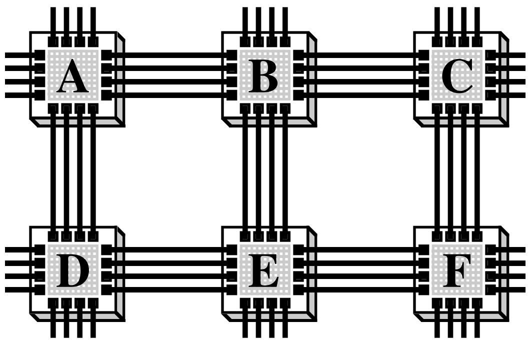
Chapter 5. Multi-FPGA System Hardware

In Chapter 3 we discussed the applications of multi-FPGA systems, and concentrated on logic emulation in Chapter 4. In this chapter we will explore some of the existing multi-FPGA systems themselves. There are a large number of systems that have been constructed, for many different purposes, with a wide range of structures. Note that this section is intended to illustrate only the types of systems possible, and is not meant to be an in-depth discussion of all the details of existing systems. Thus, some details of the topologies, as well as the amount of connectivity on links in the systems, have been omitted.



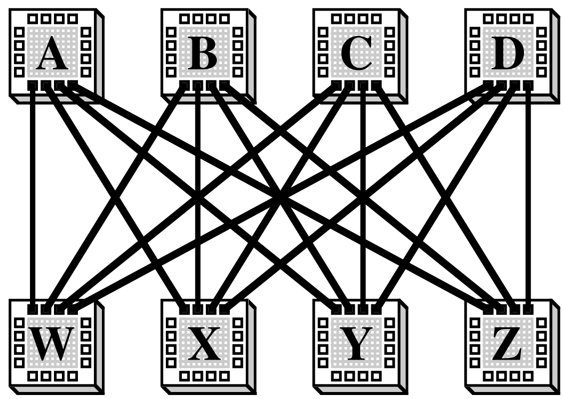


Figure 27. Mesh (left) and crossbar (right) topologies. In the crossbar, chips **A**-**D** are routing-only, while **W**-**Z** hold all the logic in the system.

The most important difference between multi-FPGA systems is in the topology chosen to interconnect the chips in the system. The most common topologies are mesh and crossbar (or bipartite graph) topologies. In a mesh, the chips in the system are connected in a nearest-neighbor pattern (Figure 27 left). These topologies have the advantage of simplicity, because of the purely local interconnection pattern, as well as easy expandability, since meshes can be grown by adding resources to the edge of the array. Numerous 2D mesh-based systems have been built [Kean92, Shaw93, Bergmann94, Blickle94, Tessier94, Yamada94], as well as 3D meshes [Sample92, Quénot94]. Linear arrays, which are essentially 1-dimensional meshes, have also been built [Gokhale90, Raimbault93, Monaghan94]. More details on mesh topologies can be found in Chapter 7.

Crossbar topologies separate the chips in the system into logic-bearing and routing-only (Figure 27 right). The logic-bearing FPGAs contain all the logic in the system, while the routing-only chips are used purely for inter-FPGA routing. Routing-only chips are connected only to logic-bearing FPGAs, and (usually) have exactly the same number of connections to all logic-bearing FPGAs. Logic-bearing FPGAs are connected only to routing-only FPGAs. The idea behind this topology is that to route between any set of FPGAs requires routing through only one extra chip, and that chip is any one of the routing-only chips. Because of the symmetry of the system, all routing-only chips can handle this role equally well. This gives much more predictable performance, since regardless of the locations of the source and destinations, the delay is the same. In a topology like a mesh, where it might be necessary to route through several intermediate chips, there is a high variance in the delay of inter-FPGA routes. There are two negative features of this topology. First, crossbar topologies are not expandable, since all routing-only chips need to connect to all logic-bearing FPGAs, and thus the system is constrained to a specific size once the connections to any specific routing-only chip are determined. Second, the topology potentially wastes resources, since the routing-only chips are used purely for routing, while a mesh can use all of its chips for logic and routing. However, since the bottleneck in multi-FPGA systems is the inter-chip routing, this waste of resources may be more than made up for by greater logic utilization in the logic-bearing chips. Also, some of the cost of the wasted resources can be avoided by using less expensive devices for the routing-only chips. Possibilities include FPICs, crossbars, or cheaper FPGAs (either because of older technology or lower logic capacity). Several pure crossbar topologies have been constructed [Chan92, Ferrucci94, Kadi94, Weiss94].

A middle-ground between the two topologies, which combines the expandability of meshes and the simpler routing of crossbars, is hierarchical crossbars [Varghese93]. As shown in Figure 28, crossbars can be stacked together hierarchically, building up multiple levels of routing chips. There are two simple crossbars in the system, one consisting of routing-only chips **E-H** and logic-bearing FPGAs **M-P**, and a second one consisting of routing-only chips **I-L** and logic-bearing FPGAs **Q-T**. Routing chips **E-L** will be called the first-level crossbars, since they connect directly to the logic-bearing FPGAs. To build the hierarchy of crossbars, the simple crossbars in the system can be thought of as logic-bearing chips in an even larger crossbar. That is, a new crossbar is built with routing-only chips and logic-bearing elements, but in this crossbar the logic-bearing elements are complete, simple crossbars. Note that the connections within this higher-level crossbar go to the routing-only chips in the simple crossbars, so first-level and second-level routing-only chips are connected together. This hierarchy can be continued, building up other crossbars with third-level and higher routing-only chips. In an N-level hierarchical crossbar, the chips are arranged as above, with routing-only chips at the *Ith* level connected to chips at the *(I+1)th* and *(I-1)th* level, where the *0th* level is the logic-bearing chips. Note that in contrast to the simple crossbar topology, in a hierarchical crossbar the logic-bearing FPGAs are not connected to all the routing-only chips (even those at the first-level). Full connectivity occurs at the top (*Nth*) level, where all *Nth*-level routing chips are connected to all *(N-1)th* level routing chips.

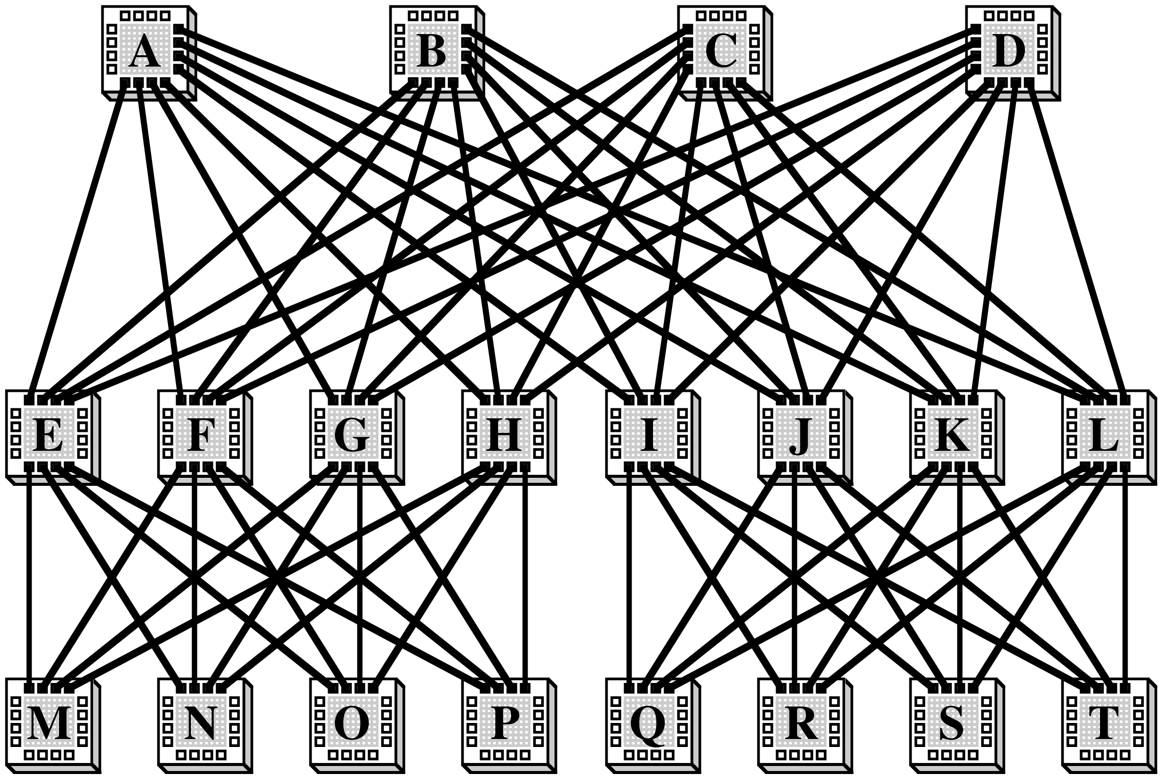


Figure 28. A hierarchy of crossbars. FPGAs **M-T** hold all the logic in the system. Chips **E-H** and **I-J** form two first-level crossbars, and chips **A-D** form a second-level crossbar.

Routing between two logic-bearing FPGAs in the system simply requires determining the level at which the source and destination share an ancestor, and then routing from the source up to one of these shared ancestors, and back down to the destination. The routing from the source to the shared ancestor requires routing through exactly one routing-only chip in the intervening levels, as does the routing from the ancestor to the destination. Because of the symmetry of the topology (and ignoring conflicts from other routes), any of the ancestors of the source (for the route up) or destination (for the route down) at a given level can be used to handle the routing, regardless of what other chips are part of the route.

As mentioned earlier, the advantage of a hierarchical crossbar topology is that it has much of the expandability of a mesh, yet has much of the simplified routing of a crossbar. Since levels of hierarchy can be added to a hierarchical crossbar, it can easily grow larger to handle bigger circuits. Levels of the hierarchy tend to map onto components of the system, such as having a complete first-level crossbar on a single board, a complete second-level crossbar contained in a cabinet, and a complete third-level crossbar formed by interconnecting cabinets [Butts91]. The routing simplicity, as shown above, demonstrates a large amount of flexibility in the routing paths, and a large amount of symmetry in the system. How easy it is to route between logic-bearing FPGAs is simple to determine, since if two logic-bearing chips are within the same first-level crossbar, then they are as easy to route between as any other pair of logic-bearing chips within that crossbar. Thus, when mapping onto the topology, the system tries to keep most of the communication between chips in the same first-level crossbar, and most of the rest of the communication between chips in the same second-level crossbar, and so on.

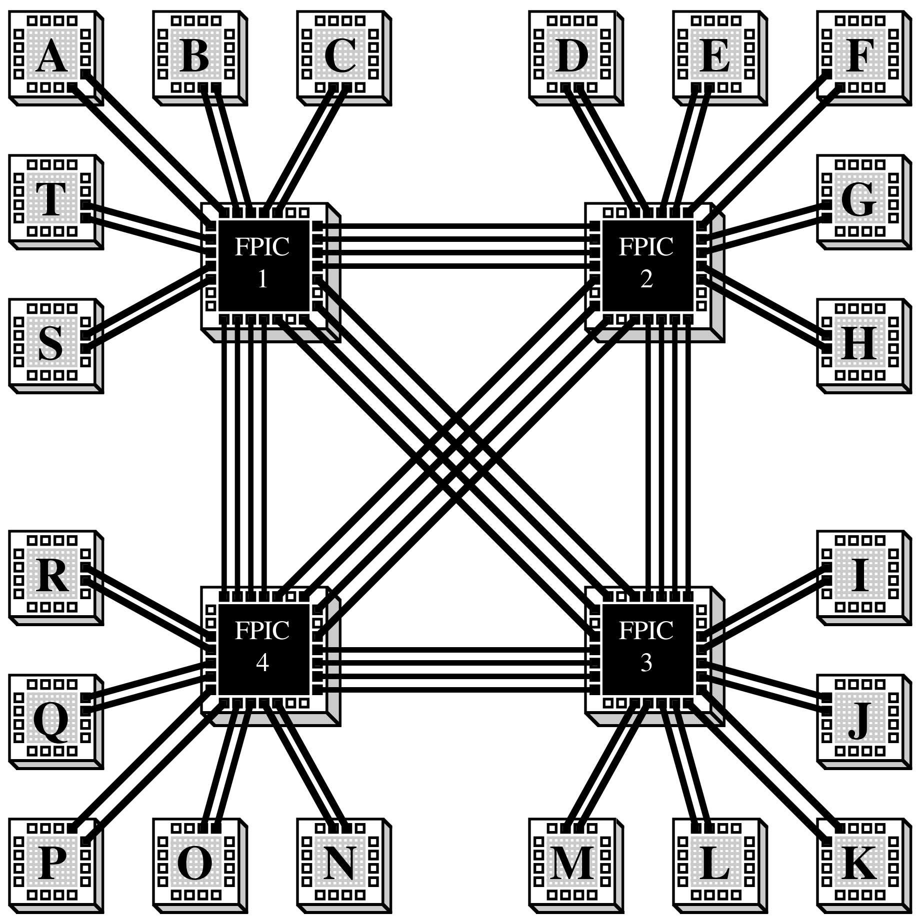


Figure 29. The Aptix AXB-AP4 topology [Aptix93b]. Logic-bearing FPGAs are connected only to routing-only FPICs, but the FPICs connect to both FPGAs and other FPICs.

There are two downsides to this topology. First, signals may have to go through many more routing-only chips than in a simple crossbar, since they could potentially have to go all the way up to the top level of the hierarchy to make a connection. However, the maximum routing distance is less than in a mesh, since the length (in chips routed through) of the maximum route in a mesh grows by $\sqrt{N}$ (where N is the number of logic-bearing FPGAs in the system), while the length in a hierarchical crossbar grows by *log(N)*. The other problem is that the hierarchical crossbar topology requires a large amount of resources to implement the routing-only chips in the system. If one could instead use the routing-only chips as logic-bearing chips, the capacity of the system might be greatly increased. However, if this extra logic capacity cannot efficiently be used, it will be of little value.

Some systems use a two-level topology, which is somewhat similar to the pure crossbar topologies. Just as in the crossbar topologies, FPGAs in the system are connected only to routing-only chips. However, unlike the pure crossbar, the routing-only chips in the system are connected to both logic-bearing and routing-only chips. That is, there is a topology of connections between the routing-only chips in the system, and the FPGAs connect to these chips. Thus, these systems are similar to multiprocessors, in that for two processing elements to communicate (the FPGAs in a two-level topology), they must send signals through a router connected to the source (an FPIC or crossbar). The signal then travels through intervening routers until it reaches the router connected to the destination, at which point it is sent to that processor (an FPGA in our case). An example of such a system is the Aptix AXB-AP4 [Aptix93b]. As shown in Figure 29, five FPGAs are connected to each FPIC, and all the FPICs are connected together. In this system, the longest route requires moving through 2 FPICs, and no intermediate FPGAs are used for routing. If instead a mesh was built out of the 20 FPGAs in this system, it potentially could require routing through many FPGAs to reach the proper destination, increasing the delay, and using up valuable FPGA I/Os. However, whether the routing flexibility of the two-level topology justifies the substantial cost of the FPICs is unclear. Other two-level systems have been constructed with a variety of topologies between the routing-only chips [Adams93, Galloway94, Njølstad94].

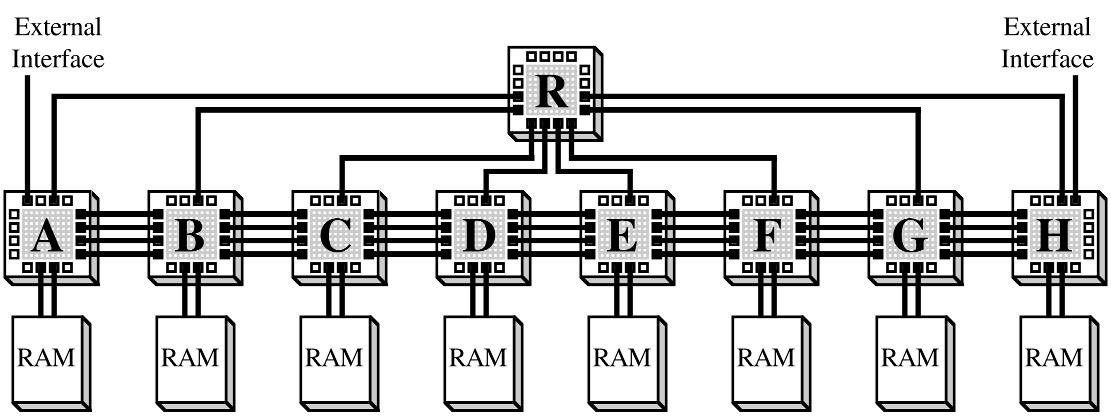


Figure 30. The Splash 2 topology [Arnold92]. The linear array of FPGAs (**A-H**) is augmented by a routing-only crossbar (**R**). Note that the real topology has 16 FPGAs in the linear array.

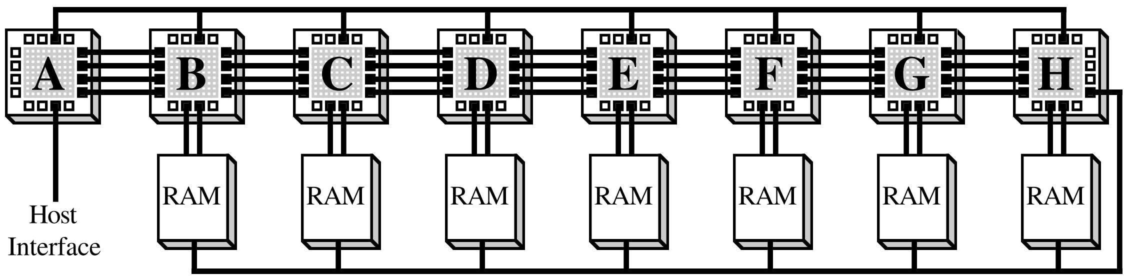


Figure 31. The Anyboard topology [Thomae91]. The linear array of FPGAs is augmented with a global bus.

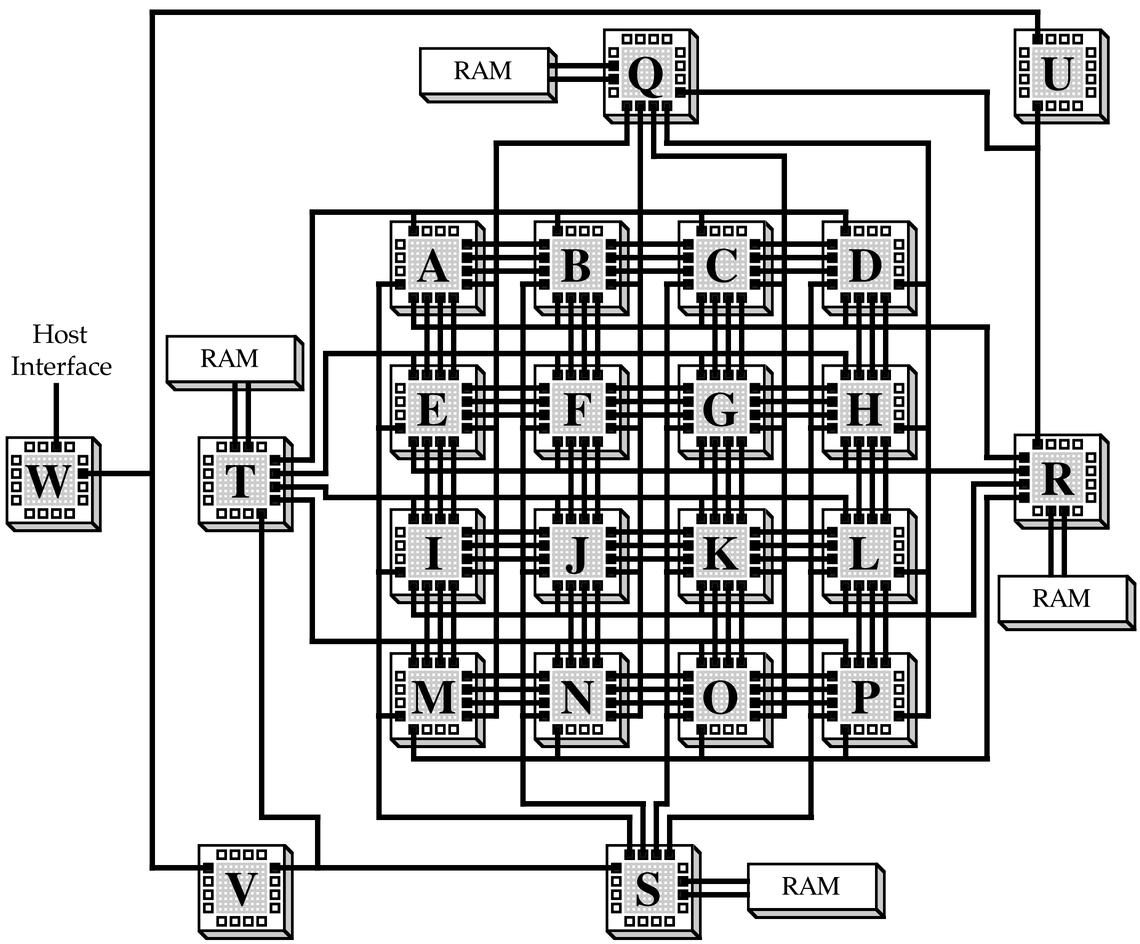


Figure 32. The DECPeRLe-1 topology [Vuillemin95]. The central 4x4 mesh (**A-P**) is augmented with global buses to four support FPGAs (**Q-T**), which feed to three other FPGAs (**U-W**).

Some systems are more of a hybrid between different topologies than a single topology. One example of this is the Splash 2 architecture [Arnold92]. The Splash 2 machine takes the linear array of its predecessor Splash [Gokhale90], and augments it with a crossbar interconnecting the FPGAs (Figure 30). In this way the system can still efficiently handle the systolic circuits that Splash was built to support, since it retains the nearest-neighbor interconnect, but the crossbar supports more general communication patterns, either in support of or as replacement to the direct interconnections. There are several other augmented linear arrays [Benner94, Box94, Darnauer94, Carrera95]; one example is the Anyboard [Thomae91], which has a linear array of FPGAs augmented by global buses (Figure 31). Another hybrid topology is the DECPeRLe-1 board [Vuillemin95], which has a 4x4 mesh of FPGAs augmented with shared global buses going to four support FPGAs (Figure 32). These are then connected to three other FPGAs that handle global routing, connections to memory, and the host interface. The DECPeRLe-0 board is similar [Bertin93].

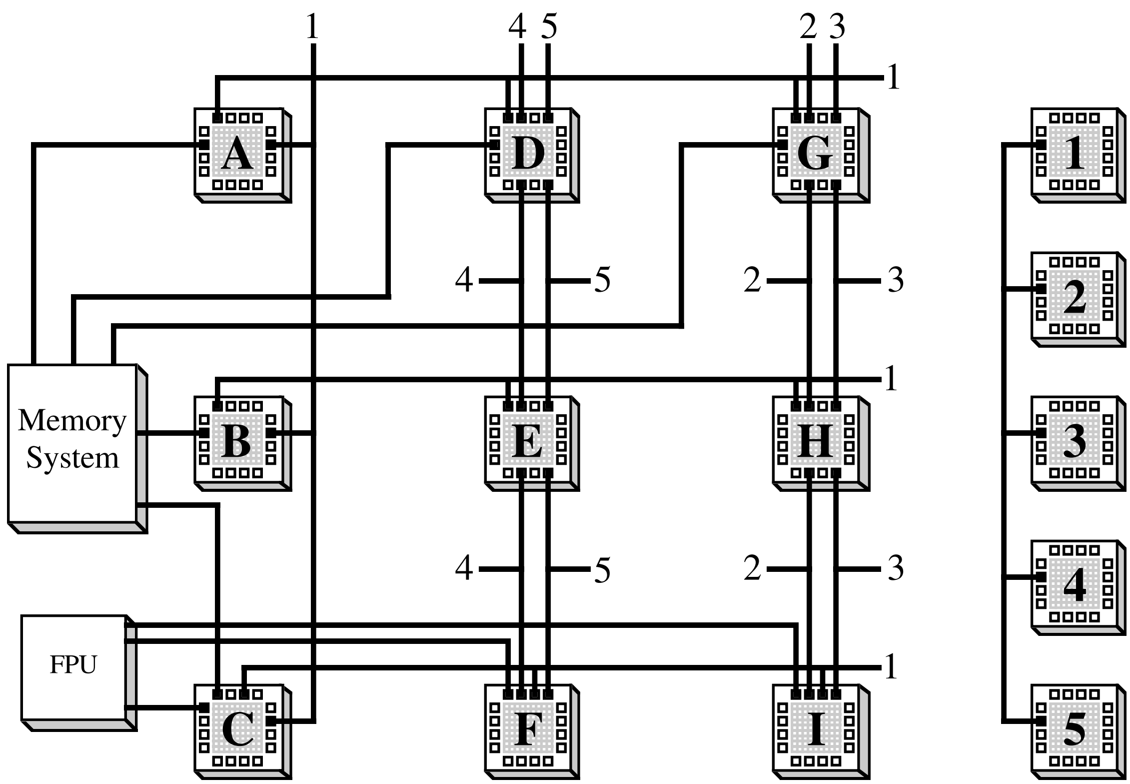


Figure 33. The Marc-1 topology [Lewis93]. The complete topology consists of two copies of the left subsystem (**A**-**I**, Memory & FPU), and one copy of the right (**1**-**5**). Numbers by themselves indicate connections to the FPGAs at right.

Some topologies are unique [Engels91, Cox92, Erdogan92, Casselman93, Herpel93, Lewis93, Wazlowski93, Howard94a, Nguyen94, Saluvere94, Hayashi95, Högl95, Van den Bout95]. These are often machines primarily built for a specific application, and their topology is optimized for the features of that application domain. For example, the Marc-1 (Figure 33) [Lewis93] is a pair of 3x3 meshes of FPGAs (**A**-**I**), where most of the mesh connections link up to a set of FPGAs intended to be used as crossbars (**1**-**5**). While the vertical links are nearest-neighbor, the horizontal links are actually buses. There is a complex memory system attached to some of the FPGAs in the system, as well as a floating point unit. This machine architecture was constructed for a specific application - circuit simulation (and other algorithms) where the program to be executed can be optimized on a per-run basis for values constant within that run, but which may vary from dataset to dataset. For example, during circuit simulation, the structure of the gates in the circuit are fixed once the program begins executing, but can be different for each run of the simulator. Thus, if the simulator can be custom compiled on a per-run basis for the structure of the circuit being simulated, there is the potential for significant speedups. Note that this is different than logic emulation, since a special-purpose processor is mapped onto the Marc-1 system, and this processor mapping simulates the circuit, while a logic emulator directly maps the logic onto the multi-FPGA system. Because of the restricted domain of compiled-code execution, the topology was built to contain a special-purpose processor, with the instruction unit in FPGAs **A**-**C**, and the datapath in the FPGAs **D**-**I**.

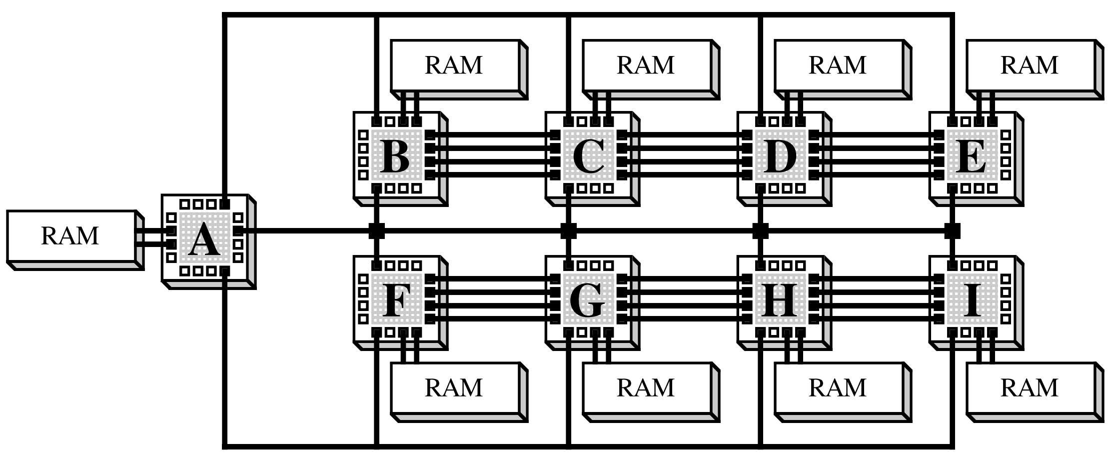


Figure 34. The RM-nc system [Erdogan92]. The linear arrays can be grown larger than the system shown.

Another example of a system optimized for a specific task is the RM-nc system [Erdogan92]. As shown in Figure 34, the system has a controller chip **A**, and two linear arrays of FPGAs **B-E** and **F-I**. Each FPGA has a local memory for buffering data. The system contains three major buses, with the outside buses going to only one linear array each, while the center bus is shared by all FPGAs in the system. This system is optimized for neural network simulation, with the controller chip handling global control and sequencing, while the FPGAs in the linear array handle the individual neuron computations. A similar system for neural-network simulation, with buses going to all FPGAs in the system, and with no direct connections between neighbors, has also been built [Eldredge94].

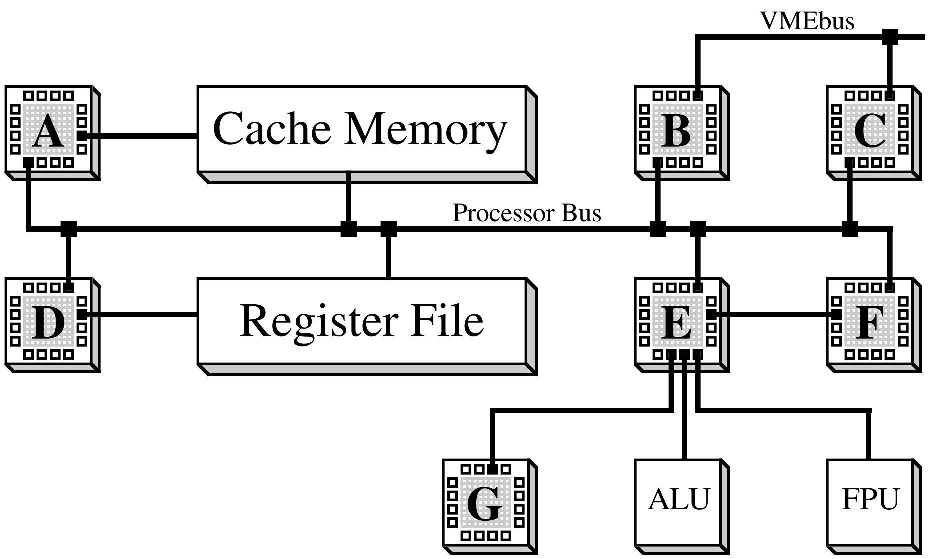


Figure 35. System similar to the Mushroom processor prototyping system [Williams91].

One of the most common domains for the development of a custom multi-FPGA system is the prototyping of computers. A good example of this is the Mushroom system [Williams91], which is a system of FPGAs, memories, and computation chips meant to be used for processor prototyping (the exact interconnection pattern is unavailable, but an approximate version is shown in Figure 35). Seven FPGAs are included in the system, and each of these has a specific role. FPGA **A** is a cache controller, **B** & **C** serve as a VMEbus interface, **D** handles register accesses, **E** is for processor control, **F** performs instruction fetches, and **G** is for tag manipulation and checking. Memories are included for caches and the register file, while an ALU and FPU chip are present for arithmetic operations. The advantage of this system is that the FPGAs can be reconfigured to implement different functions, allowing different processor structures to be explored. The arithmetic functions and memory, features that are inefficient to implement in FPGAs and which are standard across most processors, are implemented efficiently in chips designed specifically for these applications. Other systems have adopted a similar approach, including another system intended for developing application-specific processors [Wolfe88], a workstation design with FPGAs for I/O functions and for coprocessor development [Heeb93], and a multiprocessor system with FPGA-based cache controllers for exploring different multiprocessor systems and caching policies [Öner95]. A somewhat related system is the CM-2X [Cuccaro93], a standard CM-2 supercomputer with the floating-point unit replaced with a Xilinx FPGA. This system allows custom coprocessors to be built on a per-algorithm basis, yielding significant performance increases for some non-floating-point intensive programs.

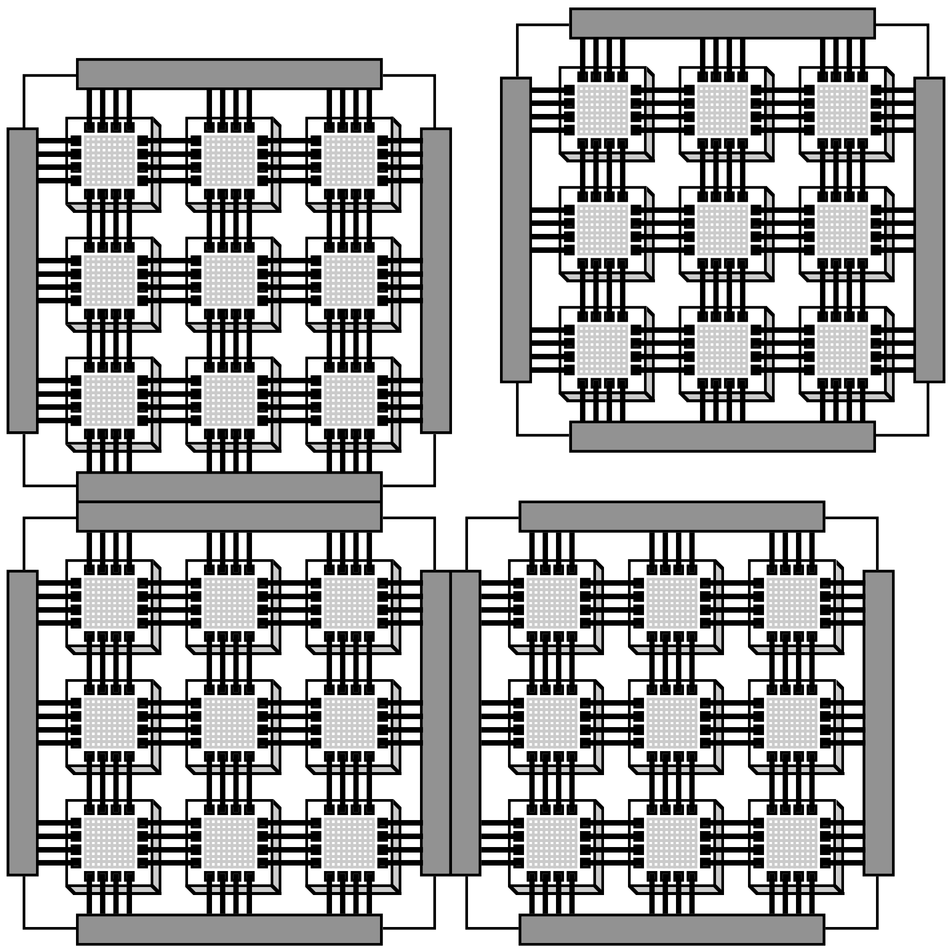


Figure 36. Expandable mesh topology similar to the Virtual Wires Emulation System [Tessier94]. Individual boards are built with edge connectors and a small amount of logic, and can be interconnected to form a larger mesh.

There are interesting features of multi-FPGA systems beyond just the interconnect topology. One of these is the ability to increase the size of the multi-FPGA system. As described previously, systems based on hierarchical crossbars [Varghese93] can grow in size by adding extra levels of hierarchy. Other systems provide similar expandability, with special interconnection patterns for multi-board systems [Amerson95, Högl95]. In the case of meshes and linear arrays, the systems can be built of a basic tile with external connectors and a small amount of on-board logic (Figure 36), and the system can be expanded by connecting together several of these boards [Thomae91, Arnold92, Shaw93, Tessier94, Drayer95]. The GERM system [Dollas94] is similar, except that the four external connectors are built to accommodate ribbon cables. In this way, fairly arbitrary topologies can be created.

Multi-FPGA systems are often composed of several different types of chips. The majority of systems are built from Xilinx 3000 or 4000 series FPGAs [Xilinx94], though most commercial FPGAs have been included in at least one multi-FPGA system. There are some system designers that have chosen to avoid commercial FPGAs, and develop their own chips. Some have optimized their FPGAs for specific applications, such as general custom-computing and logic emulation [Amerson95], emulation of telecommunication circuits [Hayashi95], or image processing [Quénot94]. Another system uses FPGAs optimized for inter-chip communication on an MCM substrate [Dobbelaere92], since MCMs will be used to fabricate the system to achieve higher density circuits. A final example is the Pegasus system [Maliniak94], which uses a hybrid processor/FPGA chip with multiple configurations. By rapidly switching configurations the hardware can be time-division multiplexed. In this way, the hardware is reused and the communication pipelined, hopefully yielding higher density mappings.

Many multi-FPGA systems include non-FPGA chips. By far the most common element to be included is memory chips. These chips are usually connected to the FPGAs, and are used as temporary storage for results, as well as general-purpose memories for circuit emulation. Other systems have included integer and/or floating point ALUs [Wolfe88, Williams91, Lewis93, Benner94], DSPs [Engels91, Bergmann94, vom Bögel94, Zycad94a], and general-purpose processors [Shaw93, Raimbault93, Benner94, Koch94, vom Bögel94, Zycad94a] to handle portions of computations where dedicated chips perform better than FPGA solutions. Another common inclusion into a multi-FPGA system is crossbars or FPICs (Chapter 2). For example, a multi-FPGA system with a crossbar or hierarchical crossbar topology requires chips purely for routing. An FPIC or crossbar chip can handle these roles. If the FPIC or crossbar has a lower unit cost, is capable of higher performance or higher complexity routing, or has a larger number of I/O pins, then it can implement the routing functions better than a purely FPGA-based solution. There have been several systems that have used crossbar chips or FPICs in crossbar [Kadi94, Weiss94] and hierarchical crossbar topologies [Varghese93], as well as hybrid crossbar/linear array topologies [Arnold92, Darnauer94, Carrera95] and other systems [Adams93, Aptix93b, Casselman93, Galloway94, Njølstad94, Högl95, Van den Bout95].

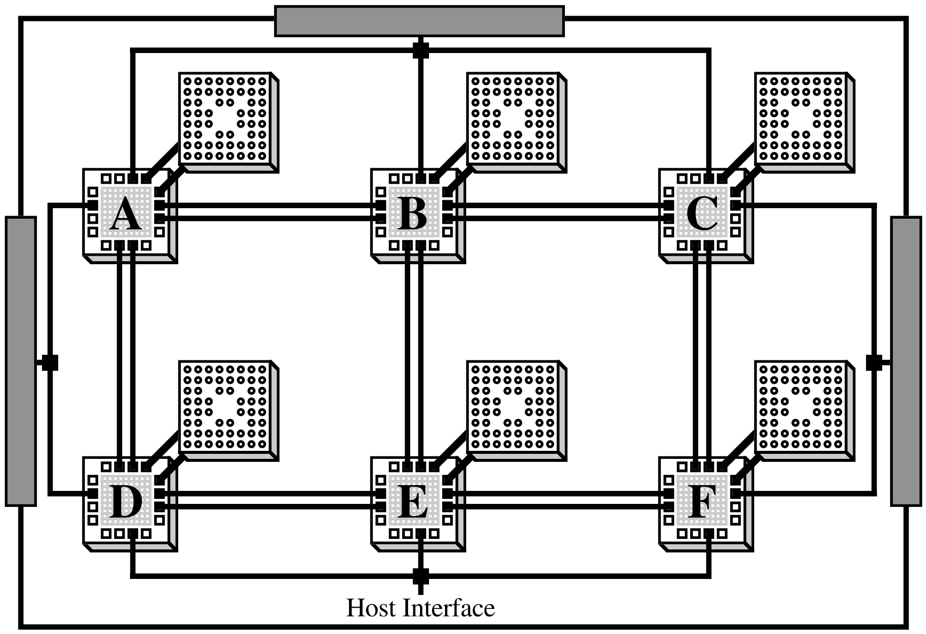


Figure 37. The MORRPH topology [Drayer95]. Sockets next to the FPGAs allow arbitrary devices to be added to the array. Buses connected to external connectors allow multiple boards to be hooked together to build an arbitrarily large system.

While adding fixed, non-FPGA resources into a multi-FPGA topology may improve quality for some types of mappings, it is possible to generate a more general-purpose solution by allowing the inclusion of arbitrary devices into the array. For example, in the MORRPH topology [Drayer95] sockets are placed next to the FPGAs so that arbitrary devices can be inserted (Figure 37). Thus, in a mapping that requires extra memory resources, memories can be plugged into the array. In other circumstances, DSPs or other fixed-function chips can be inserted to perform complex computations. In this way, the user has the flexibility to customize the array on a per-mapping basis. Other systems have adopted a similar strategy [Butts91, Sample92, Aptix93b], with a limited ability to insert arbitrary chips into the multi-FPGA system.

Some systems are more of an infrastructure for bringing to bear the best mix of resources rather than a specific, fixed multi-FPGA system. One example of this is the G800 system [Giga95]. The board contains two FPGAs, some external interface circuitry, and four locations to plug in compute modules (Figure 38). Compute modules are cards that can be added to the system to add computation resources, and can be stacked four deep on the board. Thus, with four locations that can be stacked four deep, a total of 16 compute boards can be combined into a single system. These compute boards are connected together, and to the FPGAs on the base board, by a set of global buses. Compute modules can contain an arbitrary mix of resources. Examples include the X210MOD-00, which has two medium Xilinx FPGAs, and the X213MOD-82, which has two large FPGAs, 8MB of DRAM, and 256 KB of SRAM. The users of the system are free to combine whatever set of modules they desire, yielding a system capacity ranging from only two medium FPGAs (a single X210MOD-00), to 32 large FPGAs (16 X213MOD-82s) and a significant amount of RAM. Similar systems include DEEP [vom Bögel94] and Paradigm RP [Zycad94a]. The Paradigm RP has locations to plug in up to eight boards. These boards can include FPGAs, memories, DSPs, and standard processors. In the G800 and Paradigm RP systems, cards with new functionality or different types of chips can easily be accommodated.

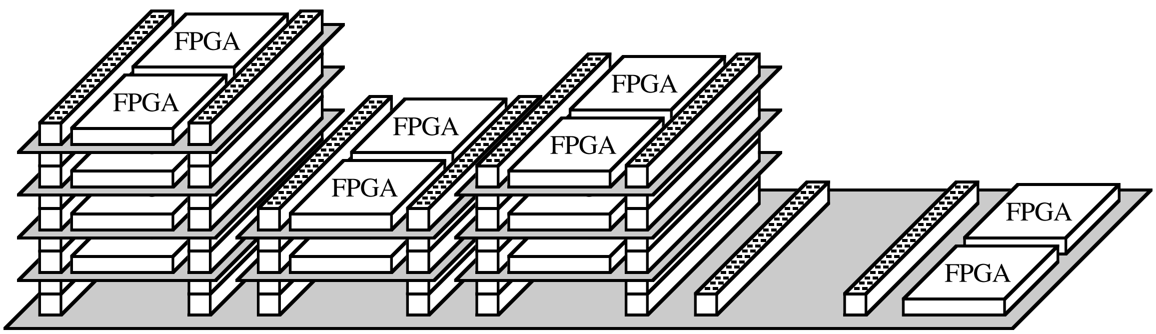


Figure 38. The G800 board [Giga95]. The base board has two FPGAs and four sockets. The socket at left holds four computing module boards (the maximum allowed in a socket), while the socket at right has none.

Even more flexible systems are possible. One example is the COBRA system [Koch94]. As shown in Figure 39, the system is made up of several types of modules. The standard *base module* has four FPGAs, each attached to an external connection at one of the board's edges. Boards can be attached together to built a larger system, expanding out in a 2D mesh. Other module types can easily be included, such as modules bearing only RAM, or a host interface, or a standard processor. These modules attach together the same way as the base module, and will have one to four connectors. One somewhat different type of module is a bus module. This module stacks other modules vertically, connecting them together by a bus.

Systems like COBRA allow an arbitrary resource mix to be brought to bear on a problem. Thus, a custom processing system can quickly be built for a given task, with FPGAs handling the general logic, while standard processors handle complex computations and control flow. Thus, the proper resource is always used for the required functionality. In Chapter 6 we will present another such system, which is optimized for the rapid-prototyping of multi-chip circuits. It offers the expandability of COBRA, with MORRPH's ability to include arbitrary devices, and was originally proposed before either of these architectures.

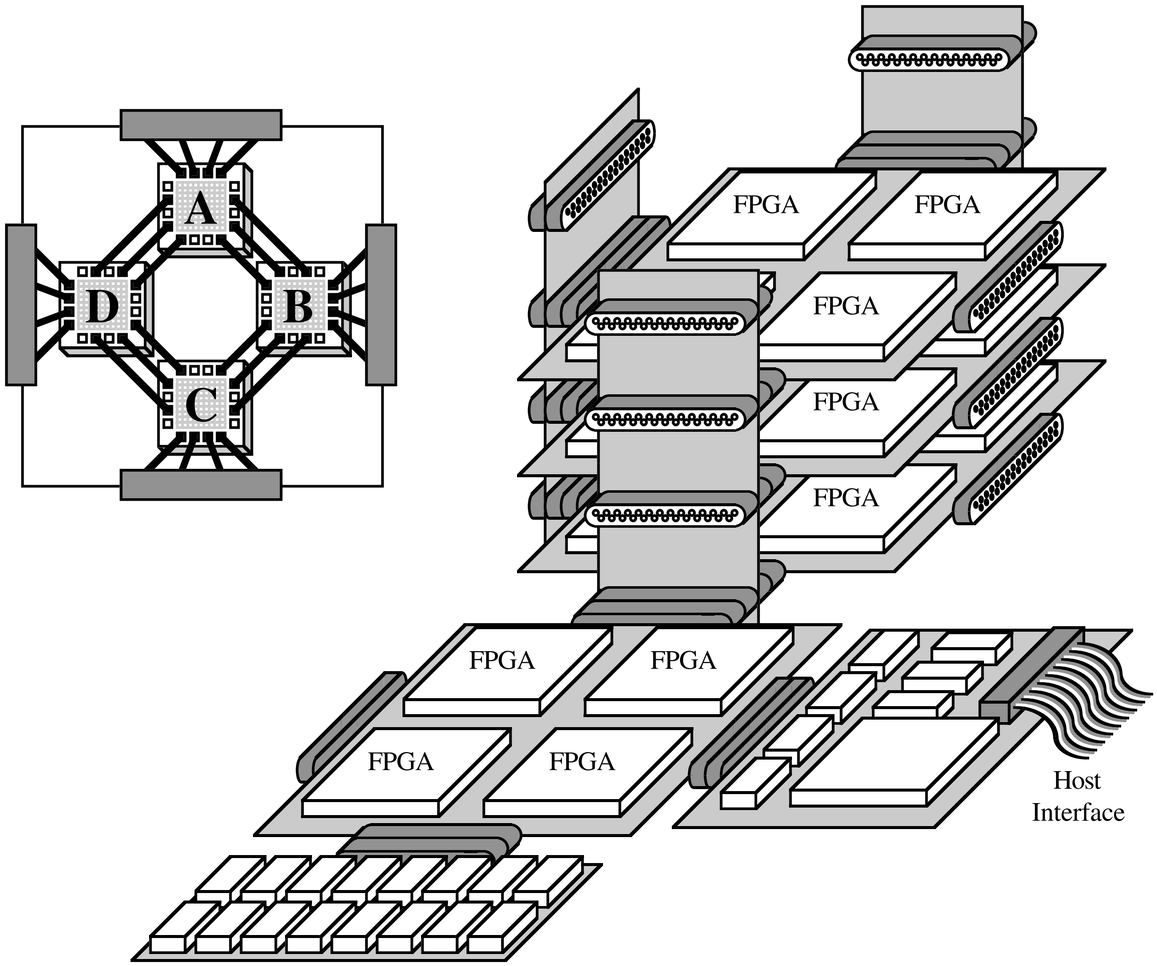


Figure 39. Base module (upper left) and example topology built in the COBRA system [Koch94]. The mapping includes a tower of three base modules surrounded by three bus modules (top), a base module (center), a RAM module (bottom), and an I/O module (right).

In the following chapters we will delve into several issues in multi-FPGA system hardware, including multi-chip system prototyping in Chapter 6, routing topologies for multi-FPGA systems in Chapter 7, and interface support for logic emulators in Chapter 8. Chapter 9 begins the discussion of software support for multi-FPGA systems.