Chapter 3. Multi-FPGA System Applications

In Chapter 2 we discussed many different implementation technologies for digital logic, considering primarily how these various mediums support standard circuit designs. However, with the development of FPGAs, there are now opportunities for implementing quite different systems than were possible with other technologies. In this chapter we will discuss many of these new opportunities, especially those of multi-FPGA systems.

When FPGAs were first introduced, they were primarily considered to be just another form of gate array. While they had lower speed and capacity, and had a higher unit cost, they did not have the large startup costs and lead times necessary for MPGAs. Thus, they could be used for implementing random logic and glue logic in small volume systems with non-aggressive speed and capacity demands. If the capacity of a single FPGA was not enough to handle the desired functionality, multiple FPGAs could be included on the board, distributing the functionality between these chips.

FPGAs are more than just slow, small gate arrays. The critical feature of (SRAM-based) FPGAs is their in-circuit reprogrammability. Since their programming can be changed quickly, without any rewiring or refabrication, they can be used in a much more flexible manner than standard gate arrays. One example of this is multi-mode hardware. For example, when designing a digital tape recorder with error-correcting codes, one way to implement such a system is to have separate code generation and code checking hardware built into the tape machine. However, there is no reason to have both of these functions available simultaneously, since when reading from the tape there is no need to generate new codes, and when writing to the tape the code checking hardware will be idle. Thus, we can have an FPGA in the system, and have two different configurations stored in ROM, one for reading and one for writing. In this way, a single piece of hardware handles multiple different functionalities. There have been several multi-configuration systems built from FPGAs, including the just mentioned tape machine, generic printer interface cards with configurations for specific printers, pivoting monitors with landscape and portrait configurations, as well as others [Xilinx92, Fawcett94].

While the previous uses of FPGAs still treat these chips purely as methods for implementing digital logic, there are other applications where this is not the case. A system of FPGAs can be seen as a computing substrate with somewhat different properties than standard microprocessors. The reprogrammability of the FPGAs allows one to download algorithms onto the FPGAs, and change these algorithms just as general-purpose computers can change programs. This computing substrate is different from standard processors, in that it provides a huge amount of fine-grain parallelism, since there are many logic blocks on the chips, and the instructions are quite simple, on the order of a single five input, one output function. Also, while the instruction-stream of a microprocessor can be arbitrarily complex, with the function computed by the logic changing on a cycle by cycle basis, the programming of an FPGA is in general held constant throughout the execution of the mapping (exceptions to this include techniques of partial reconfigurability [Lysaght94, Hadley95, Jones95, Wirthlin95], which change a portion of the logic while the rest of the system continues operating). Thus, to achieve a variety of different functions in a mapping, a microprocessor does this temporally, with different functions executed during different cycles, while an FPGA-based computing machine achieves variety spatially, having different logic elements compute different functions. This means that microprocessors are superior for complex control flow and irregular computations, while an FPGA-based computing machine can be superior for data-parallel applications, where a huge amount of data must be acted on in a very similar manner. Note that there is work being done on trying to bridge this gap, and develop FPGA-processor hybrids that can achieve both spatial and limited temporal function variation [Ling93, Bolotski94, Maliniak94].

There have been several computing applications where a multi-FPGA system has delivered the highest performance implementation. An early example is genetic string matching on the Splash machine [Gokhale90]. Here, a linear array of Xilinx 3000 series FPGAs was used to implement a systolic algorithm to determine the “edit distance” between two strings. The edit distance is the minimum number of insertions and deletions necessary to transform one string into another, so the strings “flea” and “fleet” would have an edit distance of 3 (delete “a” and insert “et” to go from “flea” to “fleet”). As shown in [Lopresti91], a dynamic-programming solution to this problem can be implemented in the Splash system as a linear systolic circuit, with the strings to be compared flowing in opposite directions through the linear array. Processing can occur throughout the linear array simultaneously, with only local communication necessary, producing a huge amount of fine-grain parallelism. This is exactly the type of computation that maps well onto a multi-FPGA system. The Splash implementation was able to offer an extremely high performance solution for this application, achieving performance approximately 200 times faster than supercomputer implementations. There have been many other applications where a multi-FPGA system has offered the highest performance solution, including: mathematics applications such as long multiplication [Bertin89, Vuillemin95], modular multiplication [Cuccaro93], and RSA cryptography [Vuillemin95]; physics applications such as real-time pattern recognition in high-energy physics [Högl95], Monte Carlo algorithms for statistical physics [Monaghan93, Cowen94], second-level triggers for particle colliders [Moll95], and Heat and Laplace equation solvers [Vuillemin95]; general algorithms such as Monte Carlo yield modeling [Howard94b], genetic optimization algorithms [Scott95], stereo matching for stereo vision [Vuillemin95], hidden Markov Modeling for speech recognition [Schmit95], and genetic database searches [Lopresti91, Hoang93, Lemoine95].

One of the most successful uses for FPGA-based computation is in ASIC logic emulation. The problem is that the designers of a custom ASIC need to make sure that the circuit they designed correctly implements the desired functionality. Software simulation can perform these checks, but does so quite slowly. In logic emulation, the circuit to be tested is instead mapped onto a multi-FPGA system, yielding a solution several orders of magnitude faster than software simulation. A more detailed discussion of logic validation options, and how logic emulation fits into this process, can be found in Chapter 4.

There are many different types of FPGA-based systems. In this thesis we concentrate on multi-FPGA systems, namely those systems that contain more than one FPGA, and are constrained by the inter-chip connection topology. Before we discuss what kinds of systems fit this multi-FPGA system model, we first need to detail some of the systems that do not fall into this category. These are "topology-less" systems, systems where there are no specific, predefined connections between the FPGAs in the system. Topology-less systems are single-chip and custom multi-chip systems, which includes both the standard gate array applications of FPGAs, as well as the multi-mode hardware applications. Here the user is not constrained by any specific board topology, since the printed circuit board will be designed and fabricated after the mappings to the FPGAs have been determined, and these mapping will dictate the board interconnection pattern. Thus, if there will be more than a single FPGA in the system, the connections between these FPGAs are designed in response to the needs of the specific application. This tends to make the inter-FPGA routing problem much less significant than in the multi-FPGA domain, greatly altering the solutions adopted.

While topology-less systems have been the subject of significant amounts of work [Kuznar93, Woo93, Kuznar94a, Chan95, Huang95], it is not clear how important this domain really is. This is because many of these systems have their mapping finalized after the circuit board is sent for fabrication. In some cases, the need to get the system completed is significant enough that the board design cannot wait for the FPGA's logic to be completely specified. In others, even though a mapping may have been generated for the FPGAs before the board was designed, new functionality or bug fixes may mandate revisions to the FPGA mappings. In either case, the final mapping to the FPGAs must fit within the fixed inter-FPGA topology, and the requirements of a fixed topology move these systems into the multi-FPGA system domain. Note that some multi-FPGA systems are actually treated as a set of single-chip mappings, with each FPGA having a specific role independent of the other chips in the system, and the communication protocol between the chips is fixed. These systems would be considered topology-less, since they also avoid most multi-chip implementation issues.

As we just discussed, the primary characteristic of a multi-FPGA system is that it includes several FPGAs connected in a constrained routing topology. Within this multi-FPGA system domain, the most significant attribute is whether the system will be the target of automatic mapping software, or if it will be used primarily for hand-generated mappings. This difference can have a significant impact on the structure of the multi-FPGA system, since features that automatic-mapping software can best optimize for are often quite different from those that are appropriate for a human designer.

The ideal answer is to have all mappings to a multi-FPGA system generated by automatic-mapping software, since the hand-mapping process is often quite complex and time-consuming. Thus, if automatic-mapping software could deliver high-quality mappings quickly, there would be no reason for a human to waste time on this process. However, the problem is that current automatic-mapping software usually delivers much lower quality mappings than a human can achieve, especially on simple or highly-structured circuits. Thus, for many domains, automatic-mapping software simply produces unacceptable results, and the user is either forced to map the circuit by hand, or not use a multi-FPGA system at all.

To some extent, the applications for multi-FPGA systems can be seen as a spectrum, with highly structured (and thus easily hand-mapped) circuits at one end, and highly irregular (and thus impossible to hand-map) on the other. Currently, automatic-mapping software is only able to support the highly irregular mappings, not because the tools are somehow able to provide particularly good results for these circuits, but because the circuits are too complex for a human to do a reasonable job of hand-mapping. Thus, logic emulation tasks, where quite complex circuits are mapped to a multi-FPGA system, rely completely on automatic-mapping software. However, currently the more highly-structured applications are almost exclusively the domain of hand-mappings. One of the major research areas in multi-FPGA systems is developing automatic-mapping software that can produce higher-quality mappings. As this happens, the portion of the spectrum handled by automatic-mapping software will grow, taking over some of the mapping tasks currently the exclusive domain of the hand-mapping approach. This is a welcome development not only from the standpoint of lowering the amount of effort needed to use a multi-FPGA systems, but also because it increases the application domains of multi-FPGA systems. This is because there are many circuits that could perform well on a multi-FPGA system, but the unacceptable quality delivered by current automatic-mapping software, as well as the excessive effort necessary to hand-map these applications, forces these users to seek other solutions. Thus, while automatically-mapping circuits to a multi-FPGA system currently has a somewhat restricted application domain, as the automatic-mapping software improves this domain will become increasingly important. In this thesis we concentrate on hardware and software systems for multi-FPGA systems designed to support automatically-mapped circuits.

This distinction between multi-FPGA systems for hand-mappings and those for automatic-mappings is quite significant, and affects the entire implementation effort. When mapping by hand, the algorithm to be mapped is carefully chosen to fit the requirements of the multi-FPGA system, and may be quite different from the approach taken when the target is a microprocessor or other implementation medium. Most or all steps in the mapping are done by the user, who exerts significant effort to create the best possible mapping. The multi-FPGA system itself is designed to be simple and regular, with an emphasis on local communication, since this matches the structure of the circuits that can reasonably be hand-mapped. While this is how many of the high performance mappings have been developed for multi-FPGA systems, it limits the areas in which a multi-FPGA system can be used. In many situations, potential beneficiaries of a multi-FPGA system's performance do not have the time or the expertise to create a hand-mapping of their application, or the application may be too complex for anyone to reasonably map by hand. Thus, they will never adopt this strategy. However, hand-mapping may be necessary in some applications to achieve reasonable speedups over a microprocessor implementation, since current automatic-mapping software cannot develop truly high quality mappings.

In multi-FPGA systems designed for automatically-mapped circuits, the burden of taking a circuit from specification to implementation is handled by software (though variants that harness software for only lower-level portions of the mapping, or as assistants to a hand-mapping approach, are possible). Currently this is a fairly restricted class of systems, since the automatic mapping software often does not deliver nearly the mapping quality of hand-mappings. However, as the tools get better we should see more applications using automatic mapping software. Also, if multi-FPGA systems are to ever become commonplace, they will need to be mapped to automatically, because only a small portion of the possible users are willing and able to hand-map their circuits and algorithms.

The distinction between multi-FPGA systems for hand-mappings and systems for automatic mappings is more than a software issue. The design of the multi-FPGA system itself may be altered by how mappings are generated. For example, a system created for hand-mappings will usually have a simple topology, so that it is easier for the designer to use the system. However, complex and irregular features that automatic-mapping software can optimize for could be included in a system targeted for automatic mappings, while these features would only complicate the problem for a designer creating a mapping by hand. The pin permutations of Chapter 7 are a good example of this. While they can improve the quality of automatically generated mappings, they are probably too irregular to be used for hand-mappings.

In this thesis, we will be concerned primarily with multi-FPGA systems designed to support automatically-mapped circuits, though some of the results will also be applicable to hand-mappings as well. This bias is important, because many of the concerns of multi-FPGA systems with constrained topologies are quite different from topology-less systems. As indicated earlier, when a mapping does not need to fit into a predefined topology, the inter-FPGA routing problem is much less of a concern. Also, mapping turnaround time is a much greater concern in the systems we consider, since a fixed topology can be ready to use instantly, while custom multi-FPGA systems will need to wait for boards to be fabricated.

In the next chapter we will discussion logic validation, and how multi-FPGA systems can form an integral part of a complete validation strategy. Chapter 5 will then provide an overview of current multi-FPGA system hardware. Multi-FPGA system software is covered in Chapter 9.