be achieved by running an algorithm on a multi-FPGA system is swallowed up by the time to create the mapping in the first place. Thus, high-quality, high-performance mapping tools are a necessity for realizing the full potential of multi-FPGA systems.

Part of the reason why these deficiencies exist in current multi-FPGA systems is that there has been relatively little work done on optimizing the hardware and software structures of these machines. While there are a huge number of multi-FPGA systems proposed and constructed, there are almost no studies investigating what the best architectures really are. The software systems suffer as well, with many individual optimizations proposed, but no real understanding of how to bring these optimizations together to achieve the best possible results. Thus, today's systems are largely grab-bags of appealing features with little knowledge of what the systems should really look like.

In 1992 we were in the same position as many other potential multi-FPGA system builders. We saw the need for a new system, which in our case was an emulation system optimized for the demands of board-level prototyping. While we knew in general what the system should look like, there was little guidance on exactly what the hardware and software architecture should be. At this point, instead of blindly going forward with our best guesses on how the system should be, we set out to look at the underlying issues and determine what really were the right architectures. By performing a disciplined investigation of multi-FPGA hardware and software systems, we not only would learn how to build our prototyping system correctly, but would also give a better understanding of some of the issues to the multi-FPGA community at large. It is these investigations that form the heart of this thesis. While we have not yet built our prototyping system (the high-level design of which is contained in Chapter 6), our work has yielded insight into many of the problems facing today's multi-FPGA systems.

Our investigations have focused on several different areas. In the hardware arena, we examined the routing topologies of multi-FPGA systems. Although the FPGAs in a multi-FPGA system are reprogrammable, the connections between the FPGAs are fixed by the circuit board implementing the system. With the main bottleneck in current multi-FPGA systems being the limited bandwidth for inter-FPGA routing, the routing topology of a multi-FPGA system has a significant impact on the capacity, as well as performance, of a multi-FPGA system. In our investigations we examined how best to construct a multi-FPGA system, yielding topologies with significantly lower I/O requirements, faster performance, and higher bandwidth.

Since our goal was to construct a prototyping system for board-level designs, we also had to consider the issue of how to support the external interfaces of a logic emulation system. By examining how to solve this problem in general, we have shown that a generic interface transducer, a simple board with FPGAs and memories for filtering the incoming and outgoing datastreams, is capable of supporting many different types of protocols. It has a flexible architecture that enables it to be prefabricated, yet it still allows customization to a specific application's requirements. Also, the required customization is easy to develop, and a small amount of programming in a high-level hardware description language is usually sufficient.

We have also spent time investigating the software required to map to a multi-FPGA system. Perhaps the most critical of these algorithms is partitioning, the process of splitting an input mapping into pieces small enough to fit into the individual FPGAs in a multi-FPGA system. While much work has been done on this problem previously, there was little insight into the best possible approach. Many researchers have proposed software solutions, but there was little comparative assessment of different approaches. Thus, one major portion of our work was to sift through the different approaches, and develop an algorithm that brings together the best possible set of optimizations to achieve the best overall results. This produced a survey of many of the different approaches, with each optimization implemented in a common framework, and compared quantitatively on a set of benchmark circuits. We now have an algorithm that produces results significantly better and faster than any other published approach.

We also recognized that flexible routing topologies are becoming more common in multi-FPGA systems, with architectures capable of significant customization in their interconnection patterns being proposed. To deal with these systems, it became apparent that the mapping tools not only needed to be fast and high-quality, but they also must be able to adapt to an arbitrary topology. This is not only important for flexible multi-FPGA hardware, but it also results in software that can easily be adapted to different machines. In pursuit of this goal, we have developed partitioning software capable of mapping to an arbitrary multi-

FPGA system topology, as well as routing software that can do the same. This routing software has the added advantage that it improves the quality of the mapping over current approaches, yielding both better-quality mappings and faster run-times. This latter benefit is due to the fact that poor routing complicates the following mapping steps, and thus by improving the routing quality we decrease the total mapping time.

The result of all of this work is new insights into how best to build and use multi-FPGA systems, as well as solutions to some of the outstanding problems. These results are not just important for building prototyping systems like the one we envisioned, but to multi-FPGA systems in general. The routing topologies are generally useful, offering improved topologies for both emulation and custom-computing. The mapping tools provide fast, high-quality mapping solutions for arbitrary system topologies. Finally, the interface transducers deliver a generic solution for interfacing both current single-chip emulators, as well as future board-level prototyping systems.

This thesis details the results of these investigations. It begins with a discussion of the underlying technologies that make multi-FPGA systems practical, including a survey of FPGAs and other logic implementation technologies in Chapter 2, as well as the application domains of multi-FPGA systems in Chapter 3. Chapter 4 delves deeper into one of the most promising application areas for multi-FPGA systems, logic emulation. Chapter 5 then gives an overview of current multi-FPGA system hardware. With the background provided by these chapters, we then present some new investigations and approaches to multi-FPGA system hardware. In Chapter 6 we present a new multi-FPGA system optimized for the prototyping of board-level designs. This system combines the inclusion of arbitrary devices with a flexible and extensible architecture to provide an extremely adaptable prototyping solution. Then, Chapter 7 examines routing topologies for multi-FPGA systems. This yields constructs with lower I/O demands, smaller delays, and higher bandwidth than standard topologies. Finally, Chapter 8 investigates how to support external interfaces for logic emulators, one of the significant problems with current systems.

The discussion of multi-FPGA system software begins with a general introduction in Chapter 9. Chapter 10 considers logic partitioning for multi-FPGA systems, presenting a comprehensive investigation into bipartitioning techniques. This results in a combined partitioning algorithm that achieves results better than the current state of the art, in significantly less time. Chapter 11 presents an iterative partitioning algorithm that automatically adapts to an arbitrary topology. This allows bipartitioning to be applied to any multi-FPGA system, as well as multi-sectioning and multi-way partitioning algorithms where appropriate. Finally, Chapter 12 investigates pin assignment algorithms for multi-FPGA systems. This speeds up the placement and routing time for multi-FPGA systems and produces higher quality results. We conclude this thesis with some overall results and potential future work in Chapter 13.