Chapter 6. Springbok

Introduction

As pointed out in Chapter 4, logic emulation should be an important part of any logic validation environment. To perform logic emulation requires an appropriate hardware substrate and an integrated software mapping solution. In Chapter 5 we described many multi-FPGA systems. In general, they provide a constrained amount and type of resources in a fixed hardware system. For ASIC emulation tasks this is usually sufficient. However, board-level designs, designs where multiple chips are connected on a circuit board, will not fit onto these systems. The problem is that many of the chips in such systems are quite complex. Thus, if they had to be mapped onto a standard multi-FPGA system, they would require a huge amount of FPGA resources. In some cases, this process may not even be possible, since the manufacturers of these complex chips will (rightfully) be reluctant to divulge the exact implementation details of their chips. However, most or all of the chips used to construct the board-level design will be prefabricated. Thus, if we could include the actual chips into the logic emulation, this will yield a much more efficient implementation.

While some multi-FPGA systems allow the inclusion of a few arbitrary chips, most multi-FPGA systems are not designed to support a emulation style primarily dependent on premade components. This is because these systems rely primarily on FPGAs to implement the logic in the system, and the number of premade components is limited.

In the rest of this chapter we will discuss Springbok, a system to aid in the development and debugging of board-level designs. It has a flexible topology and resource mix, with the ability to include arbitrary chips into the system. While there are other multi-FPGA systems that have much of Springbok's capabilities (such as COBRA [Koch94] and MORRPH [Drayer95], described in Chapter 5), Springbok predates all of these systems [Hauck94]. An alternative approach to prototyping with a multi-FPGA system is the Aptix field-programmable circuit boards, which offer a sea-of-holes for plugging in arbitrary devices. However, the system is based around very expensive FPIC chips and multi-layer boards. Also, since the Aptix systems are based around FPIC hubs, there is no logic capacity (other than the chips added to the topology) for handling random logic, chip slow-down circuitry, or Virtual Wires multiplexing.

In the next section we discuss the proposed Springbok architecture, and then the mapping tools. We also consider Field-Programmable Interconnects (FPICs), and flexible multi-FPGA systems, which are promising approaches to circuit board testing, and compare them to the Springbok system.

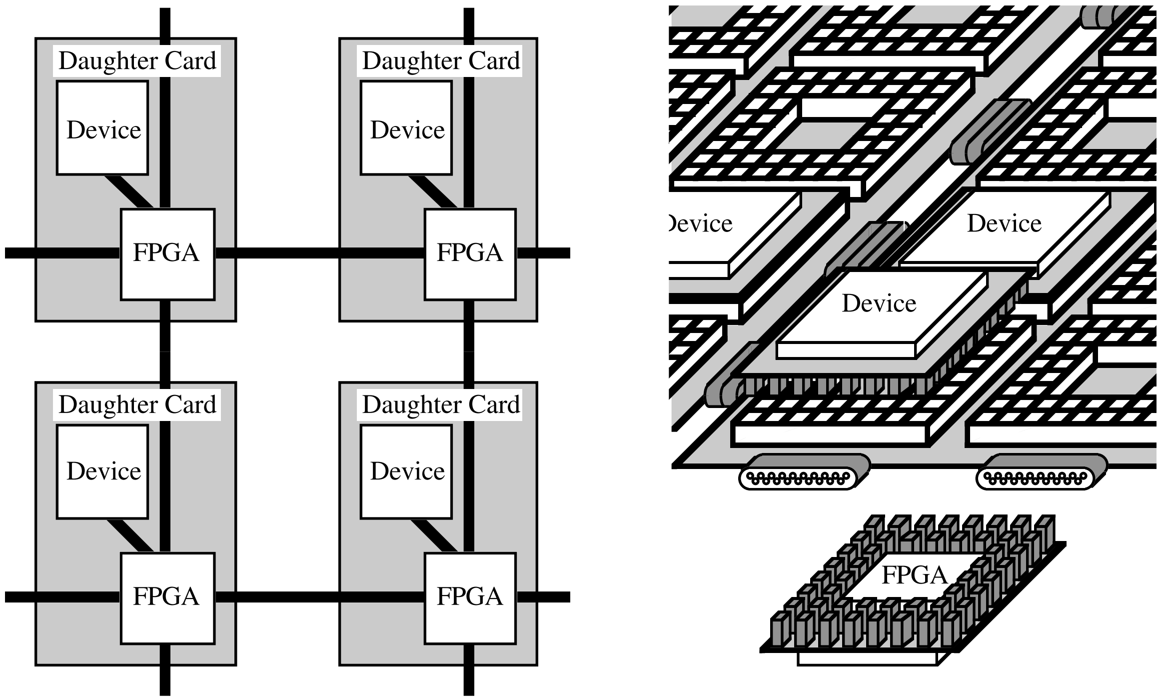

Figure 40. The Springbok interconnection pattern (left), and two connected Springbok baseplates with four daughter cards (right). The card at front is similar to the other daughter cards, but is shown upside-down.

The Springbok Architecture

The Springbok system is based on the philosophy that to develop and test board-level designs one needs a practical way to perform incremental development and testing using many of the actual chips of the final system without incurring the effort and expense of either wire-wrap or complete board fabrication. Our approach is to allow the important, complex chips comprising a design to be embedded in an FPGA-based structure, which uses these FPGAs for both the routing and rerouting of signals, as well as the implementation of random logic (Figure 40). To allow a specific circuit to be implemented in this structure, the Springbok system is comprised of a baseplate with sites for daughter cards. The daughter cards are large enough to contain both an arbitrary device on the top, as well as an FPGA on the bottom. Note that the device can be a chip, such as a processor or memory, I/O elements such as switches and LCD interfaces, or whatever else is necessary to implement the system. If necessary, daughter cards can be built that span several locations on the baseplate to handle higher space or bandwidth requirements. The daughter cards plug into the baseplate, which handles power, ground, and clock distribution, FPGA programming, and inter-daughter card routing. The baseplates include support for communicating with a host computer, both for downloading programming and for uploading data captured during prototype runs. The baseplates are constructed such that they can be connected together with each other, forming an arbitrarily large surface for placing daughter cards. The inter-daughter card routing structure is a 1-hop mesh, with the specific pin interconnections as detailed in Chapter 7. In many ways, this approach is similar both to mesh-connected multiprocessors, as well as the approach suggested in [Seitz90].

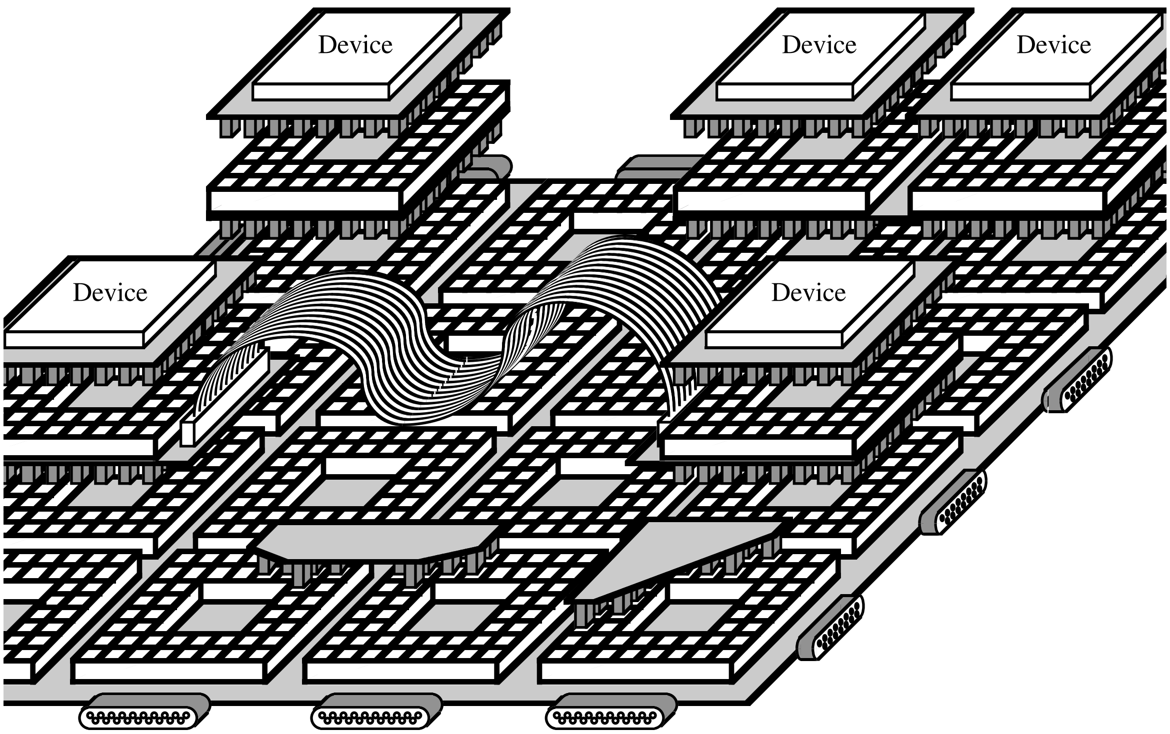

Figure 41. Non-device daughter cards and extender cards, including cards to add more FPGA logic (top left), bandwidth (double-sized card at top right), long-distance communication (middle), and edge bandwidth (bottom middle and bottom right). All but the edge bandwidth cards have FPGAs on the bottom.

An important feature of the Springbok system is the ability to insert system-specific chips on daughter cards placed into the array. This also allows us to include other, specialized daughter cards. For example, early in the design cycle the specific chips to be used to implement much of the logic may be unspecified. Thus, instead of inserting only chip-carrying daughter cards into the array, other cards with only FPGAs on them could be included. As in most other FPGA systems, there is also the potential that the simple connection scheme described above will not be able to accommodate all of the logic or routing assigned to a given location. However, as opposed to a fixed multi-FPGA system, we can insert new "extender" cards between a daughter card and the baseplate to deal with these problems (Figure 41). For example, if the logic assigned to a given FPGA simply will not fit, an extender card with another FPGA can be inserted so that it can handle some of the logic. If too many signals need to be routed along a given link in the mesh structure, an extender card spanning several daughter card positions can be added, with new routing paths included on the inserted card. Note that while most routing limitations could be dealt with via Virtual Wires (Chapter 9), added cards for routing will still serve a purpose by reducing the reliance on Virtual Wires, thus decreasing cycle time and increasing logic capacity. For signals that must go long distances in the array, sets of extender cards with ribbon cable connections can be inserted throughout the array to carry these long-distance wires. Also, at the edge of the mapping where edge effects can limit available bandwidth, dummy daughter cards which simply contain hardwired connections between their neighbors can be inserted. Thus, the Springbok approach to resource limitations is to add resources wherever necessary to map the system. In contrast, a fixed array cannot afford a failure due to resource limitations, since it would then have to redo the costly step of mapping to all of the constituent FPGAs. Thus fixed arrays must be very conservative on all resource assignments, underutilizing resources throughout the system, while Springbok simply fixes the problems locally as they arise.

Another important benefit of Springbok is how it supports hardware subroutines. In many design environments there will not be just one system developed, but instead a family of products may be built. Many of these products have subsystems shared across the entire family. For example, a company developing disk controllers would have SCSI interfaces on most of its products, as well as an interface to the host computer's bus (e.g. [Katz93]). In the Springbok model, such heavily replicated elements can be fabricated or wire-wrapped as a single daughter card, and from then on used as a hardware subroutine for all subsequent designs. One could use a similar approach in a wire-wrap domain by developing prototyping boards with these functionalities fabricated on the board. However, in such a system one must fix the number and type of these subroutines ahead of time, and this mix cannot be increased. Thus, a prototyping board designed with one SCSI interface would be useless for prototyping a two SCSI port controller, and a board for one type of computer bus could not be used for a different bus. In Springbok this is not an issue, because the number and type of daughter cards can vary from system to system, and cards with functionality not even considered in the first system of a family can be added easily in later designs.

While the construction of the Springbok system fixes many problems encountered in rapid-prototyping, there are two important concerns that remain. First, the physical wires in the target system are replaced with digital connections in Springbok. Second, while speeds achieved may be orders of magnitude faster than simulation, they will still be less than the final system speeds. The problem with the way Springbok handles wires is that not all physical wires in mapped systems are used in a purely digital, unidirectional manner. While some systems use analog subsystems, this is beyond the scope of Springbok, and any such subsystems will need to be either included as specialized daughter cards, or not handled at all. For non-unidirectional flow, signals such as buses and bi-directional signals must be handled. As shown in Figure 42, a tristate bus can be replaced by a pure logic structure [Butts91]. Note that the circuit shown at right implements a floating low bus. A floating high bus can be constructed by inverting the inputs and outputs of the circuit. Once the tristate bus is converted to the implementation shown, the gates implementing the communication can be mapped to the FPGA identically to the rest of the random logic in the system.

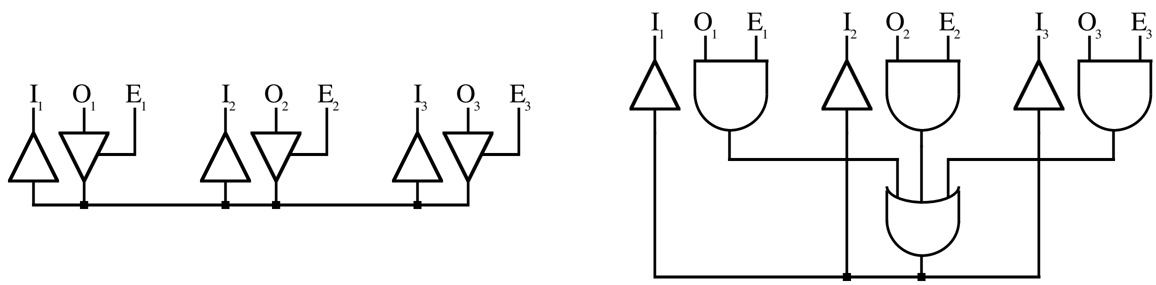

Figure 42. Method for handling bi-directional signals [Butts91]. The tristate bus at left is replaced with the logic structure at right. Note that all signals are unidirectional in the structure at right.

The second problem mentioned above, that a Springbok mapping will almost always be somewhat slower than the system being prototyped, causes several concerns. First, one of the goals of a prototype is to test the system in the actual target environment. However, other systems that interact with the circuit will be expecting certain timing assumptions to be upheld, such as those contained in a bus protocol. This same problem has been encountered by ASIC logic emulator vendors such as Quickturn, and their customers have dealt with this by building protocol-specific buffering solutions. Such an approach would work equally well with Springbok. An even better answer is detailed in Chapter 8, which presents a general solution to the logic emulator interface issue.

The second manifestation of the slow-down problem is that the chips used in the system must be slowed down as well. Many chips can simply be operated at the slower clock rate and still function properly. Springbok mappings will operate fast enough that charge leakage from dynamic nodes will not be an issue for most chips. The primary concern is for chips with phase-locked loops which can only operate at certain clock frequencies. Solutions to this problem include stalling processor chips either through manipulation of an explicit stall signal or insertion of stall instructions into the chip's instruction stream, dummy data and buffering for pipelined chips, or even replacement of very restrictive chips with slower members of the chip family or mapping it into FPGA logic.

The Springbok Mapping Tools

As is found with logic emulators, the target architecture is only half the system. Just as important as a flexible medium for implementing the desired functionality is the need for an integrated software system to map target designs onto the architecture.

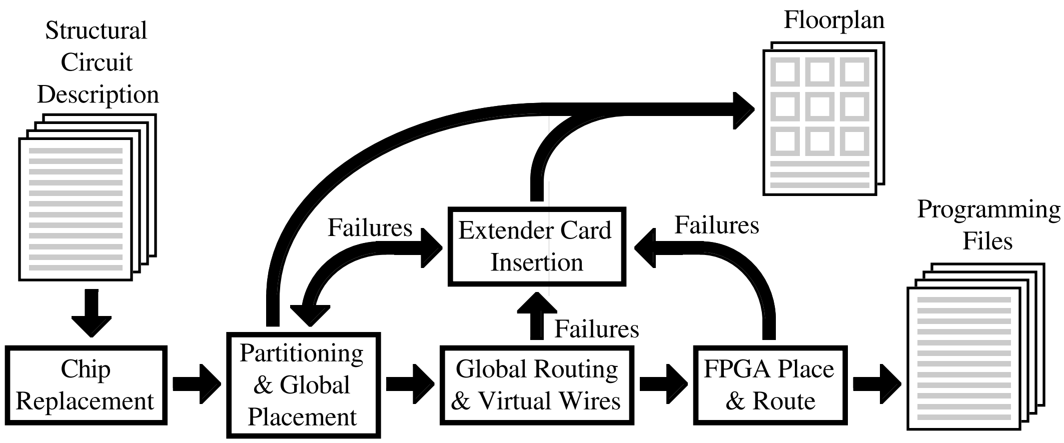

Figure 43. Springbok software flow diagram.

The overall software system flow in Springbok is shown in Figure 43. The mapping tools start with a structural description of the system to be mapped, where the system logic can either be assigned to specific chips, or possibly have portions of random logic which have not yet been assigned to chips. Through a series of steps this logic is mapped down to a form that can be placed onto the Springbok architecture, producing both a floorplan describing how daughter cards and extender cards must be placed onto baseplates, and programming files for configuring the individual FPGAs.

The first step in the mapping process is chip replacement. We expect Springbok to handle systems ranging from those with only the critical components specified (the rest left as random logic) to those with most or all of the logic implemented with specific chips. All logic not assigned to chips in the original structural description will be handled by FPGAs in Springbok, and the Springbok system may decide to assign logic mapped to chips in the description to FPGAs as well. For example, in a completely specified design, a chip that simply buffers signals might be included in the original description, but would not be necessary in the

Springbok prototype. Thus, by replacing this chip with the corresponding logic, the resulting mapping might save some hardware. Also, logic placed into a specific FPGA in the source description restricts the placement choices in the Springbok implementation, and might also be removed. Finally, since Springbok is fairly FPGA-intensive, mappings will be more compact if a certain percentage of the system is random, unassigned logic. Thus, during chip replacement the software could use a greedy algorithm to replace chips with random logic until the optimal percentage of random logic is reached. That is, greedily removing the chip with the least logic assigned to it should allow the removal of the most number of chips before reaching the target percentage of logic in the FPGAs, thus reducing the total amount of hardware. It might also be useful to break up the highest fanout chip (i.e., that chip connected to the most other chips) to ease the resulting routing congestion and delay, though how exactly to integrate these two considerations requires study. Note that the user can specify that any or all chips be left unreplaced.

The next step, partitioning and global placement, determines the overall layout of the Springbok mapping. It must decide which daughter cards to use, where to place them, and what logic will be implemented by each of these cards. Standard methods for partitioning such as k-way partitioning are difficult to use for this problem, since these algorithms ignore the locality required in mesh communication patterns, and also generally require that the number of partitions be fixed. As opposed to most current systems, the size and composition of the Springbok substrate is flexible, expanding or contracting to most efficiently map the target system. This flexibility makes the partitioning more difficult, but allows greater optimization opportunities. An overview of our partitioning approach is contained in Chapter 9, with further details in Chapter 10 and Chapter 11.

After partitioning and global placement, routing must be considered. The problem of deciding which intermediate FPGAs to route through is fairly straightforward, and is very similar to the single-FPGA routing problem. Thus, we can use much of the research on single-FPGA routing (such as [McMurchie95]) to handle this portion of the routing problem. However, this still leaves the problem of determining which individual I/O pins to use to route between the FPGAs. In order to handle pin assignment for multi-FPGA systems, we have developed a new technique that simultaneously reduces the routing resource usage, reducing both area requirements and delay, while also speeding up the mapping process (Chapter 12).

With partitioning, global placement, and routing completed, it is then necessary to place and route the individual FPGAs. For these tasks we can use one of several reasonable commercial software packages available. Some of their underlying algorithms are discussed in Chapter 2.

As shown in the software flow diagram, any mappings that fail are fixed by software that inserts extender cards into the Springbok array. Then, partitioning and global routing are rerun, maintaining as much of the previous mapping as possible while easing demands on the failing FPGAs. Note that this portion of the flow is also important for incremental alteration of a prototype. Specifically, as a system is debugged errors will need to be fixed, hopefully without requiring a completely new mapping of the circuit. By extending the partitioning and global routing steps to handle incremental alterations to fix failed FPGA mappings, we also have support for incremental alterations for bug fixes.

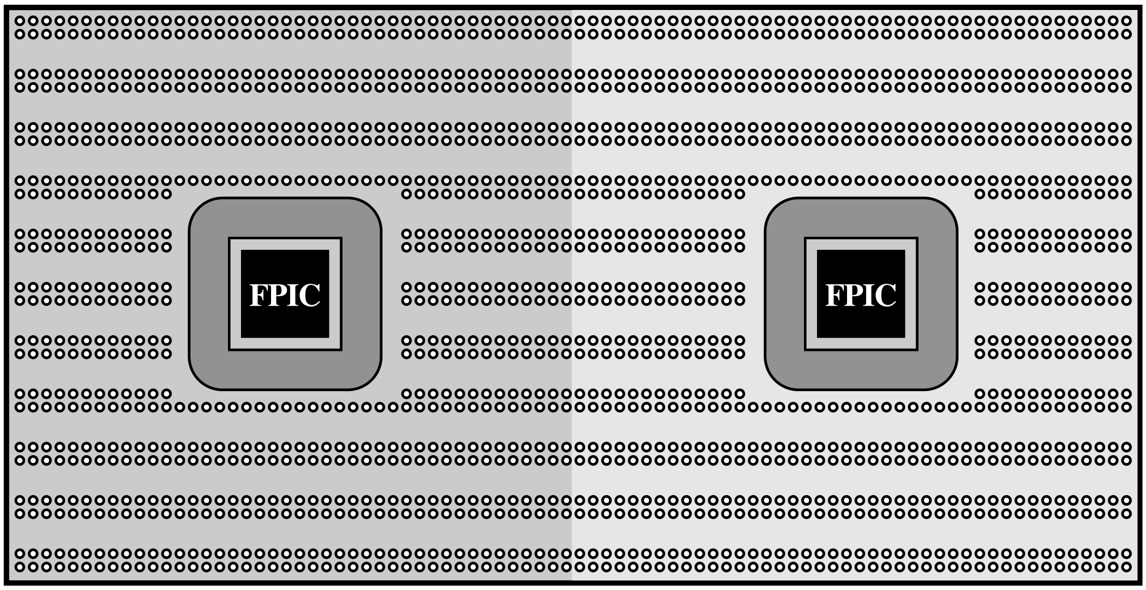

Figure 44. An FPIC based prototyping board [Aptix93a].

Springbok vs. Other Current Approaches

As mentioned earlier, another promising approach for the rapid prototyping of board-level designs is Field-Programmable Interconnects (FPICs), such as those developed by Aptix [Aptix93a]. An FPIC is a chip that can create an arbitrary interconnection of its pins. In the case of the Aptix chips, there are 936 user pins per chip, and the connections formed are passive path wire connections. This latter feature is helpful, since it means that the concerns Springbok has with bi-directional signals such as buses are easily handled with Aptix products. To use these chips for prototyping, Aptix provides circuit boards consisting of a large grid of holes for inserting arbitrary chips (Figure 44). These holes are grouped into sections, with all holes in a section leading to pins on the same FPIC. The FPIC chips communicate between themselves with direct hardwired connections. There are about 50-140 connections between each pair of Aptix FPICs (the exact number varies by the Aptix board chosen). Power and ground routing are handled by separate buses distributed throughout the grid, and jumpers can be used to connect chip pins to the power and ground lines. Clock distribution is supported with special low skew paths in the Aptix chips. Mapping to such a system is easier than for Springbok, since all that is necessary is to partition the chips into sections, minimizing wire crossings at the same time, and then route the signals through the FPICs.

There are several differences between Springbok and an FPIC approach to rapid-prototyping of board-level designs. As mentioned earlier, the software for an FPIC approach is simpler, and there is no difficulty with bi-directional signals. However, these benefits are outweighed by several problems. Most importantly, an FPIC approach is quite expensive. Since both the circuit boards necessary to handle the FPIC's high pincount package, as well as the FPICs themselves, use very aggressive technologies, they are much more expensive than standard chips and boards. In contrast, we expect the cost of a Springbok mapping to be significantly less, since it uses commodity FPGAs and simple circuit boards. Also, many Springbok mappings will use the FPGAs of the underlying routing structure to handle some of the circuit logic. Thus, much of the silicon used to implement Springbok will already be required in the target mapping, decreasing costs.

A second problem with the pure FPIC approach is that of flexibility. As stated earlier, if a mapping requires more pins than the Aptix board allows, there is no way of mapping it short of purchasing a larger board and extra FPICs, adding more expense. In contrast, the Springbok system easily expands to larger sizes, since all that is required to add more capacity is the addition of another baseplate. Thus, instead of requiring several Aptix boards in several sizes, Springbok baseplates can be used in any size mapping. Also, the Aptix boards have a fixed 50-140 pin limit on communication between FPIC sections of the grid. Again, if this number is exceeded, there is no way of mapping the prototype. In Springbok, capacity can be added to deal with any pin limitations. Also, Virtual Wires (Chapter 9), a method usable in Springbok to ease some pin limitations, cannot be directly used in an FPIC system since the FPICs do not have programmable logic with which to multiplex signals. Thus, to use Virtual Wires an FPIC system would have to add extra FPGAs, FPGAs that will also increase the number of pins in the mapping, pins which must also pay the per-pin costs.

Just as many of the Springbok FPGAs will be used to implement logic from the system being mapped, other portions of the FPGAs will be required to slow down chips. Both Springbok and FPIC mappings will operate slower than the target system. As discussed earlier, some chips cannot simply be clocked at a slower rate, but instead require special additional logic to operate correctly. In Springbok, this logic can be accommodated in the FPGAs connecting that chip into the Springbok routing structure. In an FPIC system, extra FPGAs would have to be added to handle this functionality, increasing total pin count. More importantly, these added FPGAs are not directly connected to the chip to be slowed, but instead must communicate with it through the FPIC. The FPIC introduces delays, reducing the portion of the clock period available in the FPGA slowing the chip, which makes it more difficult to properly perform these slowing functions.

A final limitation of the FPIC approach is that it does not support hardware subroutines well. As discussed earlier, in many design environments there are common subsystems used in several different circuits. In Springbok, these subsystems can be built into custom daughter cards, and then used freely in subsequent mappings. In an FPIC system one would need to develop a custom board (an activity Aptix supports with special software) which would contain the subsystem logic as well as the FPICs and the pin grid. This means not only that the resulting new board would be more complex than that necessary for the Springbok system, it also establishes a limit on both the number of such subsystems and the number of chip pins that can be used. This is because both the number of subsystems as well as the size of the pin grid is fixed on any specific Aptix board. Again, in the Springbok system, building a custom daughter card only fixes the contents of that daughter card. Springbok mappings are still free to choose the type and number of custom daughter cards to use, and the mapping can still grow to any size.

While we have spent most of this section discussing the difficulties with using a purely FPIC solution to board-level prototyping, an interesting alternative is to add FPICs into the Springbok framework. FPICs could be included on extender cards, cards which could help ease routing in hotspots of a given mapping. Also, special connections could be built into the baseplates which could lead to a centralized FPIC hub. These baseplate connections would connect a small number of pins on every daughter card position to the centralized FPIC hub. In this way, a network for more efficiently handling long-distance connections could be built without requiring many ribbon-cable extender boards scattered throughout the array. In each of these cases, the FPICs are used sparingly. Hopefully, this could yield a system with all of Springbok's advantages, while harnessing the power of FPICs to perform relatively quick, arbitrary connection patterns. Whether the added functionality is worth the increased cost is unclear, and requires further study.

As mentioned earlier (Chapter 5), there have been many multi-FPGA systems developed. However, most of these are unusable for board-level prototyping, since they do not allow the inclusion of arbitrary chips into the array, or do so to a limited extent. Some systems, such as the G800 or the Paradigm RP, allow a small amount of custom boards to be included into the system. However, many multi-chip systems will require tens or hundreds of chips to be included into the prototype, something that is beyond the capabilities of these systems.

Two systems (which were proposed after Springbok) with a lot of potential for board-level prototyping are COBRA and MORRPH (Chapter 5). Like Springbok, these systems are based around a flexible interconnection topology that allows the system to grow to an arbitrary size. The COBRA system is based around tiles with fixed resource mixes, in a fixed mesh structure. However, adding a prototyping board with sockets for arbitrary devices, as well as boards with long-distance communication links, should not be difficult. The MORRPH system is based around a single tile type, with sockets for arbitrary devices. However, there is nothing in the system construction that precludes adding boards with different functionalities and capacities. Thus, if we combine COBRA's ability to include arbitrary cards with MORRPH's ability to include arbitrary devices onto these cards, we would achieve much of the flexibility of Springbok.

Conclusions

As we have shown, Springbok is a novel approach to the rapid-prototyping of board-level designs that offers many advantages over current systems. Its flexible architecture accommodates a great range of system sizes and topologies. With the ability to solve problems as they occur, Springbok more efficiently uses its resources than fixed FPGA-based systems, which require a very conservative style. Including arbitrary devices and subsystems into the Springbok structure allows even greater efficiency and accuracy. Finally, the use of FPGAs instead of FPICs for the routing structure reduces overall costs, adds flexibility, and more easily handles the functionality necessary to interface to timing-inflexible components.

In the following chapters, we will discuss many of the issues necessary to implement Springbok. Mesh routing topologies are covered in Chapter 7, yielding topologies with higher bandwidth, lower delay, and reduced I/O pin usage. Then we discuss how the external interfaces of a prototype can be supported (Chapter 8). On the software end of multi-FPGA systems, we consider an efficient bipartitioning algorithm (Chapter 10), as well as methods for recursively applying bipartitioning to an arbitrary topology (Chapter 11). We also cover pin assignment software (Chapter 12), which handles part of the global routing step for mapping to multi-FPGA systems.