

Springbok: A Rapid-Prototyping System for Board-Level Designs

Scott Hauck, Gaetano Borriello, Carl Ebeling
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Abstract

Multi-chip, board-level designs form a large portion of today's digital system designs. Unfortunately, traditional methods for debugging these designs, such as prototype fabrication, wire-wrap and software simulation, are inadequate. Prototype fabrication is time-consuming and it is difficult to isolate errors, wire-wrap is complex and error-prone, and simulation is too slow for full testing. Recent advances in FPGA-based systems offer hope for significant improvements in board-level prototyping, yet currently focus exclusively on ASIC prototyping. In this paper, we present Springbok, an integrated system for board-level prototyping that promises near-speed testing with the construction and modification simplicity of software simulation. It is composed of an integrated software system for mapping designs to a flexible, extensible hardware implementation platform. This allows designs to take advantage of FPGA flexibility while using the complex chips that will implement the final design.

Introduction

Developing today's complex and high-speed designs usually requires fabrication of a test board. This allows at-speed testing of the hardware design in the target environment. Unfortunately, there is a high lag time introduced, both because of the actual fabrication time and also because one must wait until late in the development cycle, since the entire design must be specified fully before it can be built. Thus, incremental development and testing is impossible. Bug fixes are difficult and time-consuming, especially with multi-layer boards, and isolating subcircuits for testing is almost impossible. The cost of multiple fabrication runs also makes this technique unattractive.

The traditional alternative for developing digital systems is to use simulation to debug portions of a system, and then use wire-wrap prototype for complete system testing. Unfortunately, wire-wrap has many problems, which has caused many designers to abandon it. Primarily, it requires a large amount of effort to connect the components, introducing a large potential for errors and a significant lag time. Even when the system is built, wire-wrap complicates testing because wiring cannot easily be changed to isolate subcircuits for fear of introducing new errors, and incorrect or defective wire-wrap must always be considered a potential culprit in any anomalous

behavior. Furthermore, wire-wrap cannot handle many high-speed systems.

Using software simulation for complete system debugging is promising, but has many problems of its own. While it offers a great deal of flexibility and removes concerns of implementation medium failures, it is much slower than the other alternatives. It is possible to speed the simulation up somewhat by abstracting away some details of individual chips, for example replacing a microprocessor with some predetermined instruction traces, but this can both limit the testable behaviors and introduce new errors. Even with these optimizations, simulation is still orders of magnitude slower than the target system. Also, without a physical prototype the system cannot be tested in the target environment, and much of a system's interface behavior remains untested. Each of these problems greatly limits the scope of testing possible in a purely simulation-based approach, making it unsuitable for many complex systems.

One can draw parallels between the problems of board-level prototyping and those of ASIC development. Until recently, ASIC design and debugging were also limited by the speeds of software simulation. However, FPGA-based logic emulation systems (such as those developed by Quickturn [Varghese93]) have been recently introduced, giving ASIC designers the benefits of software simulation with several orders of magnitude higher speed. However, the approach taken by ASIC logic emulators is not sufficient for designs involving multiple complex chips. While today's logic emulators are capable of mapping an entire microprocessor into FPGAs, they require an extremely large number of FPGAs to do so. When one considers a complete board-level design containing perhaps a microprocessor, several memories and other controller chips, it is clear that the cost to map these circuits into FPGAs is prohibitive. Even worse, elements such as A/D converters cannot be implemented in FPGAs at all. Thus, while the general concept of harnessing FPGAs to help the board-level designer has merit, the methods used for FPGA-based ASIC emulators are inadequate for system design.

In the rest of this paper we will discuss Springbok, a system being developed to aid in the development and debugging of board-level designs. We discuss the proposed architecture and mapping tools. We also consider Field-Programmable Interconnects (FPICs), another promising approach to circuit board testing, and

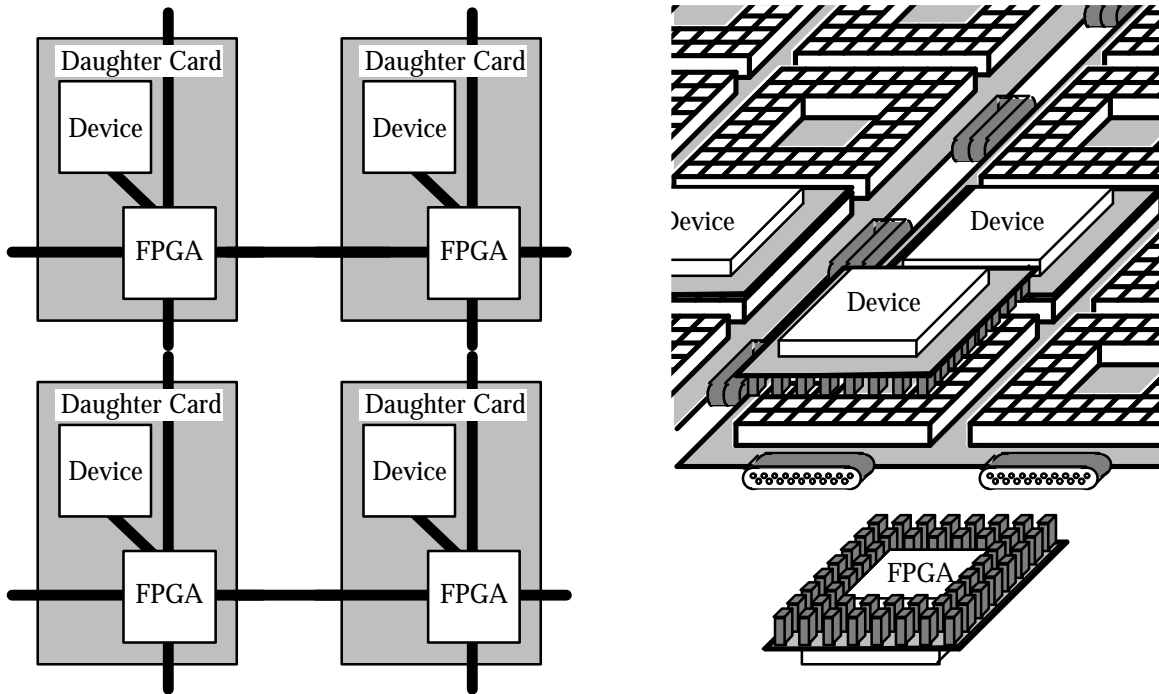


Figure 1. The Springbok interconnection pattern (left), and two connected Springbok baseplates with four daughter cards (right). The card at front is similar to the other daughter cards, but is shown upside-down.

compare them to the Springbok system. We then discuss our current status, as well as some overall conclusions.

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 use many of the actual chips of the final system for incremental development and testing, 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 1). To allow a specific circuit to be implemented in this structure, the Springbok system is composed of a baseplate with sites for daughter cards. The daughter cards are large enough to contain an arbitrary device on the top, as well as an FPGA on the bottom. Note that the device can be a chip, 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 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 [Hauck94a]. In many ways, this approach is similar to mesh-connected multiprocessors, as well as the approach suggested in [Seitz90].

An important feature of Springbok 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 adding only chip-carrying daughter cards into the array, 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 the logic or routing assigned to a given location. However, as opposed to a fixed FPGA array, we can insert “extender” cards between a daughter card and the baseplate to deal with these problems (Figure 2). If the logic assigned to a given FPGA will not fit, an extender card with another FPGA can be inserted to handle some of the logic. If too many signals need to be routed along a given link, 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 can be dealt with by Virtual Wires [Babb93], a method for multiplexing several circuit wires onto one physical connection, added cards for routing will reduce the reliance on Virtual Wires, thus decreasing both area and cycle time. For signals that must go long

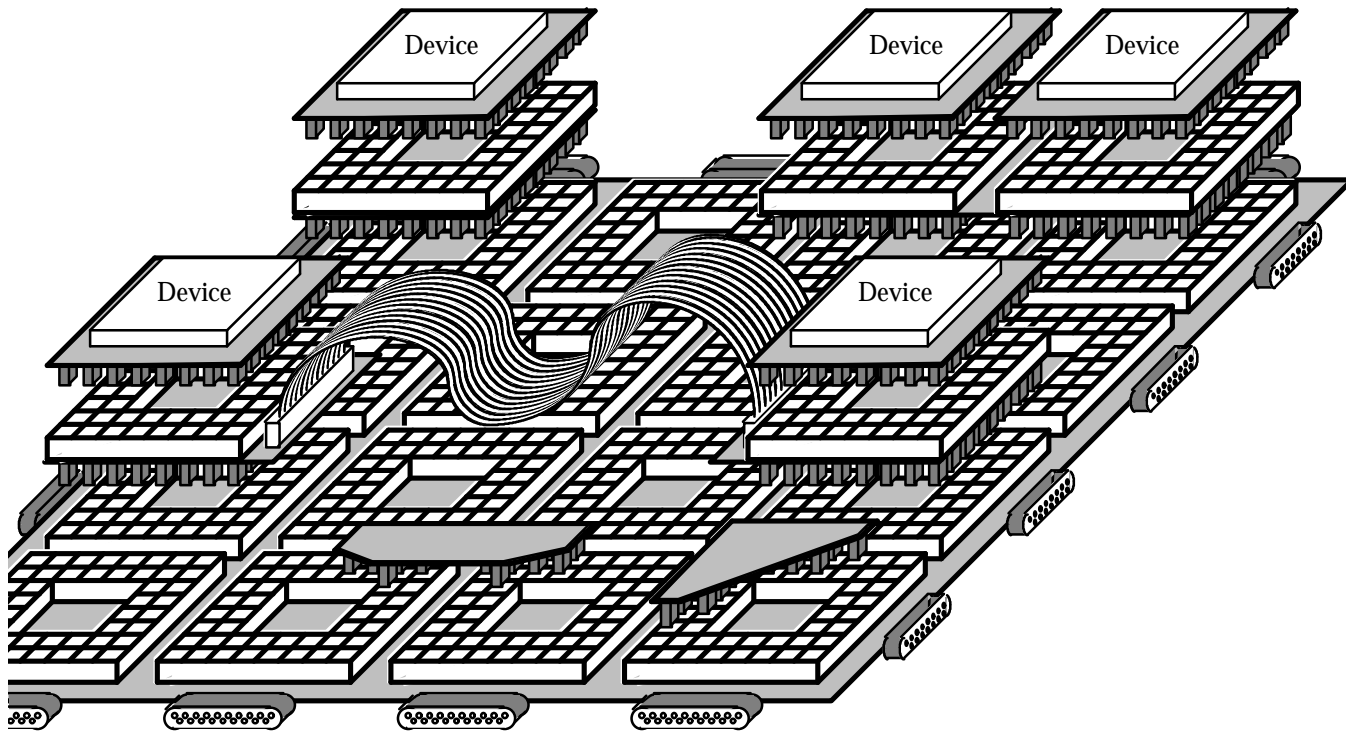


Figure 2. Non-device daughter cards and extender cards, including cards to add more FPGA logic (top left), bandwidth (double-sized card at top right), long-distance communication (middle), and edge bandwidth (bottom middle and bottom right). All but the edge bandwidth cards have FPGAs on the bottom.

distances in the array, sets of extender cards with ribbon cable connections can be inserted in 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 that simply contain hard-wired 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 the constituent FPGAs. Thus fixed arrays must be conservative on all resource assignments, underutilizing resources throughout the system, while Springbok simply fixes the problems locally as they arise. Note that since Springbok was introduced [Hauck94d], a similar system called Cobra has been developed [Koch94], which shares Springbok's ability to provide a flexible resource mix on a per-mapping basis.

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 Springbok, 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. The physical wires in the target system are replaced with digital connections in Springbok, and 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 systems 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. There are two ways these can be handled in Springbok (see Figure 3). For most systems, the chip

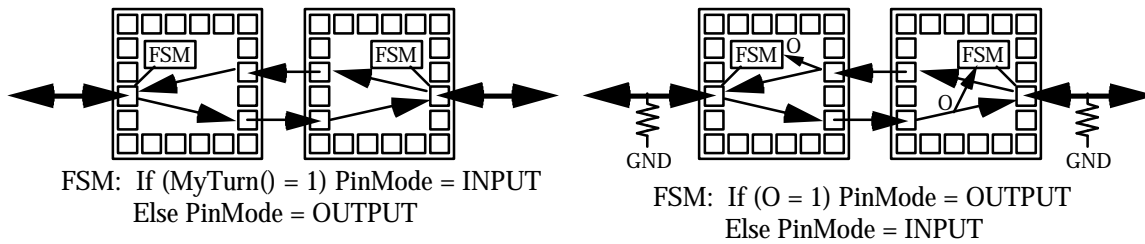


Figure 3. Two methods for handling bi-directional signals. Using an FSM when direction information can be maintained or inferred from control signals (left), and a wired-OR for less determinable directionality (right).

allowed to drive the signal can easily be determined either by the state of certain control signals, or by a simple state-machine. In these cases, logic can be added to reconfigure the unidirectional wires and pins according to the required signal flow. In cases where this may be too difficult to handle, a wired-OR (or wired-AND) can replace the signal, with pull-down resistors on all connected chip pins. Specifically, when no-one is driving the signal, all pins will be pulled down to 0, and the signal is 0. In such a case either no-one will be reading, or a pull-down resistor is included in the original circuit (cases where the circuit to be mapped has a pull-up resistor on the signal can be handled by a wired-AND). If the chip supposed to be driving the signal is outputting a 0, the same situation holds, and the signal is valid at 0. If a chip drives a 1, the FPGA this leads to can sense this value, determine that this is the chip that should be driving the signal, and configure the wires accordingly.

The second problem mentioned above, that a Springbok mapping will almost always be somewhat slower than the system being prototyped, causes several concerns. First of all, 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 [Hauck95a], which presents a programmable interface transducer board design that can handle many different protocols, as well as a discussion of how it can be applied in many situations. The second part of the slow-down problem is that the chips used in the system must be slowed down as well. Many chips can be operated at the slower clock rate and they will 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 phase-locked loops that can only operate at certain frequencies. Solutions include stalling processor chips 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 restrictive chips with slower members of the chip family or mappings into FPGA logic.

The Springbok Mapping Tools

As in the case of logic emulators, the target architecture is only half the system. Just as important as a flexible implementation medium is the need for an integrated software system to map target designs onto the architecture.

The overall software system flow in Springbok is shown in Figure 4. 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 that 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 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 proportion of the system is random, unassigned logic.

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

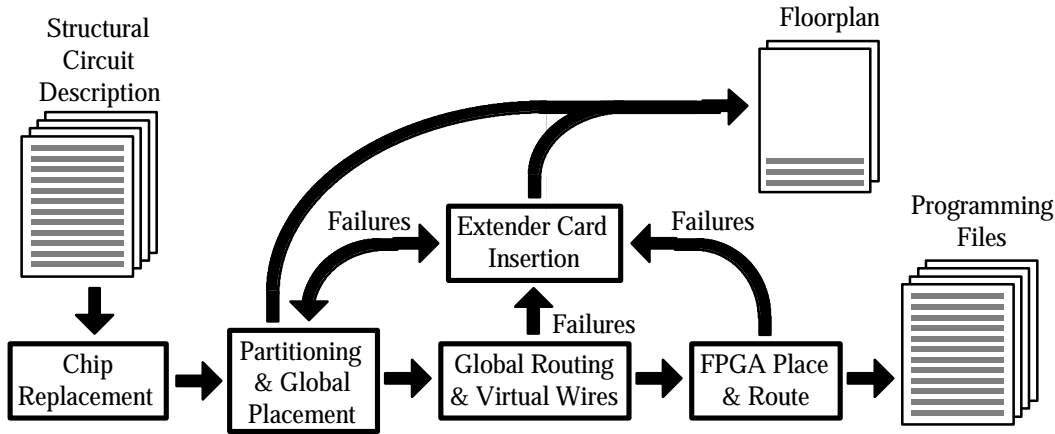


Figure 4. Springbok software flow diagram.

partitioning are difficult to use for this problem, since these algorithms ignore the locality required in mesh communication patterns, and generally require that the number of partitions be predetermined. As opposed to most current systems, the size and composition of the Springbok substrate is flexible, expanding or contracting to map the target system efficiently. This flexibility makes the partitioning more difficult, but allows greater optimization opportunities. To handle this problem, we have decided to apply iterative bi-partitioning, which has the advantages of both fast run-times, as well as the ability to handle limited interconnections. To apply iterative bipartitioning to an arbitrary topology, we have developed an algorithm to automatically decide the order of the partitionings necessary to optimize to an arbitrary topology [Hauck95c]. We then can apply our bipartitioning algorithm, which is based on the Fiduccia-Mattheyses variant of the Kernighan-Lin algorithm [Hauck95b]. Our optimized version of this algorithm is quite fast, and yields results significantly better than the current state-of-the-art.

After partitioning and global placement, routing must be considered. Deciding which intermediate FPGAs to route through is fairly straightforward, and is very similar to the single-FPGA routing problem. That is, the software must find routes in a fixed routing topology, with finite resources, simultaneously attempting to make all the routes fit and possibly optimize for delay. Thus, we can use the research on single-FPGA routing (such as [McMurchie95]) to handle this portion of the routing problem. However, this leaves the pin assignment problem, the problem of determining which I/O pins to use to route between the FPGAs. Specifically, if FPGAs A and B are directly connected, and the global router has decided to route a signal between them, there will be some connections between the FPGAs that will work better than others to route the signal. One connection may be closer to the route's destination than the others. To handle the pin assignment problem for multi-FPGA systems, we have developed a technique that reduces the routing resource usage in the system, reducing area requirements and delay,

while also speeding up the mapping process [Hauck94b, Hauck94c].

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. For Springbok we would simply need to ensure that the output of previous stages matches the place and route software's demands, and determine how to capture information on failures. This information is important because Springbok does not include significant margins to ensure every logic assignment to an FPGA will succeed, but instead fixes problems as they occur. 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, which maintain 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 bug fixes.

Springbok vs. an FPIC-based Approach

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 [Aptix93]. 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. These holes are grouped into sections, with all

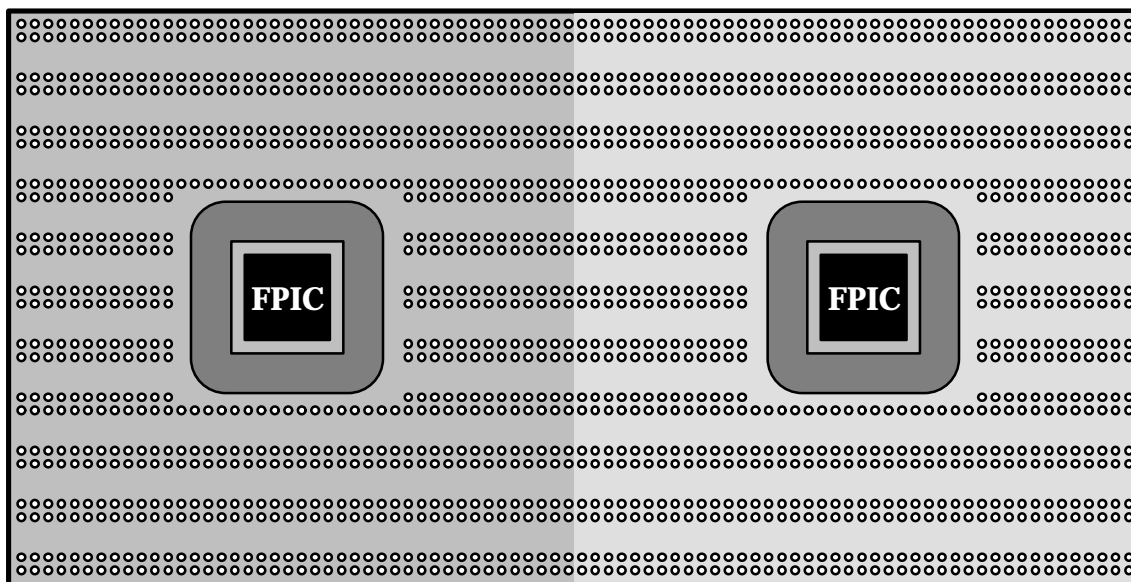


Figure 5. An FPIC based prototyping board [Aptix93].

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 while minimizing wire crossings, 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. If we use every available user pin on an Aptix board, the cost is about \$4.5 - \$5 a pin for the FPICs and boards themselves (This assumes that Aptix FPIC/D's [Aptix93] are used. Cost savings can be achieved by using FPIC/R's, but with a significant loss in testability). Unfortunately, one will rarely use all the pins, both because of fragmentation due to chip package sizes and because of the large step function on grid sizes. Specifically, boards are sized in roughly 1500 pin increments, forcing one to use a board that is 750 pins too large on average. Also, since a mapping that is a few pins too large for the board chosen cannot be mapped, the size of the FPIC board must be chosen conservatively, anticipating size increases during debugging. All of these factors will tend to drive actual per pin costs much higher than the baseline. In contrast, we expect the per pin cost of a Springbok mapping to be significantly less. Also,

since many Springbok mappings will use the FPGAs of the underlying routing structure to also handle some of the circuit logic, the number of chip pins found in the typical Springbok mapping will be less than an Aptix mapping. Thus, much of the silicon used to implement Springbok will already be required in the target mapping, serving to decrease 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 buying a larger board and extra FPICs, each costing several thousand dollars. In contrast, the Springbok system easily expands to larger sizes, with 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, the method used 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 that 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 of at least 5ns in each direction. This reduces the portion of the clock period available in the FPGA slowing the chip, making it harder to 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 type and amount of logic used for that daughter card. The individual Springbok mappings are still free to choose the type and number of custom daughter cards to use in any particular system, and the mapping can still grow to any size.

Note that 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 that could help ease routing in hotspots of a given mapping. Also, special connections could be built into the baseplates that 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 currently, and requires further study.

Status

We are currently well into the development of the Springbok system. Work on Springbok's mesh routing topology has been completed, yielding topologies with higher bandwidth, lower delay, and reduced I/O pin usage [Hauck94a]. With this we have gained an understanding of how the Springbok baseplates and standard daughter cards should be constructed, though we have not yet fabricated the actual hardware. We have also examined how the external interfaces of a prototype can be supported [Hauck95c].

On the software end of the Springbok system, we have completed work on an efficient bipartitioning algorithm [Hauck95b], as well as methods for recursively applying bipartitioning to an arbitrary topology [Hauck95c]. We have also developed pin assignment software [Hauck94b, Hauck94c], which handles part of the global routing step.

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, decreases cost steps, and more easily handles the functionality necessary to interface to timing-inflexible components.

References

- [Aptix93] Aptix Co, *Data Book*, San Jose, CA, 1993.
- [Babb93] J. Babb, R. Tessier, A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 142-151, 1993.
- [Hauck94a] S. Hauck, G. Borriello, C. Ebeling, "Mesh Routing Topologies for Multi-FPGA Systems", *International Conference on Computer Design*, pp. 170-177, 1994.
- [Hauck94b] S. Hauck, G. Borriello, "Pin Assignment for Multi-FPGA Systems (Extended Abstract)", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 11-13, 1994.
- [Hauck94c] S. Hauck, G. Borriello, "Pin Assignment for Multi-FPGA Systems", *University of Washington, Dept. of Computer Science & Engineering Technical Report #94-04-01*, 1994.

- [Hauck94d] S. Hauck, G. Borriello, C. Ebeling, "Springbok: A Rapid-Prototyping System for Board-Level Designs", *ACM/SIGDA 2nd International Workshop on Field-Programmable Gate Arrays*, 1994.
- [Hauck95a] S. Hauck, G. Borriello, C. Ebeling, "Achieving High-Latency, Low-Bandwidth Communication: Logic Emulation Interfaces", *University of Washington, Dept. of Computer Science & Engineering Technical Report #95-04-04*, 1995.
- [Hauck95b] S. Hauck, G. Borriello, "An Evaluation of Bipartitioning Techniques", Chapel Hill Conference on Advanced Research in VLSI, pp. 383-402, 1995.
- [Hauck95c] S. Hauck, G. Borriello, "Logic Partition Orderings for Multi-FPGA Systems", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 32-38, 1995.
- [Katz93] R. Katz, P. Chen, A. Drapeau, E. Lee, K. Lutz, E. Miller, S. Seshan, D. Patterson, "RAID-II: Design and Implementation of a Large Scale Disk Array Controller", *Research on Integrated Systems: Proceedings of the 1993 Symposium*, pp.23-37, 1993.
- [Koch94] G. Koch, U. Keschull, W. Rosenstiel, "A Prototyping Environment for Hardware/Software Codesign in the COBRA Project", *Third International Workshop on Hardware/Software Codesign*, 1994.
- [McMurchie95] L. E. McMurchie, C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 111-117, 1995.
- [Seitz90] C. L. Seitz, "Let's Route Packets Instead of Wires", *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pp. 133-138, 1990.
- [Varghese93] J. Varghese, M. Butts, J. Batcheller, "An Efficient Logic Emulation System", *IEEE Transactions on VLSI Systems*, Vol. 1, No. 2, pp. 171-174, 1993.