CHAPTER 30

Multi-FPGA Systems: Logic Emulation

Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst

Application specific integrated circuit (ASIC) verification has been an important and commercially successful application of field-programmable gate arrays (FPGAs) for over a decade. By mapping the logic of a new chip design onto a system of FPGAs, logic emulation systems provide a high-speed simulation of the design under development. As FPGA technology has matured and FPGA logic capacity has grown, the use of FPGAs for functional logic emulation has increased. Contemporary emulation systems often include a sizable number of FPGA and memory devices organized in topologies that allow for efficient logic evaluation and inter-FPGA communication.

Although the hardware architecture of an emulator plays an important role in defining its effectiveness, system usability is often most closely tied to an emulator's compilation environment. To successfully map a complete ASIC design to an emulation system, emulators require optimized compilation steps that effectively distribute design logic across available FPGA resources and coordinate intra-FPGA computation and inter-FPGA communication.

To illustrate contemporary approaches to FPGA-based logic emulation, we profile here the hardware and software systems of a commercial FPGA-based emulator. We show that, although off-the-shelf FPGAs have been used effectively in a number of commercial logic emulators, several issues related to FPGA compile time, design debugging, and emulator host interfacing must be addressed to maintain their commercial viability.

30.1 BACKGROUND

Research in reconfigurable computing has been active for well over a decade, but the widespread commercial use of FPGAs as computing devices has been limited. A notable commercial success story for reconfigurable computing has been the use of FPGAs in ASIC logic verification. Over the past decade, the number of transistors that can be integrated into application-specific devices has grown exponentially with Moore's Law, leading to an increased need to verify design functionality prior to device fabrication. Currently, it is estimated that 60

to 80 percent of ASIC design time is spent performing verification [29], primarily because of the high nonrecurring engineering (NRE) cost associated with ASIC fabrication. The flexibility, parallelism, and reprogrammability of FPGAs make them an ideal platform for verifying, prior to fabrication, the functionality of ASIC designs. The availability of automatic FPGA mapping tools, such as those described in Chapters 13, 14, and 17, have streamlined the design conversion process, making the path from ASIC design to FPGA implementation more straightforward.

FPGA-based logic verification is often used to augment or replace microprocessor-based simulation of register transfer level (RTL) or gate-level designs. The primary source of emulation speed improvement versus simulation is the parallel implementation of circuit logic in the FPGA. While the amount of logic evaluated per clock cycle in a microprocessor-based simulator is constrained by a limited number of ALUs (typically four or five at most), the number of per-cycle FPGA operations per emulation system is constrained only by the available amount of total FPGA resources. This increase in logic evaluation capacity comes at a cost. Unlike its simulation counterparts, FPGA-based emulation can provide only functional verification for designs. Because the fundamental technology used to implement the emulated logic differs from the source ASIC technology, postlayout timing information cannot be replicated. As a result, FPGA-based emulators support only cycle-accurate logic evaluation that is synchronized to design clock edges of the emulated design. Additionally, circuit debugging for emulation systems is often more complex than debugging with simulators. The sequential nature of simulation-based verification facilitates debugging and logic tracing. Logic analysis in a parallel verification environment requires the use of specialized hardware resources and debugging tools.

FPGA-based emulators take on a variety of forms, ranging from single-device systems to commercial emulation systems that include hundreds of devices. Although specific system implementations vary, most FPGA-based logic emulators contain a tightly connected collection of FPGA devices. These systems can be distinguished by their component FPGA and memory devices, interconnection topology, design-mapping software, and external interfaces. The system topology defines the positions of FPGAs and inter-FPGA communication resources. The need for multiple devices to emulate many ASIC designs is due to the cost of FPGA reconfigurability. Because the silicon area overhead of FPGA versus ASIC technology has been measured to be about 40x [15], FPGA programming technology requires that an ASIC logic design be partitioned across multiple FPGA devices to achieve the necessary device logic capacity.

For most emulators, there is a strong association between the physical architecture of the FPGA system and the compiler used to map user designs to the emulator. Like the intra-FPGA mapping flow outlined in Chapters 13, 14, and 17, emulation mapping for multi-FPGA emulators requires a series of complex and interrelated algorithms. As we will see later in this section, emulation system compilation is complicated by the variety of design features in contemporary ASICs. These features include multiple asynchronous clock domains, multiported memories, and testing and debugging interfaces, which are playing

an increasingly important role. In assessing modern emulation, the interfaces between emulators, simulators, logic analyzers, and prototype systems must be considered. It will be shown that, in the future of FPGA-based logic emulation, both design compilation and testing interfaces will play a critical role.

To illustrate the complexity of contemporary FPGA-based emulation, the hardware, compilation, and testing components of a VirtuaLogic VLE-2M emulation system from Mentor Graphics [21] will be profiled. This commercially successful system demonstrates not only the benefits of FPGA-based emulation, but also some of its limitations.

30.2 USES OF LOGIC EMULATION SYSTEMS

Logic emulation systems are typically used in one of two verification scenarios: (1) as a physical replacement for an ASIC in a target system, or (2) as a simulation accelerator. The ASIC replacement approach requires the use of a physical connection between the emulator and the target system. As shown in Figure 30.1, one end of the connection typically plugs into connectors on the emulation system that are interfaced to selected FPGA I/O pins. The other end of the connection plugs into the location on the target system that would normally hold the package of the emulated device. This emulation pod typically has the same pin configuration as the emulated device package. The use of in-circuit emulation allows for complete target system verification, including the emulated design and surrounding interfaces and peripherals. Although many times the target system is forced to operate at clock speeds of 0.5 to 5 MHz, a substantial amount of system functionality can generally be evaluated via in-circuit emulation. An attached logic analyzer is often used to probe specific design signals.

An alternative to in-circuit emulation is coverification (sometimes called cosimulation). In this mode of operation, the logic emulator works in concert with a host workstation to verify an emulated design without the use of

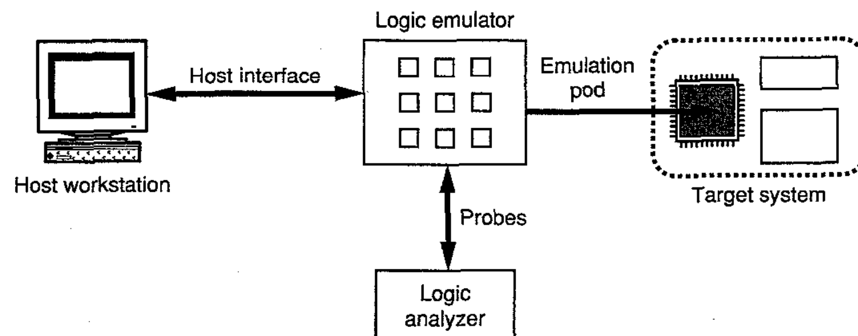


FIGURE 30.1 ■ A typical configuration of a logic emulation system.

a physical target system. Typically, the host workstation (Figure 30.1) performs the simulation of target system components and provides inputs to the emulated design via a host interface such as a backplane bus or cable. Design outputs are returned to the host workstation via the same path. In most cases, only the most time-consuming portion of the design under test is mapped to the emulator. The rest is simulated on the companion processor located in the host workstation. Coverification is often used to concurrently verify software components running on both the processor in the host workstation and in the emulated design.

In contrast to simulation, the use of in-circuit emulation and coverification allows for exhaustive prefabrication functional testing [3]. Typically, logic emulation can provide about five to six orders of magnitude speedup versus simulation for a logic design [2, 14]. Numerous commercial ASIC projects have used coverification to confirm the functionality of end applications with billions of test vectors prior to chip fabrication [3]. The speed of in-circuit emulation often allows for complete software system design verification as soon as a functionally specified ASIC design is complete. In the case of microprocessor design, a significant fraction of the emulated processor's software system can be tested long before processor fabrication, ensuring the functionality of both hardware and software. For example, Unix was successfully booted on an emulated M68060 microprocessor in about two hours [14]. This value represents a 40,000 times speedup over RTL simulation for the same processor operation.

30.3 TYPES OF LOGIC EMULATION SYSTEMS

For many designers of small ASICs, a large, expensive multi-FPGA emulation system may be unnecessary because one large FPGA and some associated external memory may be sufficient to implement the entire ASIC design.

30.3.1 Single-FPGA Emulation

The use of a single FPGA simplifies emulation system mapping because design partitioning and inter-FPGA routing are unneeded. Often, an unmodified RTL description of the ASIC design can be resynthesized for the FPGA with the use of an alternate synthesis library. Standard FPGA compilation tools are then used to complete the design mapping. As shown in Figure 30.2, the FPGA used for prototyping is typically mounted on a custom board that receives design inputs either from a target system where the completed ASIC design eventually will be located or from a workstation that provides input test vectors via a download cable. Additional interfaces are usually provided to allow for connections to a power supply and a logic analyzer. Since most FPGAs used for prototyping are SRAM based, resources must be provided to store and download the configuration bitstream to the FPGA at power-up.

As the logic capacity of FPGAs grows, it may appear that an increasing number of ASIC designs could be prototyped using a single FPGA. However, since both FPGA and ASIC gate counts follow the same VLSI process trends,

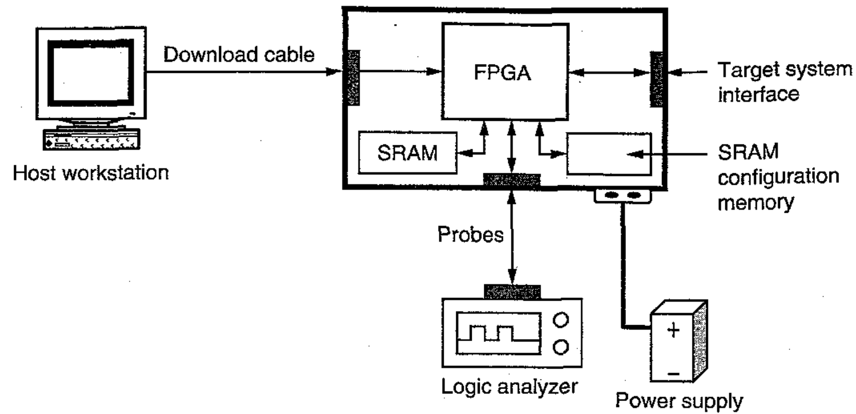


FIGURE 30.2 ■ An example of a single-FPGA logic emulation system.

it is likely that most ASIC designs will continue to require multiple FPGAs for verification.

30.3.2 Multi-FPGA Emulation

Contemporary multi-FPGA emulation systems are complex verification platforms containing hundreds of FPGA and memory chips, high-speed interfaces to target systems, hosts, logic analyzers, and support for interactive debugging [11]. Since their initial commercial introduction in 1988, these systems have evolved into important functional verification platforms [7]. Typical systems include multiple boards each containing tens of FPGA devices interconnected in a fixed topology. Interboard communication is performed via fixed connections or a backplane bus. Because of the need to communicate signals between FPGAs, the typical frequency of an emulated design is in the range of 0.5 to 5 MHz.

Two distinguishing characteristics of a multi-FPGA logic emulator are the topology used to interconnect FPGAs and the approach used to communicate interpartition logic signals between them. Before addressing the issues of topology, two possible approaches for assigning logical signals to inter-FPGA wires will be analyzed.

Consider the mapping of a simple circuit shown in Figure 30.3(a) to two FPGAs as shown in Figure 30.3(b). For this circuit, two interpartition signals (x and y) exist. One approach to mapping these signals to inter-FPGA wires is to dedicate them to inter-FPGA wires A and B, respectively, as shown in Figure 30.3(b). This *dedicated-wire* mapping preserves the original structure of the circuit and does not require the inclusion of any additional logic. In contrast, the mapping shown in Figure 30.3(c), adds pipeline flip-flops and a multiplexer to interpartition signals so that inter-FPGA wire A can be shared. From the figure it can be seen that wire A is multiplexed to transport both x and y. This *multiplexed-wire* approach allows for more efficient use of FPGA

pins and inter-FPGA wires, at the cost of additional FPGA logic and flip-flops. However, in most emulation systems I/O pins are a more precious resource than logic and flip-flops.

Both dedicated-wire and multiplexed-wire FPGA-based emulators are commercially available. Dedicated-wire systems include the SystemRealizer [24] and Mercury [25] families from Cadence; multiplexed-wire systems include Cadence Xcite [36] and the Mentor Graphics VirtuaLogic [21] and VStation [22] families. For dedicated-wire systems, design logic partitions must meet both the pin and gate count requirements of the target FPGAs. In virtually all cases, the FPGAs are pin limited, constraining the amount of logic and associated I/O that can be assigned to each FPGA. Rent's Rule [17], an empirical relationship that quantifies the growth of pin requirements as logic capacity increases, indicates that this problem is likely to get worse as FPGA logic capacity increases. As a result,

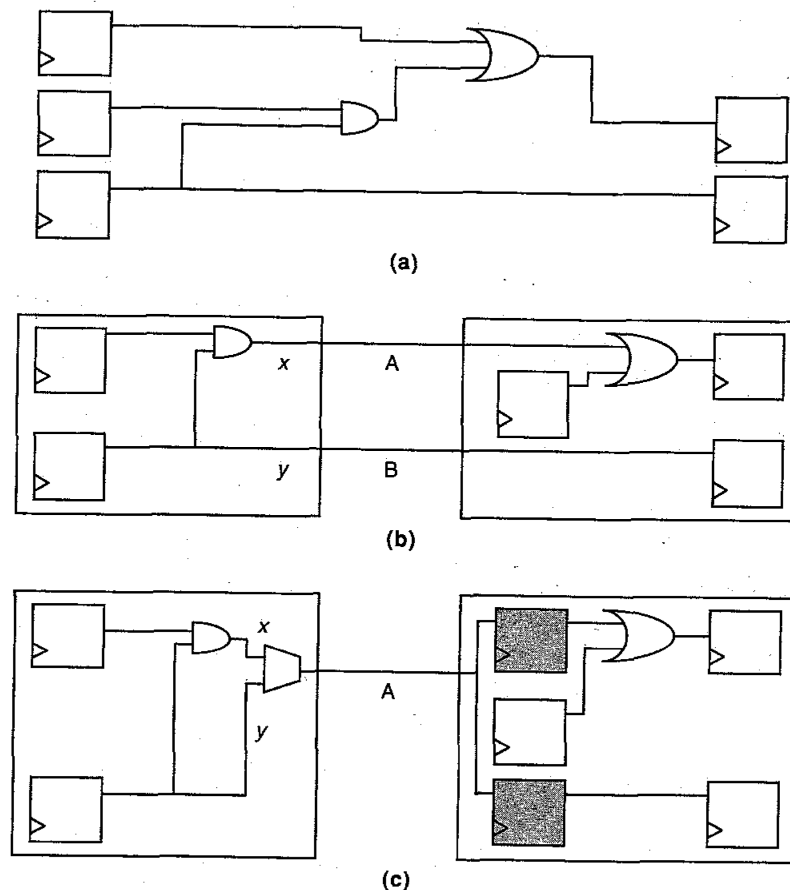


FIGURE 30.3 ■ The mapping of a simple circuit (a) by dedicated-wire(b) and multiplexed-wire (c) assignment.

the time-multiplexed use of pin resources is prevalent in contemporary emulation systems.

A series of topologies for FPGA interconnection have been investigated for both dedicated-wire and multiplexed-wire emulators. A number of early commercial dedicated-wire emulation systems organized FPGAs primarily in a near-neighbor or low-dimensional mesh topology, as illustrated in Figure 30.4(a). Although these topologies are easy to build, their lack of routing flexibility complicates design partitioning. Since many interpartition connections may not have direct FPGA-to-FPGA connections, one or more FPGAs are required to provide through-hop connectivity. Not only does this make the timing along interpartition connections unpredictable, but scarce FPGA pin resources must be dedicated to through-hop connections. As a result, direct-connect dedicated-wire systems are now used only for emulation systems with a very small number of FPGAs (typically four or less) [4]. These systems often allow direct connections between all FPGAs, eliminating the need for through-hops.

In an attempt to provide predictable FPGA delay and eliminate the need for through-hops, a series of emulation systems were developed that use specialized crossbar devices called field-programmable interconnect chips (FPICs) in addition to FPGAs [7]. These systems route most or all inter-FPGA connections through the FPICs so that the length of each inter-FPGA path is predictable. For basic systems, such as the one shown in Figure 30.4(b), some of each FPGA's I/O pins are dedicated to bidirectional connections on each FPIC device forming a crossbar. As a result, any inter-FPGA connection can be made by passing through a single FPIC, leading to predictable timing. Multiple levels of FPIC interconnect allow for system scaling to hundreds of FPGAs. The delay for each individual path is predictable because the FPIC's timing is predictable, although the number of FPICs traversed by different inter-FPGA paths may vary.

Most multiplexed-wire systems use meshes with primarily near-neighbor connectivity [7, 34]. Inter-FPGA paths are pipelined, so each path has a predictable

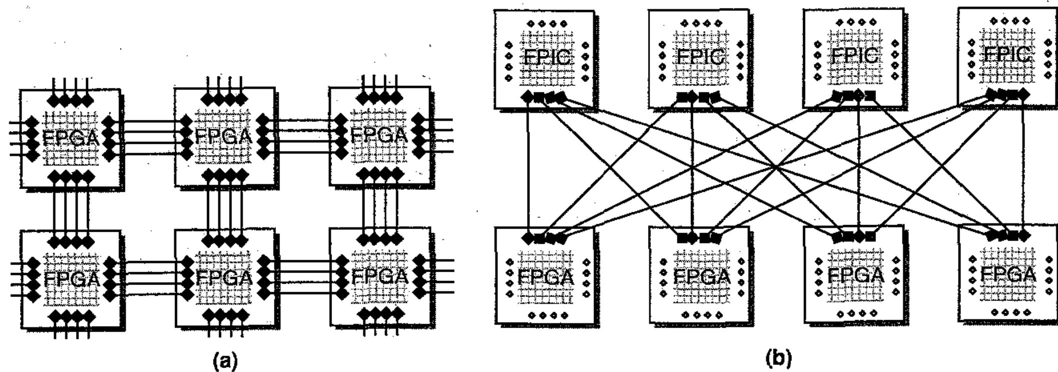


FIGURE 30.4 ■ Example FPGA-based logic emulator topologies: (a) mesh; (b) crossbar. *Source:* Adapted from Hauck [7].

delay, which is a multiple of the system clock frequency. Additionally, inter-FPGA routing congestion is overcome by the reuse of inter-FPGA routing resources, eliminating the restrictions created by through-hops. Although some multiplexed-wire systems that use partial or full crossbars (FPICs) have been proposed [19], the need for these expensive devices in time-multiplexed systems is unclear.

30.3.3 Design-mapping Overview

Several key issues drive the use of logic emulation systems. For most emulation products, system ease of use and resource utilization are important factors in system design. The translation of designs from ASIC netlist to multi-FPGA implementation must be fully or nearly automatic. These ease-of-use issues require sophisticated multi-FPGA computer-aided design approaches to process netlists in addition to the per-FPGA processing for numerous individual FPGAs.

A high-level flow for multi-FPGA logic emulation similar to the flow outlined by Hauck and Agarwal [8] is shown in Figure 30.5. It starts with a circuit description that is specified at the behavioral or register transfer level. Design translation, which typically includes logic synthesis, converts the high-level netlist to a gate-level structural equivalent. Following design translation, design logic is partitioned into pieces that will fit within the logic resources of individual FPGA devices. Partitioning is often performed to minimize required inter-FPGA interconnect, control system-wide critical path delay, and localize memory access. For some systems, partitioning must be performed so that inter-FPGA routing restrictions in terms of available FPGA pin count and system topology are considered. If the logic emulator contains memory chips that are external to the FPGA, design memory must be partitioned across memory resources to meet memory chip capacity constraints.

Partitioned design logic and memory structures are subsequently assigned to specific system devices via global placement. For some systems, swap-based placement algorithms, which are similar to the FPGA placement approaches described in Chapter 14, are used. A placement cost metric based on distance and delay is often iteratively used to judge placement quality. Partitioning and placement are sometimes combined into a single step to concurrently optimize interpartition bandwidth and inter-FPGA signal delay and distance [8]. The communication of interpartition signals between FPGAs is determined based on routing algorithms. For most multi-FPGA emulators, routing involves the determination of the shortest feasible path between FPGAs using available board interconnect resources for each inter-FPGA signal [2]. Topology constraints often require these signals to pass through intermediate (through-hop) FPGAs.

The last mapping step in logic emulation involves the individual compilation of the FPGAs. Multi-FPGA emulation systems have a number of constraints that can lead to less-than-efficient FPGA use. The FPGA compilation step may require hundreds of individual compiles. If even one design partition fails to successfully map to its target FPGA, the emulation flow shown in Figure 30.5 must be restarted from the design partitioning step. As a result, design partitions are often sized conservatively to ensure successful compilation.

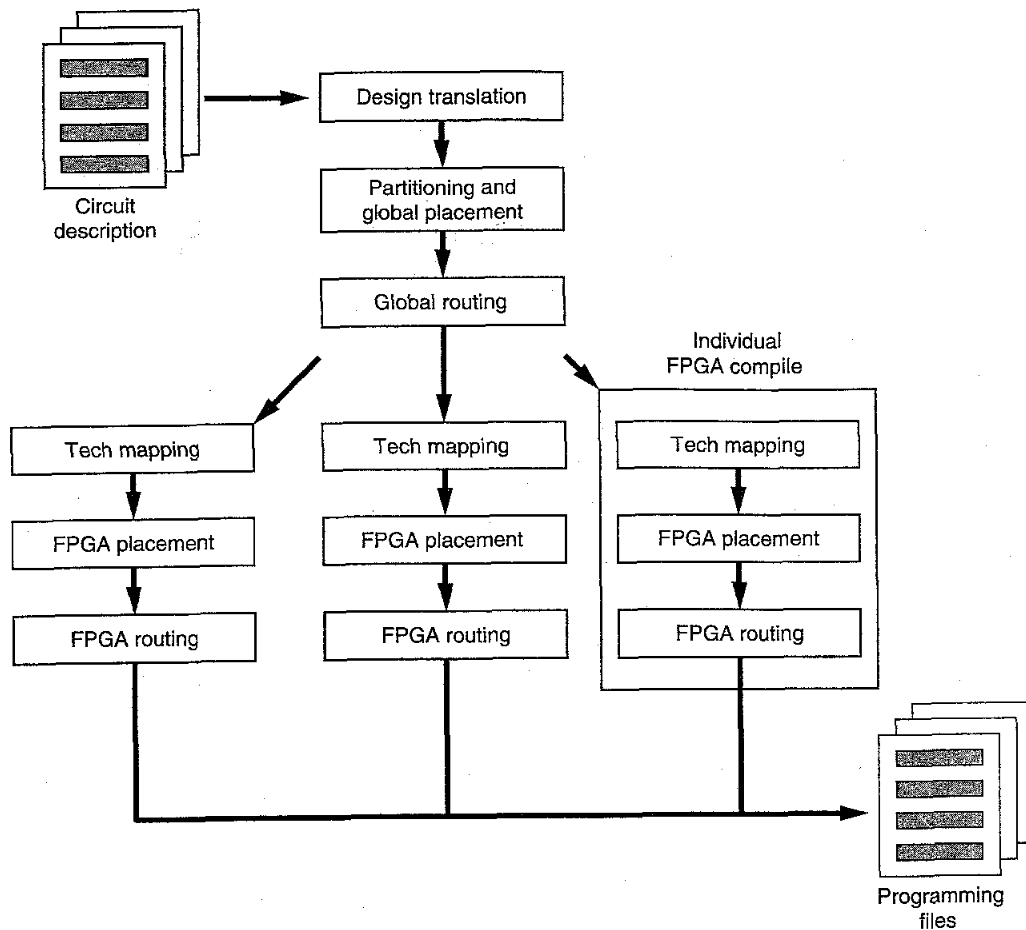


FIGURE 30.5 ■ A typical multi-FPGA emulator mapping flow. *Source:* Adapted from Hauck and Agarwal [8].

Although the steps just described define the high-level mapping flow for FPGA-based logic emulators, the specific partitioning, placement, and routing approaches used by individual emulators are heavily influenced by the approach used to communicate intermediate data signals between FPGAs. Although similar, dedicated-wire and multiplexed-wire emulators require specialized partitioning, placement, and routing algorithms.

30.3.4 Multi-FPGA Partitioning and Placement Approaches

Design partitioning and placement play an important role in system performance for dedicated-wire FPGA-based logic emulators. Because FPGA pins are such critical resources for these systems, the primary goal of partitioning is to minimize communication between partitions. A large number of algorithms

have been developed that split logic into two pieces (bipartitioning) and multiple pieces (multiway partitioning) based on both logic and I/O constraints. Unfortunately, the need to satisfy dual constraints complicates their application to dedicated-wire emulation systems.

One way to address the partitioning and placement problem is to perform both operations simultaneously [8]. For example, a multiway partitioning algorithm can be used to simultaneously generate multiple partitions while respecting inter-FPGA routing limitations [28]. Unfortunately, multiway partitioning algorithms are computationally expensive (often exhibiting exponential runtime in the number of partitions), which makes them infeasible for systems containing tens or hundreds of FPGA devices. As a result of inter-FPGA bandwidth limitations and the need for reasonable CAD tool runtime, most dedicated-wire FPGA emulation systems use iterative bipartitioning for combined partitioning and placement [6]. This approach has been effectively applied to both crossbar and mesh topologies [31].

The use of recursive bipartitioning for dedicated-wire emulators creates several problems. Although it can be used effectively to locate an initial cut, it is inherently greedy. The bandwidth of the initial cut is optimized, but may not serve as an effective start point for further cuts. This issue may be resolved by ordering hierarchical bipartition cuts based on criticality [5].

Partitioning for multiplexed-wire systems is simple compared to the dedicated-wire case, because it must meet only FPGA logic constraints, rather than both logic and pin constraints. Unlike the dedicated-wire case, partitioning and placement are generally performed not simultaneously but rather sequentially [2]. First, recursive bipartitioning successively divides the original design into a series of logic partitions that meet the logic capacity requirements of the target FPGAs. During partitioning, the amount of logic required to multiplex inter-FPGA signals must be estimated because both design partition logic and multiplexing logic must be included in the logic capacity analysis. Following partitioning, individual partitions are assigned to individual FPGAs. Placement typically attempts to minimize system-wide communication by minimizing inter-FPGA distance, particularly on critical paths. To fully explore placement choices, simulated annealing is frequently used for multi-FPGA placement [2].

30.3.5 Multi-FPGA Routing Approaches

The global routing step determines which FPGAs are used to route inter-FPGA signals. Inter-FPGA routes may directly connect source and destination FPGAs, or intermediate through-hops may be necessary. Global routing algorithms typically attempt to minimize distance and inter-FPGA routing resource usage while ensuring that no routing resources are overused.

The routing problem for dedicated-wire systems is similar to the intra-FPGA routing problem described in Chapter 17. In dedicated-wire systems, the amount of available inter-FPGA wiring is fixed, possibly leading to infeasible or inefficient routes if an effective routing algorithm is not employed. Groups of wires between FPGAs are considered a communication channel, and inter-FPGA

routing channels can be represented as a directed channel graph. As seen in Figure 30.6, for a direct-connect topology, the edge weight in the channel graph represents the number of physical wires in the channel [8]. Prior to routing, the channel graph for the system topology in Figure 30.6(a) can be represented as in Figure 30.6(b).

As routing is performed, inter-FPGA connections are assigned to wires, reducing the available capacity in each channel. A variant of maze routing [18] is typically used to assign inter-FPGA signals to specific system wires. Like the maze-routing algorithms used for intra-FPGA connections, multiple router iterations are often necessary. The maze-routing algorithm works by selecting a wire and finding the shortest feasible path from its source to its destination partition. Multiple iterations involving rip-up may be necessary to complete all routes.

The example mapping in Figure 30.7 provides an overview of the use of channel graph representation. Following the assignment of logical signals from the mapped design in Figure 30.7(a) to inter-FPGA wires, the channel availability is modified to take used wires into account. The effects of this assignment are shown in Figure 30.7(b), where the modified channels are shown with dashed lines.

For multiplexed-wire systems, both intra-FPGA computation and inter-FPGA communication are synchronized by a global system clock. This clock provides control over the sequence of events in the time-multiplexed system. Because many combinational evaluations and signal transfers occur in a single design (emulation) clock cycle, the system clock must operate at a faster speed than that of the design clock of the emulated design. Thus, routing in multiplexed-wire

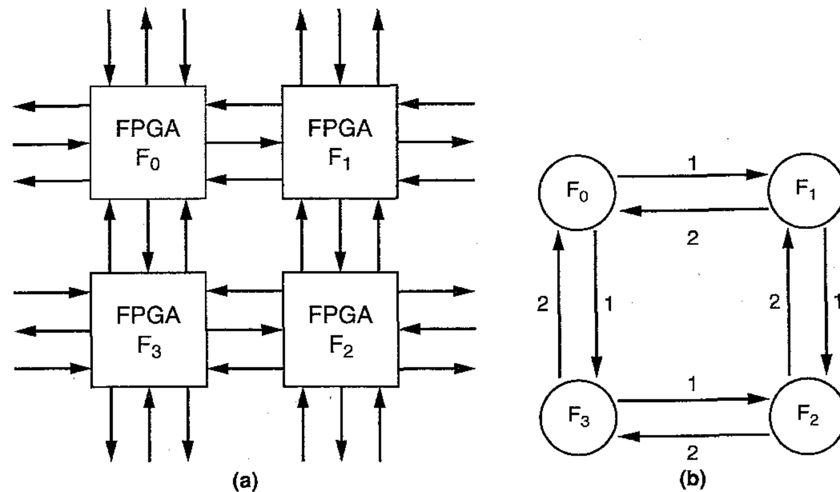


FIGURE 30.6 ■ (a) A multi-FPGA interconnection and (b) the associated channel graph for dedicated-wire routing. *Source:* Adapted from Hauck and Agarwal [8].

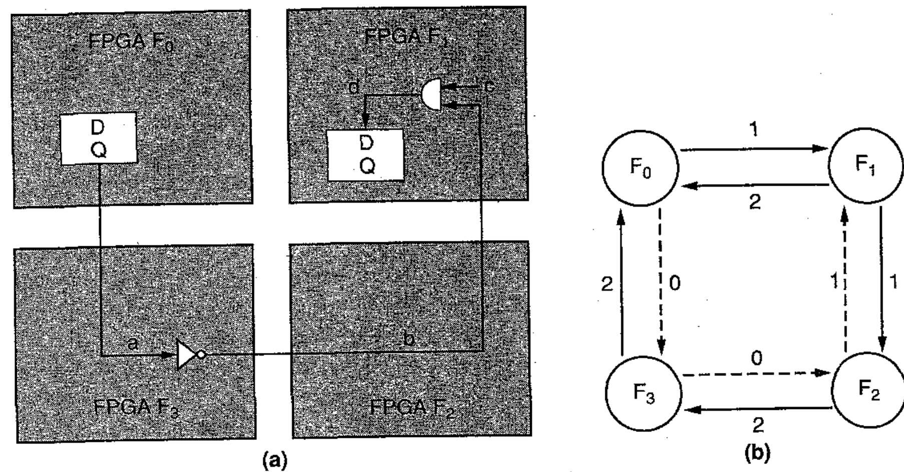


FIGURE 30.7 ■ Assignment of logic signals to inter-FPGA wires in a dedicated-wire system (a), and the resultant mapping (b).

systems assigns each interpartition wire a source–destination path schedule in both time and space.

Routing for multiplexed-wire systems generally requires two routing steps to connect an inter-FPGA signal: the determination of a feasible path between FPGAs and the scheduling of multiplexed signal transport along the path [2]. Initially, a path between source and destination FPGAs is determined using a shortest-path algorithm. Unlike dedicated-wire routing, the utilization of wires in the channel is less restrictive because a different signal may be assigned to each wire on each clock cycle. Following path selection, a data signal can be transmitted along an inter-FPGA path as soon as it is assigned a valid logic value by the flip-flop or logic gate that drives it. To complete the transmission, the signal is assigned to a series of inter-FPGA wires along the path until it reaches the destination FPGA. One clock cycle of the system clock is allowed for each inter-FPGA hop along the path. Because inter-FPGA paths are synchronized at FPGA boundaries with pipeline flip-flops, long combinational paths are effectively broken into a series of discrete timesteps. A number of scheduling algorithms that perform the assignment of interpartition signals to inter-FPGA wires have been developed [2, 32].

The result of routing using multiplexed wires is illustrated in the following example taken from Tessier and Jana [34]. In Figure 30.8, the circuit shown in Figure 30.7(a) has once again been partitioned onto FPGAs interconnected using the direct-connect FPGA topology shown in Figure 30.6(a). Each inter-FPGA signal can travel only between two FPGAs during each system clock cycle. In the figure, pipeline flip-flops, which have been added to allow multiplexed communication on each path, are shaded. Circuit communication and computation in terms of system clock cycles can be determined by evaluating the critical path from signal *a* to signal *d*, as shown in Figure 30.9. In both Figures 30.8 and 30.9, system clock cycles are labeled V_1 through V_5. In Figure 30.8, communication delays

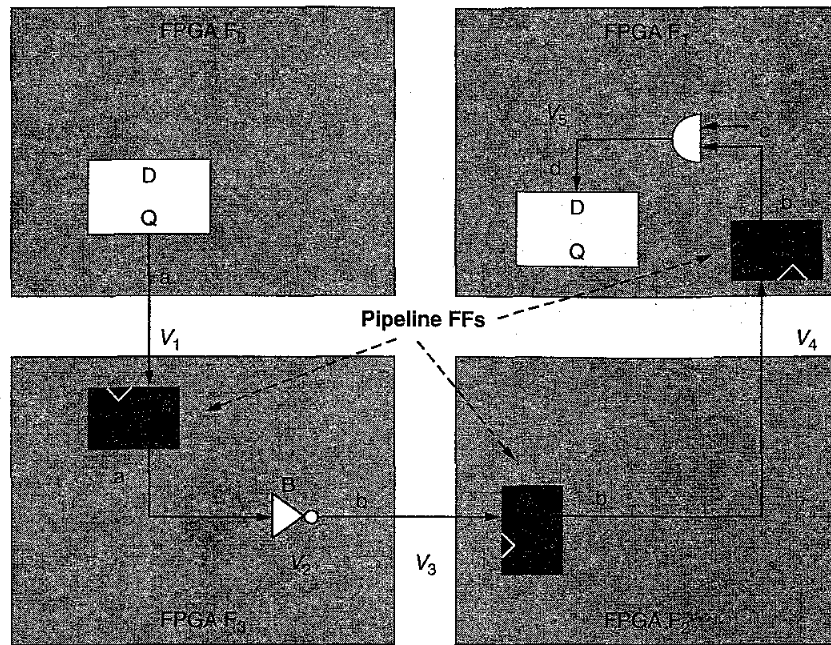


FIGURE 30.8 ■ Circuit mapping to FPGAs for a multiplexed-wire system.

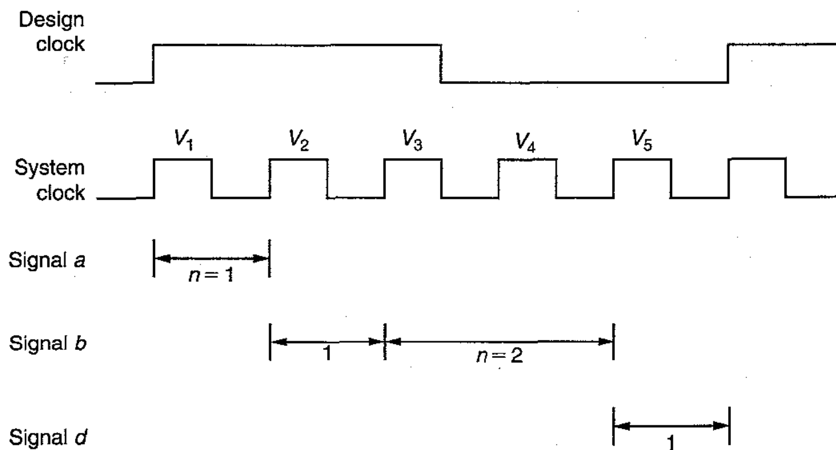


FIGURE 30.9 ■ The design clock cycle for the circuit mapping shown in Figure 30.8. Spans labeled *n* indicate a communication delay of *n* system clock cycles.

are listed, with n equal to the number of system clock cycles required for communication. Combinational evaluations are listed, with a number (e.g., 1). After system cycle V_5, signal d is latched into a design flip-flop, completing the design clock cycle.

The schedule for this example does not depend on the binary value of individual signals. Each interpartition signal is transmitted during each design cycle, whether or not it has changed. Alternative, dynamic scheduling approaches, which only transmit changed signals, have also been proposed [16]. For dynamic scheduling, the availability of the communication resources must be determined at *runtime*, which can significantly increase the amount of communication control circuitry needed in each FPGA. Kwon and Kyung [16] used a global controller and a shared bus to control dynamically scheduled data movement.

30.4 ISSUES RELATED TO CONTEMPORARY LOGIC EMULATION

30.4.1 In-circuit Emulation

As discussed in Section 30.2, a logic emulation system is often used to replace design logic in a target system. In-circuit emulation presents a series of challenges that often must be addressed by the user of the emulation system [11]. Since emulated designs operate at relatively slow clock rates, all or a portion of the target system must be modified to operate at a clock rate that is substantially less than the planned product clock rate. Special care must be taken to ensure that actions such as DRAM refresh and device phase-locked loop activity are not adversely affected. The clock for the target system must be interfaced to the emulator to control emulator logic evaluation. In some cases, the emulator provides the target system clock, simplifying synchronization.

30.4.2 Coverification

As described in Section 30.2, coverification requires the logic emulator to verify a portion of a design at the same time the rest of the design is simulated on a host workstation. Typically, the physical interface between the host and the emulator is the limiting factor to coverification performance [12]. A cycle-based approach to coverification requires a data exchange between the host and the FPGA-based emulator during each design clock cycle edge. This exchange includes collating inputs for the emulated design from the simulation database, transferring the inputs to the host interface via the appropriate software driver, collecting the generated results from the emulator, and returning the values to the simulator. The amount of time needed by the host to perform these transfer operations is often significantly longer than the time to evaluate the logic for a single design clock cycle on the emulator.

Transaction-based host–emulator interfacing has been introduced as a way to reduce interface time [12]. In transaction-based interfacing, the host-based simulator and FPGA-based emulator operate independently for a number of design clock cycles, limiting the amount of data that must be transferred across the host–emulator interface. Transaction-based interfacing often

works best for stream-based computations where dependencies between the simulated and emulated designs are minimal, allowing independent operation [27]. A detailed example of transaction-based coverification will be presented in Section 30.7.

For coverification environments, the simulation performed on the host workstation can take a variety of forms. Most commonly, an RTL or behavioral representation of a system component written in a hardware description language is simulated with a commercial HDL simulation tool. Following preliminary verification, some simulated components may then be synthesized and mapped to the logic emulator. Alternately, a software version of the simulated system components (typically in C/C++) may be used [27].

30.4.3 Logic Analysis

Logic analysis, the capturing of signal state around specific events of interest, plays an important role in FPGA-based logic emulation for both in-circuit emulation and coverification. Unlike processor-based logic simulation, which stores intermediate logic signals in a centralized memory, intermediate signals in FPGA-based emulation are physically distributed throughout the emulation system. As a result, for emulation the signal set of interest usually must be selected prior to compilation so that probing circuitry can be added to the design under test. The data collected by this circuitry can then be connected to an external logic analyzer or sent back to the host workstation for display. In some cases, combinational signals can be reconstructed from saved design flip-flop values via simulation once emulation is complete [20]. Signal reconstruction allows for a significant reduction in the amount of probe circuitry required within the logic emulator, and limits the amount of signal data transferred from the emulator after each design clock cycle.

Because of their cycle-accurate operation, logic analysis for FPGA-based emulators has several additional, unique characteristics:

- FPGA-based emulators can only perform functional verification, so only combinational and flip-flop values captured on design clock edges accurately indicate design behavior.
- If the set of design signals selected for probing is changed, one or more FPGAs may need to be recompiled to implement the change.
- Logic analysis for a design can be triggered by prespecified logic conditions in the design. This triggering circuitry can be added to the design under test.

Logic emulators can be used to evaluate millions of design clock cycles, so there often has to be a trade-off between the number of probes and the number of consecutive clock cycles probing is performed. If emulation can be stopped, intermediate probe values can be offloaded to the host workstation or to a disk. Emulation can then be restarted [20].

30.5 THE NEED FOR FAST FPGA MAPPING

Commercially available FPGAs are optimized to provide good performance and mapping efficiency to a wide range of user designs. As seen in Chapter 1, contemporary off-the-shelf FPGAs offer a diverse and flexible routing network to reach this goal. To achieve modest to high logic resource utilization (e.g., greater than 75 percent lookup table [LUT] usage) and high design performance, an FPGA's mapping tools must perform a detailed evaluation of FPGA placement and routing choices, typically requiring 30 minutes to several hours of compile time per device. As a result, most FPGA-based logic emulators suffer from long compile times, which is a major limitation to their widespread deployment. The presence in an emulator of hundreds of FPGAs with significant compile times can considerably delay the debug, redesign, and retest cycle for a design under test. As noted in Chapter 20, several research projects have investigated accelerated FPGA mapping to solve this problem.

There are several reasons why fast FPGA design mapping for logic emulation is important:

1. The sheer number of FPGAs needed for logic emulation necessitates fast compilation. If compilation can be accelerated by an order of magnitude, so too, roughly, can the turnaround time from design change to emulator implementation. For many systems, faster design turnaround time can make a substantial difference in emulator usability, especially early in the design cycle when design errors are more prevalent.

2. A fast mapping is useful for determining if all logic partitions will fit within emulation system FPGA devices. If any partition fails to map into the emulator, the entire emulation mapping flow typically must be restarted from scratch.

3. Because multiplexed-wire emulation systems require the use of a synchronous global clock to coordinate computation and communication, the overall system clock speed is dependent on the slowest FPGA. A fast evaluation of achievable clock speed is therefore important. A fast mapping helps identify if the partitions are likely to meet the emulator's target system clock speed.

4. The inclusion of probes, which are frequently changed, necessitates a fast design compilation turnaround. Changes generally affect only a small number of FPGAs, which usually can be recompiled quickly.

Of the emulation system mapping steps shown in Figure 30.5, the individual FPGA compiles collectively require over 90 percent of the total compilation time. However, unlike the other steps, individual FPGA compiles can be easily distributed to multiple PCs and workstations for parallel compilation [9]. A centralized server is used to control distribution of the compiles to the client workstations, collect the resulting FPGA configuration bitstreams, and verify that all compilation constraints have been met.

It will be difficult to significantly accelerate compilation for FPGAs with existing commercial architectures without a substantial increase in the ratio of routing resources to logic resources per device or improved parallel mapping

approaches for individual FPGAs. Fundamentally, FPGA placement and routing are dedicated resource assignment problems, and the search for a mapping solution is accelerated only through additional available resources or a parallel search. Although compile times for logic emulation can be significantly reduced by underpopulating commercial FPGA device logic in emulators, the hardware cost involved is prohibitive. Therefore, parallel FPGA placement and routing offer the most promise in improving compile times for existing FPGA architectures.

In many ways, FPGA compilation for a partition of an emulated design under test is more difficult than FPGA compilation for a single-chip design specifically created for an FPGA. All FPGA compiles for logic emulators must be performed with constrained pin assignments because inter-FPGA channel assignments are determined prior to individual FPGA compilation. Forced pin assignments make designs more difficult to map and require extended FPGA compilation times. Since partitions were not specifically designed for an FPGA, performance or utilization issues may sometimes arise during mapping.

30.6 CASE STUDY: THE VIRTUALOGIC VLE EMULATION SYSTEM

To illustrate many of the issues in logic emulation, we consider the VirtuaLogic VLE emulator from Mentor Graphics [9]. This system represents one point in a spectrum of similar FPGA-based emulation systems from Mentor Graphics, including the Avatar and the VStation [23]. The following analysis illustrates the basic approaches used by this family for system architecture, design compilation, external system interfacing, and coverification.

30.6.1 The VirtuaLogic VLE Emulation System Structure

Figure 30.10 illustrates the components of the VLE emulation system hardware, including its interfaces to a host workstation and target system [9]. The system chassis, shown on the right, can contain up to six multi-FPGA array boards, which emulate the logic and memory of a design under test. Two array boards are shown in the configuration in the figure. Each board contains 64 Xilinx XC4036XL FPGAs, arranged in an 8×8 array, and 32 32K $\times$ 32 single-port synchronous SRAM chips. As shown in Figure 30.11, each FPGA connects to its four nearest neighbors in both horizontal and vertical directions and to FPGAs two hops away in the horizontal and vertical directions. A single memory device is shared between each pair of FPGAs. Direct connections between each FPGA and the six I/O connectors on the array board provide an interface for in-circuit emulation connections, logic analysis, and host interfacing. As shown in Figure 30.10, these connectors are located at the front of each board.

The FPGA array boards connect to a passive backplane in the system chassis to create a scalable system. Each FPGA has direct connections through the backplane to FPGAs on other array boards. All intra-FPGA computation and inter-FPGA communication throughout the system is coordinated via a global

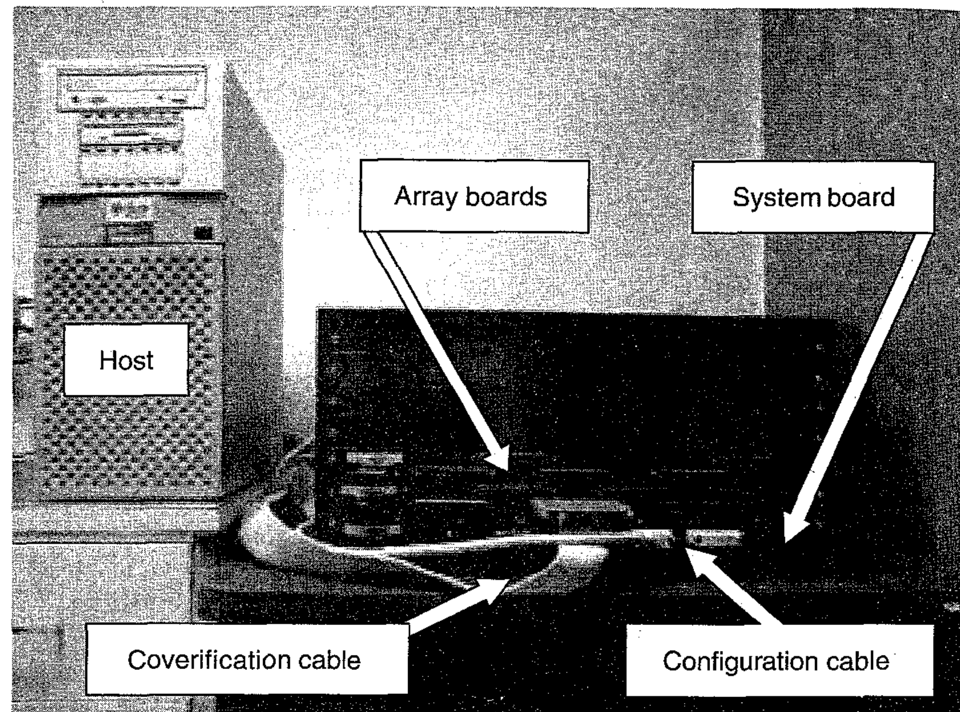


FIGURE 30.10 ■ A VirtuaLogic VLE-2M logic emulation system with two array boards.

system clock. The system board in the emulator controls the configuration of array board FPGAs and coordinates the distribution of the global system clock. Configuration bitstreams are loaded into the system board from the host workstation via an SCSI-2 cable.

30.6.2 The VirtuaLogic Emulation Software Flow

The emulation mapping flow for the VirtuaLogic VLE system follows the flow outlined earlier in this section. During design translation, an RTL netlist is converted to a gate-level design through the use of RTL synthesis. The mapped netlist is then partitioned into pieces appropriate for the logic capacity of each FPGA using algorithms that attempt to minimize bandwidth and encapsulate critical design paths within individual FPGAs.

Partitioning is performed so that the logic capacity of the FPGA is considered while partitioning to minimize bandwidth [1, 8]. For the multiplexed-wire VLE system, the number of logic gates required per partition can be represented as

$$G \geq G_P + c^*P$$

where G is the number of available gates in the FPGA, G_P is the number of user design logic gates in the partition, c is a constant representing the amount of

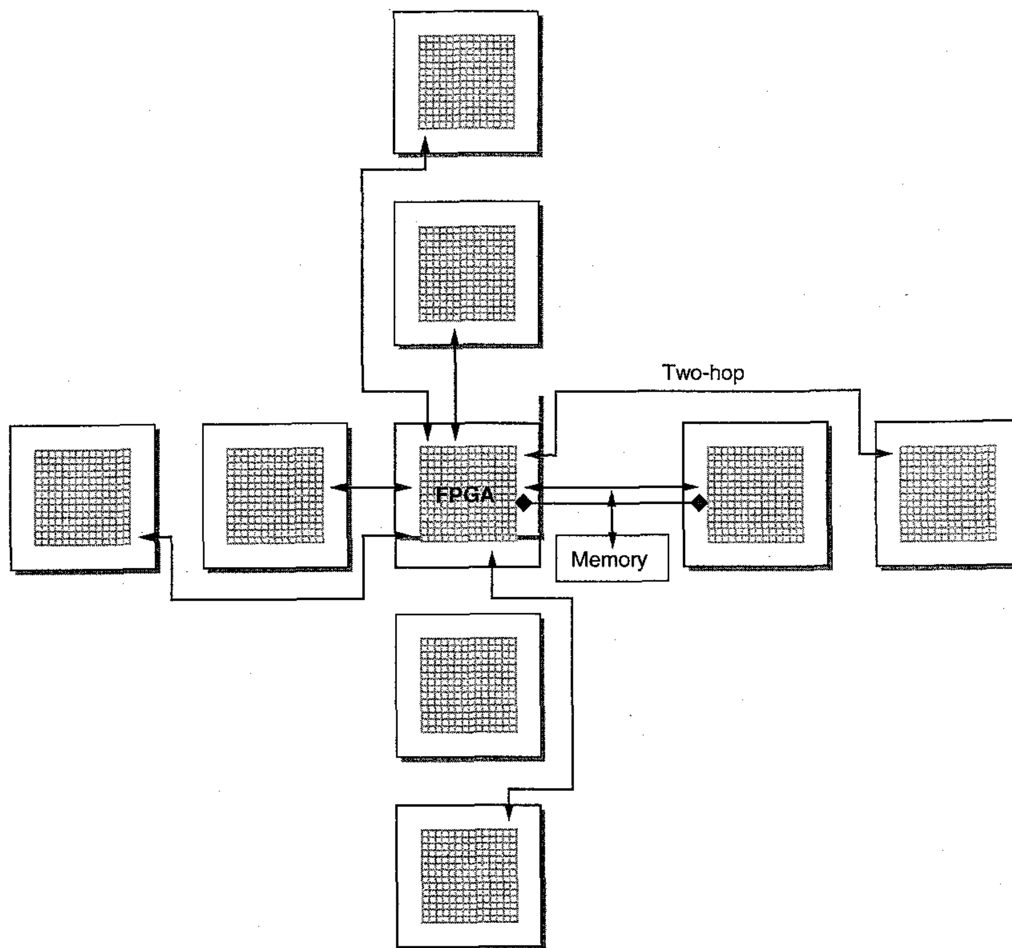


FIGURE 30.11 ■ The array board connections for an FPGA in the VLE logic emulation system.

logic required to multiplex a pin, and P is the number of I/O signals associated with the partition.

Design partitions assigned to an FPGA have a required gate count that is less than G. The partitioning process for the VLE system starts with an initial assignment of logic to partitions. Iterative mincut swapping is then performed to reduce the amount of I/O needed by each partition (the value P in the equation). Not only does this optimization reduce the amount of subsequent pin multiplexing for I/Os, but the amount of required logic per device is also reduced because G depends on P [8]. Partitions for this emulation system are subsequently placed using a simulated annealing placement algorithm [30]. In general, placement is performed to minimize the overall distance of inter-FPGA connections assuming that all connections will be scheduled along shortest paths. The logic partition

to FPGA assignment formulation is similar to the one used to place clusters inside an island-style FPGA.

A distinctive aspect of the VLE system is the statically scheduled routing approach used to make connections between signal sources and destinations. The approach used by the VirtuaLogic compiler follows that described in Section 30.4 [8, 34]. All intra-FPGA computation and inter-FPGA communication is synchronized to the global system clock cycle so that multiple system clock cycles are required to complete an emulation clock cycle. A signal may be routed between FPGAs on a specific system clock cycle once it is known to be valid for the current emulation cycle based on signal dependencies. The following steps are then taken to perform the statically scheduled routing of the signal between a source FPGA s_f and a destination FPGA d_f [34]:

1. The shortest feasible path P_{sd} between FPGAs s_f and d_f in terms of inter-FPGA channels is determined.
2. The send time T_s of the signal is determined. This is the system clock time slot at which the signal leaves s_f.
3. The signal arrives at FPGA d_f at the arrival time T_a of the signal. The arrival time is defined as $T_a = T_s + n$, where n is the number of FPGA chip boundaries (hops) between source FPGA s_f and destination FPGA d_f.

To illustrate the use of T_s and T_a, the schedule of the circuit shown in Figure 30.8 can be augmented to include send and arrival times. The communication schedule, including T_s and T_a values, is shown in Figure 30.12. Note that in Figure 30.8 signal b passes unchanged through FPGA F_2 on the path from

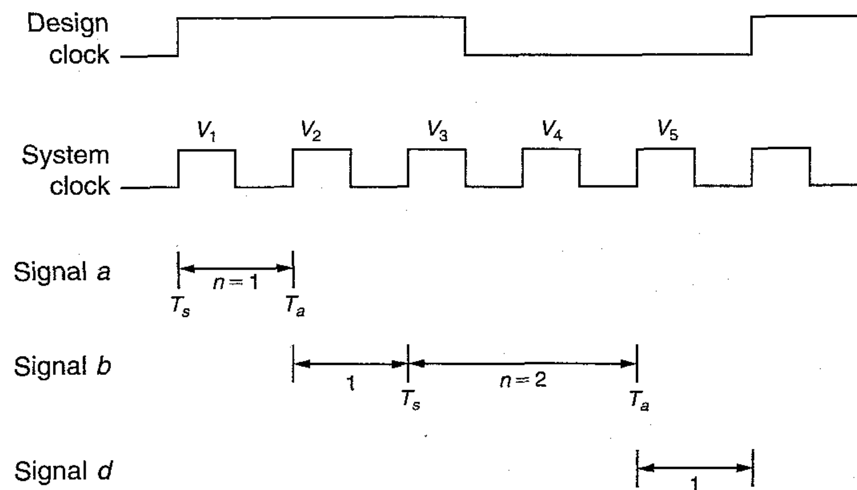


FIGURE 30.12 ■ The design clock cycle for the circuit mapping shown in Figure 30.8, including send times T_s and arrive times T_a.

FPGA F_3 to FPGA F_1. This through-hop is necessary given the lack of a direct FPGA F_3 to FPGA F_1 connection.

After each interpartition signal is scheduled for communication, the chosen schedule is implemented by synthesizing multiplexers, registers, and state machines that are added to the circuit partition for each FPGA. The resulting circuits are then applied to standard Xilinx Foundation design-mapping tools [37].

Most ASIC designs that are targeted for emulation contain complex logic and memory structures that require specialized processing outside the standard emulation mapping flow. For VLE systems, specialized mapping techniques have been developed to map complex design memories to emulation system memory chips [1], to map designs that contain multiple asynchronous design clocks [13], and to incrementally map design changes [34]. The algorithms created to address these mapping issues are important keys to system usability.

30.6.3 Multiported Memory Mapping

In a VLE system, multiple accesses to a $32K \times 32$ synchronous single-ported SRAM can be scheduled within a design (emulation) cycle to emulate the behavior of a multiport RAM. For example, Figure 30.13(a) shows a user-specified dual-port memory with two read ports and a single write port. During an emulation cycle access that requires reads from both read ports, both reads can be performed in sequence from the single-ported SRAM chip. As shown in Figure 30.13(b), a state machine can be used to sequence the application of the addresses to the single-ported SRAMs, and the storage of the read data in the output registers.

The VirtuaLogic compiler determines the schedule for data accesses in conjunction with routing address, data, and control signals to the on-board physical memory devices. Although not shown in the Figure 30.13, for data wider than the width of the physical memory, memory accesses can be made by sequentially accessing consecutive memory locations. For example, a read of a 128-bit value requires four system clock cycles. Dependency relationships for multiported RAMs (e.g., read-after-write) can be handled via the sequential scheduling of RAM accesses.

30.6.4 Design Mapping with Multiple Asynchronous Clocks

In Section 30.4 it was shown that for multiplexed-wire systems both intra-FPGA computation and inter-FPGA communication are coordinated to a global system clock. Because multiple system clock cycles are required to perform computation and communication for a single emulation clock cycle, a fixed relationship must exist between the clocks. Many contemporary ASIC designs contain multiple design clocks that operate asynchronously to each other. While synchronization between a system clock and a single design clock can be addressed by rising design clock edges that delineate functional evaluations, deriving a relationship between multiple asynchronous design clocks and a system clock is more difficult.

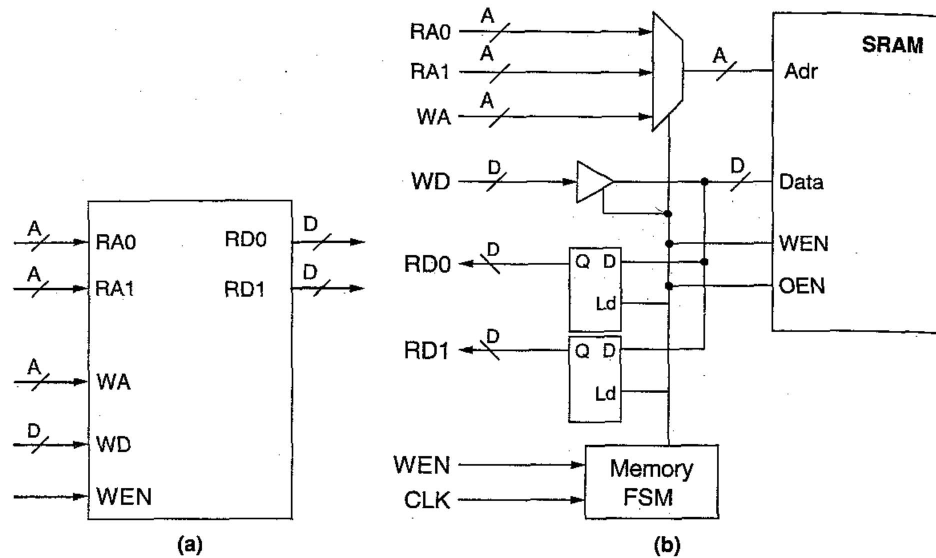


FIGURE 30.13 ■ A mapping of a multiported design memory to a single-ported emulator memory: (a) parallel-accessed multiport memory; (b) sequentially accessed single-port multiplexed memory. *Source:* Adapted from Agarwal [1].

In the circuitry shown in Figure 30.14, taken from Kudlugi and Tessier [13], the asynchronous clocks CLK1 and CLK2 drive state elements. It can be seen that signal N5 is a multidomain signal because it changes value and is sampled as a result of both CLK1 and CLK2 clock transitions. Now consider a situation where the circuit in Figure 30.14 is partitioned so the multidomain signal N5 must be transported from FPGA 1 to FPGA 4 as shown in Figure 30.15. In a multi-FPGA VLE system, the physical wires that connect FPGAs are grouped into unidirectional channels, where each physical wire is capable of carrying multiple signals that belong to the same emulation clock domain (e.g., CLK1 or CLK2).

Signal routing may include several intermediate FPGA hops. To simplify scheduling, logical signals assigned to the same inter-FPGA wire must be associated with the same clock domain. For designs with multidomain signals, this restriction requires that each multidomain signal be logically split into separate single-domain versions prior to transport. These single-domain values are then transmitted separately along separate physical channel links and combined at the destination to support multidomain behavior. Unfortunately, this approach of separately routing copies of the same signal along different links can lead to scheduling problems because each copy may arrive at the destination at different system clock cycles.

This issue is best illustrated through an example. As shown in Figure 30.15, communication for each asynchronous clock domain takes place over a different

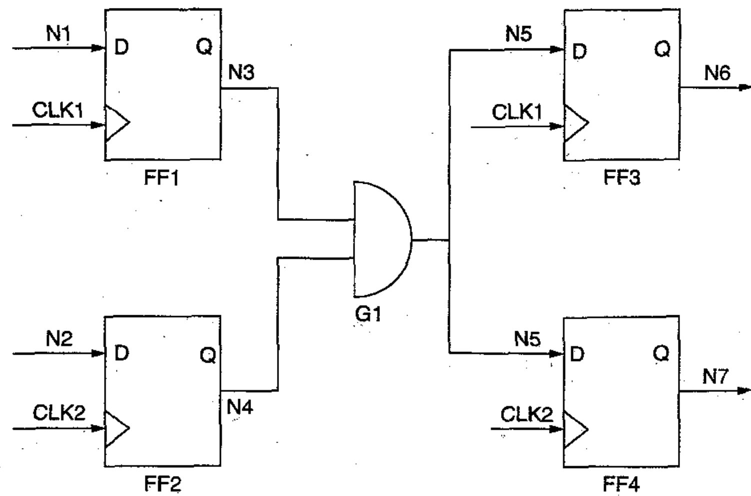


FIGURE 30.14 ■ A circuit that requires clocks from multiple asynchronous clock domains.

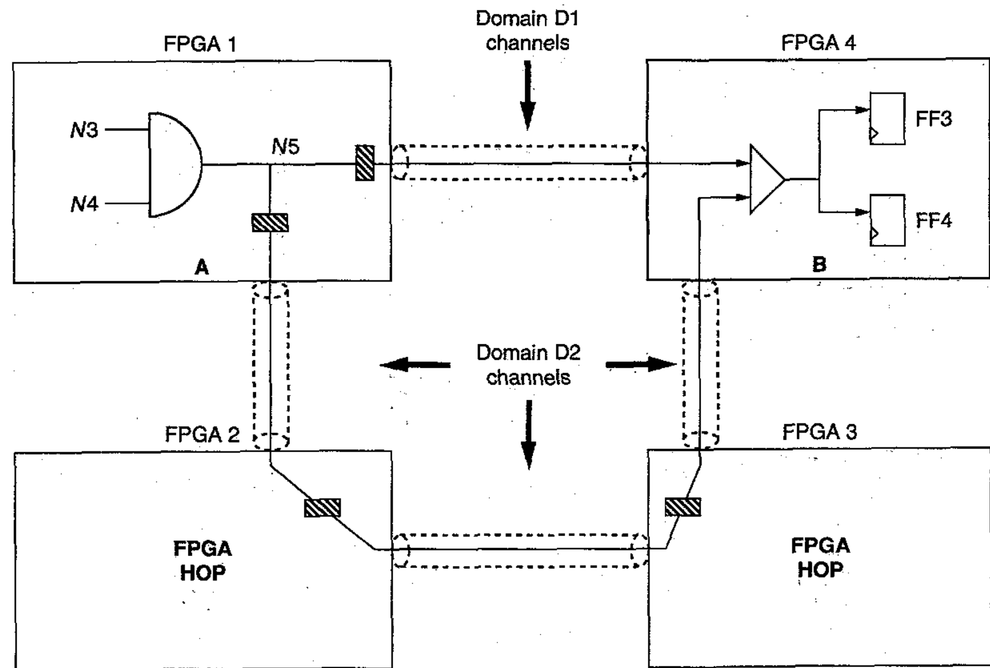


FIGURE 30.15 ■ An example of multidomain signal transport. *Source:* Adapted from Kudlugi and Tessier [13].

set of inter-FPGA channels. In the case of N5, paths using both domain 1 (D1) and domain 2 (D2) channels are needed to transport N5 between FPGA 1 and FPGA 2. The disjoint nature of multiple routing paths for the same logical signal can lead to differing arrival times for the copies of signal N5 at the destination FPGA. If both copies of signal N5 leave FPGA 1 at the same time, the D1 version of the signal will arrive at FPGA 2 two system clock cycles before the D2 version. This arrival order can lead to an incorrect logic evaluation if an attempt is made to use the D1 version of the signal before the D2 version arrives.

A requirement in transporting multidomain signals is to ensure that causality of events is guaranteed irrespective of routing delays. Causality can be preserved by ensuring that the length of the route for each domain from the source to the destination requires exactly the same number of system clock cycles. This can be accomplished by requiring the scheduler to use the same number of system clock cycles to communicate versions of the same signal to a destination FPGA. In Figure 30.16, for example, the scheduler must determine a path from FPGA 1 to FPGA 2 of length 3 for domain D1, since this is the path length of the domain D2 version. Each path now contains three pipeline flip-flops. The determination of the specific schedule may require several scheduling iterations because the length of the longest path is not known until each path is initially scheduled.

The scheduler used by the VirtuaLogic compiler takes multidomain paths into account and can handle designs with any number of asynchronous clock

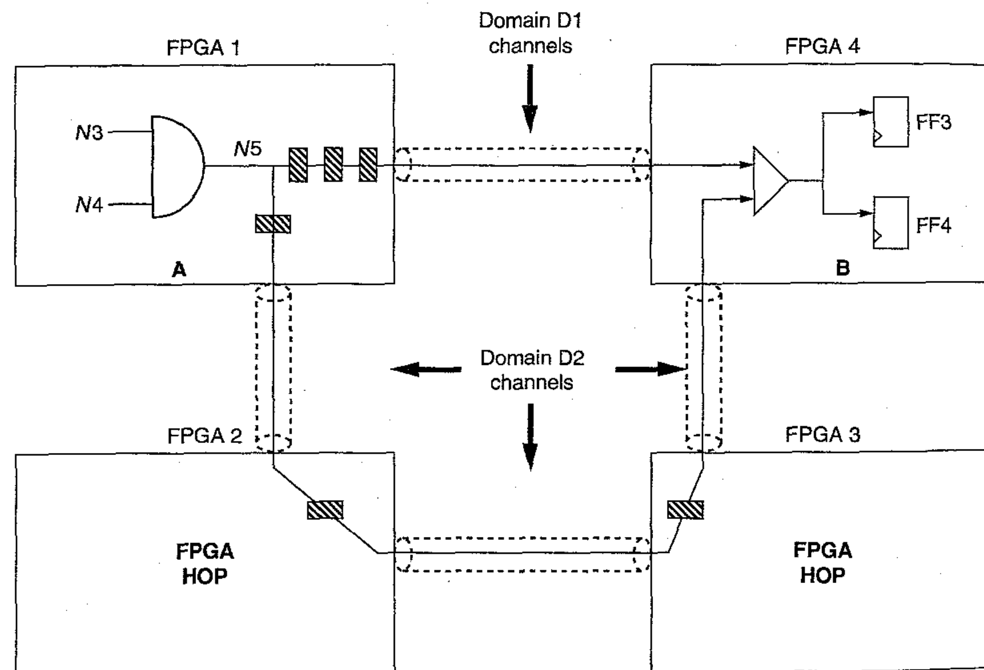


FIGURE 30.16 ▪ A retimed version of the multidomain signal transport shown in Figure 30.15.

domains. The mapping of this multidomain logic to the emulator takes place automatically. The asynchronous design clock signals may be interfaced to the emulator from outside the system through the system board.

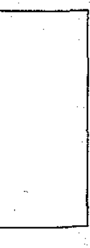
30.6.5 Incremental Compilation of Designs

The need for incremental design support in VLE systems is a result of recent interest in core-based design and system-on-a-chip integration. Most ASIC verification flows involve numerous iterations of design test, debug, and recompilation. As modifications are evaluated and errors are identified, the original design is subjected to a series of minor modifications. Often, a change may be isolated to a component that was originally spread across two or more FPGAs in the emulator. If emulator recompilation can be limited primarily to those FPGAs that contain logic affected by the change, the compilation process can be greatly accelerated. The ability to support design changes in a small set of FPGAs is crucial to avoid the need to recompile all FPGAs in the system from scratch. In addition to providing fast design turnaround, the resulting emulation performance of the incrementally compiled design should be the same or close to the same as the performance of the original design mapping [34].

The use of scheduling for VirtuaLogic inter-FPGA routing facilitates the management of incremental design compilation. A series of steps are required to address changes in the design and map them to the FPGA-based emulator [34]:

1. *Netlist comparison.* The first step in the incremental compilation process is to identify the logic and interconnect associated with the initial design that is no longer in the modified design. Subsequently, the logic and interconnect added to the initial design to create the modified design are identified. Logic removed from the initial design was assigned to a set of FPGAs as a result of initial design mapping. These modified FPGAs provide a possible destination for added logic.

2. *Incremental path identification.* In the VLE system, individual FPGAs may serve as through-hop steps for intermediate routes. Thus, even if a given FPGA does not contain logic that has changed, these FPGAs will require recompilation if they are used as through-hops for the modified logic. To limit compile time, the number of unmodified FPGAs selected to perform through-hop routing should be minimized.



3. *Incremental partitioning.* Once the modified and required through-hop FPGAs have been identified, newly added design logic can be partitioned onto them subject to processor logic and memory capacity constraints.

4. *Incremental routing.* Following incremental partitioning, routing is performed to create a path for the added design signals connecting the modified FPGAs. Because FPGAs surrounding the modified FPGAs are unaltered, this incremental routing must be performed using board-level routing resources that have not been consumed by unchanged design routes. Feasible shortest paths between FPGAs are evaluated and then incremental scheduling is used to form a communication pipeline.

The most important part of incremental compilation for multiplexed-wire systems is the scheduling of added signals onto available inter-FPGA wires (incremental routing). In some cases, portions of previously routed inter-FPGA links may need to be rerouted as a result of changed logic depth and dependency. Consider the circuit shown in Figure 30.17, taken from Tessier and Jana [34]. The circuit is the same as the one assigned to FPGAs in Figure 30.8 except that the OR gate F and signals e and f have been added. One potential incremental mapping for the modified circuit appears in Figure 30.18. A design clock

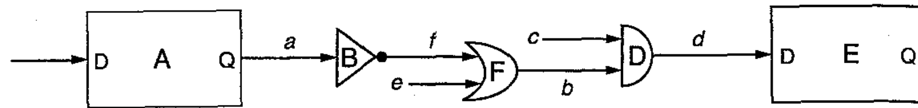


FIGURE 30.17 ■ A modified version of the circuit assigned to FPGAs in Figure 30.8. *Source:* Adapted from Tessier and Jana [34].

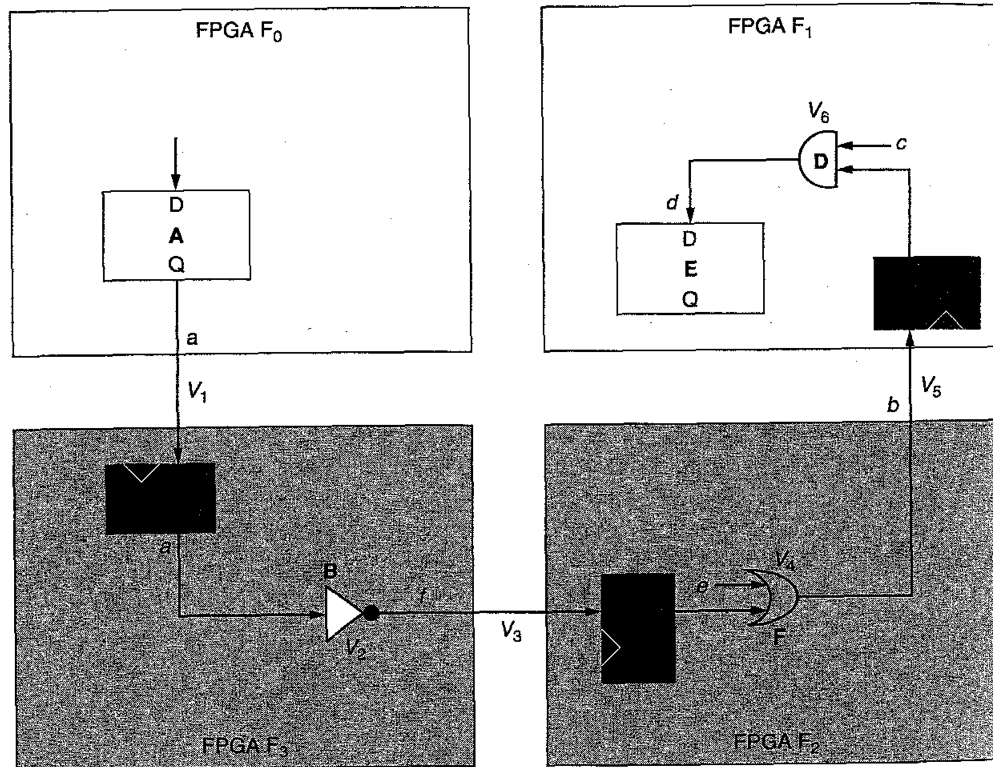


FIGURE 30.18 ■ An incremental mapping of the circuit shown in Figure 30.17.

cycle associated with the scheduled route of the circuit mapping in Figure 30.17 is shown in Figure 30.19.

When these waveforms are compared to the waveforms in Figure 30.12, it can be seen that an extra cycle of combinational delay has been added because of the OR gate evaluation in FPGA F_2, extending the number of system clock cycles needed to evaluate the design. Closer examination of the two sets of waveforms indicates that although signal b was previously routed between FPGA F_2 and FPGA F_1 in the initial design, it will have to be rerouted for the modified mapping. For the initial design, signal b has been routed between FPGA F_2 and FPGA F_1 on system clock cycle V_4. As a result of the mapping shown in Figure 30.18, signal b cannot be routed until system clock cycle V_5 because of combinational dependencies. This results in a need to recompile both FPGA F_2 to transmit the signal on cycle V_5 and FPGA F_1 to receive the value on system clock cycle V_5.

After dependencies are determined, the new links are scheduled for communication using the VirtuaLogic compiler two-step routing approach described earlier. Only added interpartition signals are routed; previously routed signals that are unchanged are left in place. Incremental routing of added signals may lead to an emulation system performance loss. For example, the waveforms shown in Figure 30.19 represent the schedule of the incrementally modified design shown in Figure 30.17. The new schedule requires six system clock cycles to complete a design clock cycle as opposed to the five required for the original design. Although not shown in Figure 30.19, a global control signal distributed to all FPGAs indicates the end of the design clock cycle. Following recompilation, this signal can be asserted every six rather than five system clock cycles. This requires FPGAs

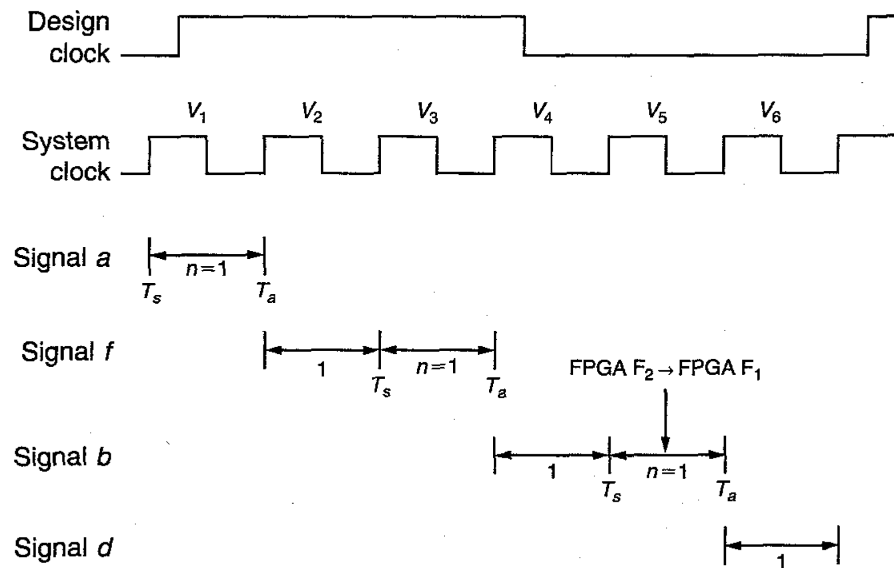


FIGURE 30.19 ■ The design clock cycle for the incremental mapping shown in Figure 30.18.

that were not recompiled to hold data values for an extra system clock cycle while the recompiled FPGAs complete computation. All results are then clocked into design flip-flops system-wide after six clock cycles by the design clock.

30.6.6 VLE Interfaces for Coverification

The VLE system has a number of interfaces to support both in-circuit emulation and coverification. For in-circuit emulation, an emulation pod can be interfaced to one of six connectors on each of the array boards shown in Figure 30.10. These signals are directly connected to FPGAs and drive/receive I/O signals on the emulated design. Tuned clock cables are used to control clocking both on the target system and in the emulator when the emulator has completed evaluation for an emulation clock. To permit in-circuit emulation the target system must be slowed to accommodate the 0.5- to 2-MHz design emulation rate.

In addition to support for in-circuit emulation, the VLE emulator has significant support for a variety of coverification modes. This support is primarily provided through a series of software interfaces created at the host workstation and on the emulator. These interfaces allow the emulator to be used in a variety of coverification scenarios [9]. Designers initiate ASIC verification by representing the ASIC using a high-level language such as C or SystemC (a C-compatible language that represents the concurrency and clocking associated with hardware implementations). As a design matures, portions of it are migrated to hardware. Inputs and outputs to the portion of the design on the emulator are interfaced to the emulator via an application programming interface (API).

The transfer, execution, and collection of results using the emulator can be represented as shown in Figure 30.20. This implementation of coverification is performed with a series of components. The software test environment interacts with an application adapter—that is, an interface to a series of library-based drivers that packetize the data and prepare it for transfer via a PCI-based board. The use of library-based drivers allows for communication at functional, bus-cycle-accurate, and cycle-accurate levels [27].

An interface circuit is required at the destination to reassemble data for subsequent use as input to the design. A transactor accepts the reassembled data, generates an emulation clock for use with the design under test, and coordinates per-cycle data transfer to and from the design. Generally, the interface circuit and transactor are created in RTL and added to the design. VLE systems use the transaction-based approach described earlier in this section. Transactions contain both data and synchronization information. A single transaction results in multiple verification cycles of work being performed by the emulator. The transaction can be as simple as a memory read or as complex as the transfer of an entire structured packet through a channel. To support coverification, the host for the VLE emulator contains an SPCI (Springtime PCI) card [27]. This custom PCI card implements the physical layer of transaction-based interfacing between the host and the emulator via a cable.

The transaction application protocol interface (TAPI) forms the application adapter for the VLE system [27]. TAPI consists of a library of C functions. The

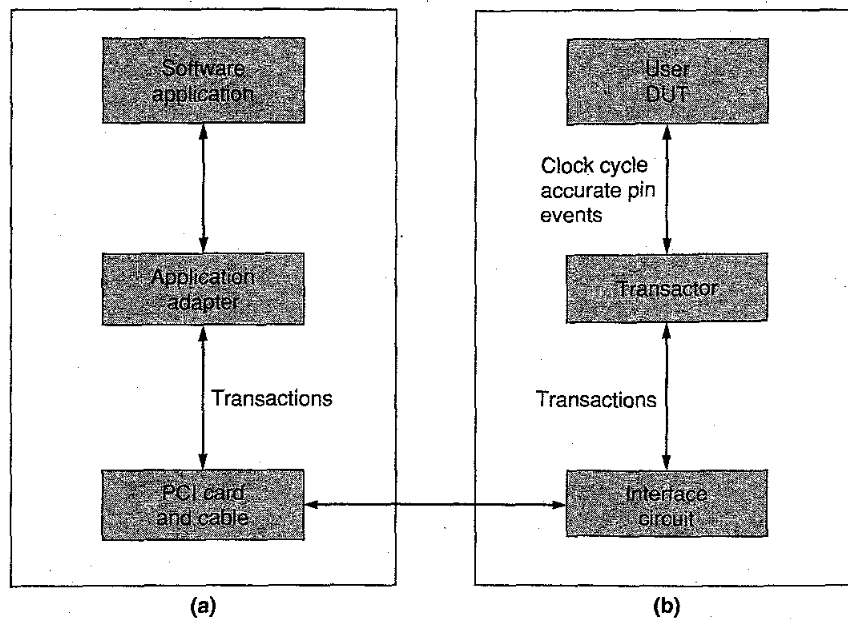


FIGURE 30.20 ■ The coverification flow between the workstation (a) and the emulator (b).

adapter is a utility package that converts raw signals into transactions by making calls to the C function library. It supports a verification environment that allows a C model to interact with an RTL model running on the emulator. The transfer of data across the host–emulator cable can be aided by buffering data in memory and transferring it as a block. This approach is preferable to the individual transfer of values from discrete memory locations in a file. Data buffering in arrays can be implemented in the same C modules that contain the TAPI driver calls for the emulator.

For the VLE system, the emulator system clock speed is set to 30 MHz. The same six multi-FPGA board connectors used for interfacing to an in-circuit emulation pod can also be used as an interface for coverification. The remaining connectors on the multi-FPGA boards can allow for direct access to logic analyzers for signal probing.

30.6.7 Parallel FPGA Compilation for the VLE System

Given the number of FPGAs in the VLE system, parallel compilation of the individual devices is a necessity. An FPGA compile server is used to distribute the numerous Xilinx XC4036XL compiles out to a number of available workstations that can perform the needed operations [9]. Unfinished compiles are held in a queue until compilation resources become available. Following design compilation, configuration bitstream information and status reports are returned to the server for subsequent transfer to the emulation system.

30.7 FUTURE TRENDS

Although FPGAs have played an important role in the development and success of commercial logic emulation hardware, current trends indicate a possibly reduced role for them in future emulation systems. Over the past few years, special-purpose custom logic processors have replaced FPGAs in a number of commercial emulation systems [26, 35]. Processor-based emulators generally contain a series of logic resources that perform a different Boolean function during every system clock cycle [10]. Data values, which are stored in on-chip RAM, are supplied to the logic resources every cycle via time-multiplexed on-chip routing resources. The per-cycle logic function definition and routing configuration information form instructions that are stored in on-chip instruction memory.

The depth of the memory constitutes the amount of multiplexing that can be performed both on the processor and in the interprocessor interconnect structure. Like multiplexed-wire FPGAs, interprocessor communications are time-sliced based on combinational logic dependencies so that processor pins are reused.

In general, the compile time for processor-based emulation is very fast compared to FPGA-based emulation. This disparity is a result of the assignment of intra-FPGA (processor) logic to interconnect resources. In multiplexed- and dedicated-wire emulation systems, internal FPGA logic and interconnect are *dedicated* to specific design resources. This has three implications:

1. For long combinational paths, each logic block and intra-FPGA wire is used only a small fraction of the time, effectively limiting system efficiency.
2. The dedicated assignment of signals to intra-FPGA wires is a problem of limited resource allocation. To significantly reduce compile time, a substantial increase in routing resources is needed relative to available logic to make FPGA routing linear time (a value of at least 20 percent is reported by Swartz et al. [33]). According to Rent's Rule, this disparity is likely to become worse as designs and FPGAs increase in size.
3. Because FPGA routers are unpredictable, it is impossible to determine both whether a device will route and what the per-FPGA (and hence global system) performance will be until all FPGAs have been successfully mapped.

In contrast, in processor-based emulator hardware, internal logic and routing structures are time-multiplexed. As a result, simpler routing structures with fixed memory to processor delays for all intra-processor paths are set. This, too, has implications:

1. Logic and interconnect resources are multiplexed over time to increase resource use efficiency per clock cycle.
2. The assignment of both inter- and intra-FPGA resources is a scheduling problem. Unlike search-based FPGA routing, scheduling algorithms

typically can be performed quickly and have runtimes largely proportional to circuit combinational depth.

3. The global system clock period is fixed by the architecture of the device, not by individual designs.

Specialized logic processors have other potential benefits. Specialized circuitry for signal probing and coverification transactions do not have to be fashioned out of generic FPGA logic, but rather can be customized to limit silicon overhead and optimize speed.

FPGA-based emulators do have some advantages. In some cases, they may provide more parallelism for certain designs that have shallow combinational depth. Rather than multiplexing logic resources, FPGAs can perform all logic operations simultaneously. The use of specialized logic processors in emulation introduces additional overhead for the emulation system provider. Because FPGAs typically use the newest silicon fabrication processes, specialized logic processors are likely to be at least one silicon generation behind the state of the art. Additionally, mapping tools for the logic processors must be developed and maintained by the emulation company rather than by the FPGA vendor. Recent trends indicate that despite these issues, the benefits of orders of magnitude faster compile time are driving emulation vendors in the direction of special-purpose logic processors.

Several developments in the design of FPGAs may swing this trend back in their favor. Recent FPGAs provide high-speed I/Os such as low-voltage differential signaling (LVDS) that support rapid I/O multiplexing. Additionally, the introduction of fixed cores, such as multipliers and microprocessors, may provide faster mapping and higher performance for emulation once they are integrated in the emulator compilation flow.

30.8 SUMMARY

FPGA-based logic emulation is a distinct example of a commercially successful reconfigurable computing application. A key aspect of its success has been the development of sophisticated software systems that can seamlessly map a large ASIC design to hundreds of FPGAs with minimal or no designer intervention. An important characteristic of most multi-FPGA emulators is the scheduling of both intra-FPGA computation and inter-FPGA communication in concert with a global system clock. The use of scheduling overcomes limited FPGA pin resources and takes advantage of signal dependencies, so that only portions of a design are active at a given time. Contemporary multi-FPGA logic emulators are used as both physical replacements in circuit and as coverification engines to accelerate design simulation. These supporting environments have advanced in recent years to include multiple asynchronous clock domains and support for incremental design changes.

Extended compile times are quickly becoming a dominant issue for FPGA-based emulators, and have motivated the development of fast FPGA compile

approaches. Although emulation systems with custom-designed logic processors have been developed, recent FPGA trends and faster compile approaches may spur renewed interest in FPGA-based emulation.

References

[1] A. Agarwal. *VirtualWires: A Technology for Massive Multi-FPGA Systems*, Mentor Graphics Corp., 2002.

[2] J. Babb, R. Tessier, M. Dahl, S. Hanano, D. Hoki, A. Agarwal. Logic emulation with virtual wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(6), June 1997.

[3] M. Butts. Future directions of dynamically reprogrammable systems. *IEEE Custom Integrated Circuits Conference*, May 1995.

[4] C. Chang, K. Kuusilinna, B. Richards, R. Broderson. Implementation of BEE: A real-time, large-scale hardware emulation engine. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 2003.

[5] S. Hauck, G. Borriello. Logic partition orderings for multi-FPGA systems. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 1995.

[6] S. Hauck, G. Borriello. An evaluation of bipartitioning techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(8), August 1997.

[7] S. Hauck. The role of FPGAs in reprogrammable systems. *Proceedings of the IEEE* 86(4), April 1998.

[8] S. Hauck, A. Agarwal. *Software Technologies for Reconfigurable Systems*, Technical report, Department of ECE, Northwestern University, 1996.

[9] IKOS Systems. *VirtuaLogic VLE Emulation System Manual*, 2001.

[10] D. Jones, D. Lewis. A time-multiplexed FPGA architecture for logic emulation. *IEEE Custom Integrated Circuits Conference*, May 1995.

[11] H. Krupnova, G. Saucier. FPGA-based emulation: Industrial and custom prototyping solutions. *International Conference on Field-Programmable Logic and Applications*, August 2000.

[12] M. Kudlugi, S. Hassoun, C. Selvidge, D. Pryor. A transaction-based unified simulation/emulation architecture for functional verification. *ACM/IEEE Design Automation Conference*, June 2001.

[13] M. Kudlugi, R. Tessier. Static scheduling and multidomain circuits for fast functional verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21(11), November 2002.

[14] J. Kumar. Prototyping the M68060 for concurrent verification. *IEEE Design and Test of Computers* 24(1), January 1997.

[15] I. Kuon, J. Rose. Measuring the gap between FPGAs and ASICs. *International Symposium on Field-Programmable Gate Arrays*, February 2006.

[16] Y. Kwon, C. Kyung. Performance-driven event-based synchronization for multi-FPGA simulation accelerator with event time-multiplexing bus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24(9), September 2005.

[17] B. Landman, R. Russo. On a pin versus block relationship for partitioning of logic graphs. *IEEE Transactions on Computers* C20(12), December 1971.

[18] C. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers* EC-10(2), September 1961.

[19] J. Li, C.-K Cheng. Routability improvement using dynamic interconnect architecture. *IEEE Workshop on FPGA-Based Custom Computing Machines*, April 1995.

[20] J. Marantz. Enhanced visibility and performance in functional verification by reconstruction. *ACM/IEEE Design Automation Conference*, June 1998.

[21] Mentor Graphics Corp. *VirtuaLogic Datasheet*, 2002.

[22] Mentor Graphics Corp. *VStation Datasheet*, 2004.

[23] Mentor Graphics. Emulation products web page: *http://www.mentor.com/emulation*, April 2006.

[24] Quickturn Design Systems. *System Realizer Data Sheet*, 1998.

[25] Quickturn Design Systems. *Mercury Data Sheet*, 1999.

[26] Quickturn Design Systems. *Cobalt Systems User Guide*, 2001.

[27] R. Ramaswamy, R. Tessier. The integration of SystemC and hardware-assisted verification. *International Conference on Field-Programmable Logic and Applications*, September 2002.

[28] K. Roy-Neogi, C. Sechen. Multiple FPGA partitioning with performance optimization. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 1995.

[29] M. Santarini. ASIC prototyping: Make versus buy. *EDN*, November 21, 2005.

[30] K. Shahookar, P. Mazumder. VLSI cell placement techniques. *ACM Computing Surveys* 23(1), June 1991.

[31] G. Snider, P. Kuekes, W. Culbertson, R. Carter, A. Berger, R. Amerson. The Teramac configurable compute engine. *International Conference on Field-Programmable Logic and Applications*, August 1995.

[32] H. Su, Y. Lin. A phase assignment method for virtual-wire-based hardware emulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(7), July 1997.

[33] J. S. Swartz, V. Betz, J. Rose. A fast routability-driven router for FPGAs. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 1998.

[34] R. Tessier, S. Jana. Incremental compilation for parallel verification systems. *IEEE Transactions on VLSI Systems* 10(5), October 2002.

[35] Tharas Systems. *Tharas Hammer Product Brief*, 2002.

[36] P. Tseng. Reconfigured engines REV simulation. *EE Times*, July 10, 2000.

[37] Xilinx, Inc. *Xilinx Foundation Tools User Guide*, 2002.