Edward V. Anderson (SBN 83148)
Edward C. Kwok (SBN 144302)
Georgia K. Van Zanten (SBN 116869)
Michael J. Halbert (SBN 173913)
Matthew J. Brigham (SBN 191428)
SKJERVEN MORRILL MACPHERSON LLP
25 Metro Drive, Suite 700
San Jose, California 95110
Phone: (408) 453-9200
Facsimile: (408) 453-7979

Attorneys for
IKOS SYSTEMS, INC. and
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN JOSE DIVISION

IKOS SYSTEMS, INC., a Delaware Corporation, and MASSACHUSETTS INSTITUTE OF TECHNOLOGY, a Massachusetts Corporation,

Plaintiffs,

vs.

AXIS SYSTEMS, INC., a Delaware Corporation,

Defendant.

Case No.: 01-21079 JW

SECOND AMENDED COMPLAINT FOR PATENT INFRINGEMENT AND DEMAND FOR TRIAL BY JURY

Plaintiffs IKOS Systems, Inc. and Massachusetts Institute of Technology state the following allegations for their second amended complaint of patent infringement against Defendant Axis Systems, Inc.

The Parties

1. Plaintiff IKOS Systems, Inc. ("IKOS") is a corporation organized under the laws of the state of Delaware, having an office and principal place of business at 79 Great Oaks Blvd., San Jose, California 95119.

2. Plaintiff Massachusetts Institute of Technology ("M.I.T.") is a corporation organized under the laws of the state of Massachusetts, located in Cambridge, Massachusetts 02138.

3. Defendant Axis Systems, Inc. ("Axis") is a corporation organized under the laws of the state of Delaware, having an office and principal place of business at 209 Java Dr., Sunnyvale, California 94089.

Jurisdiction And Venue

4. This action arises under the patent laws of the United States, Title 35 of the United States Code, including, but not limited to, 35 U.S.C. §§ 271 and 281.

5. Defendant Axis has and continues to commit, actively induce, and contribute to acts of patent infringement throughout the United States.

6. Subject matter jurisdiction is proper in this Court under 28 U.S.C. §§ 1331 and 1338(a).

7. Personal jurisdiction over Defendant Axis is proper under the United States Constitution and because Defendant Axis has its principal place of business in the state of California. Thus, this Court has personal jurisdiction over Defendant Axis.

8. Venue in this Court is pursuant to an Order from the District of Delaware, dated September 18, 2001, granting Defendant Axis's Motion to Transfer Venue to the Northern District of California Pursuant to 28 U.S.C. § 1404(a).

First Claim For Relief

9. Plaintiffs IKOS and M.I.T. incorporate the allegations stated in paragraphs 1 through 8 in this first claim for relief.

10. M.I.T. is the owner of United States Patent No. 5,596,742 ("the '742 patent") entitled "Virtual Interconnections For Reconfigurable Logic Systems." IKOS is the exclusive licensee of the '742 patent with full rights in and to the claims and causes of action involved in this suit. The '742 patent duly and legally issued on January 21, 1997. A true and correct copy of the '742 patent is attached to this Second Amended Complaint as Exhibit A.

11. Axis has been and is directly infringing the '742 patent, contributorily infringing the '742 patent, and inducing infringement of the '742 patent throughout the United States, by making, using, selling, offering for sale, and/or importing infringing products that are covered by the '742 patent, including, but not limited to, products designated as Xtreme, Xcite 2000, and Xcite 1000.

12. Axis's infringement of the '742 patent is willful, wanton, deliberate, without license, and with full knowledge of M.I.T. and IKOS's rights.

13. Unless restrained and enjoined by this Court, Axis will continue its acts of infringement and the resulting damage to IKOS and M.I.T. will be substantial, continuing, and irreparable.

WHEREFORE, IKOS and M.I.T. pray for judgment as set forth in the prayer for relief.

Second Claim For Relief

14. IKOS incorporates the allegations stated in paragraphs 1 through 13 in this second claim for relief.

15. IKOS is the owner of United States Patent No. 5,649,176 ("the '176 patent") entitled "Transition Analysis And Circuit Resynthesis Method And Device For Digital Circuit Modeling," with full rights in and to the claims and causes of action involved in this suit. The '176 patent duly and legally issued on July 15, 1997. A true and correct copy of the '176 patent is attached to this Second Amended Complaint as Exhibit B.

16. Axis has been and is directly infringing the '176 patent, contributorily infringing the '176 patent, and inducing infringement of the '176 patent throughout the United States, by making, using, selling, offering for sale, and/or importing infringing products that are covered by the '176 patent, including, but not limited to, products designated as Xtreme, Xcite 2000, and Xcite 1000.

17. Axis's infringement of the '176 patent is willful, wanton, deliberate, without license, and with full knowledge of IKOS's rights.

18. Unless restrained and enjoined by this Court, Axis will continue its acts of infringement and the resulting damage to IKOS will be substantial, continuing, and irreparable.

WHEREFORE, IKOS prays for judgment as set forth in the prayer for relief.

Third Claim For Relief

19. IKOS and M.I.T. incorporate the allegations stated in paragraphs 1 through 18 in this third claim for relief.

20. M.I.T. is the owner of United States Patent No. 5,761,484 ("the '484 patent") entitled "Virtual Interconnections For Reconfigurable Logic Systems." IKOS is the exclusive licensee of the '484 patent with full rights in and to the claims and causes of action involved in this suit. The '484 patent duly and legally issued on June 2, 1998. A true and correct copy of the '484 patent is attached to this Second Amended Complaint as Exhibit C.

21. Axis has been and is directly infringing the '484 patent, contributorily infringing the '484 patent, and inducing infringement of the '484 patent throughout the United States, by making, using, selling, offering for sale, and/or importing infringing products that are covered by the '484 patent, including, but not limited to, products designated as Xtreme, Xcite 2000, and Xcite 1000.

22. Axis's infringement of the '484 patent is willful, wanton, deliberate, without license, and with full knowledge of M.I.T. and IKOS's rights.

23. Unless restrained and enjoined by this Court, Axis will continue its acts of infringement and the resulting damage to IKOS and M.I.T. will be substantial, continuing, and irreparable.

WHEREFORE, IKOS prays for judgment as set forth in the prayer for relief.

Prayer For Relief

IKOS and M.I.T. request that this Court enter judgment that:

24. Axis, its officers, directors, employees, agents, licensees, servants, successors, and assigns, and any and all persons acting in privity or in concert with them, be preliminary and permanently restrained and enjoined from further infringement of the '742 patent, '176 patent, and '484 patent (35 U.S.C. § 283);

25. Damages be awarded to IKOS and M.I.T. against Axis in an amount adequate to compensate IKOS and M.I.T. for Axis's infringement of the '742 patent and '484 patent (35 U.S.C. § 284);

26. Damages be awarded to IKOS against Axis in an amount adequate to compensate IKOS for Axis's infringement of IKOS's '176 patent (35 U.S.C. § 284);

27. Damages be increased three times the amount found or assessed due to Axis's willful infringement (35 U.S.C. §284);

28. This is an exceptional case and IKOS and M.I.T. be awarded their costs, expenses, and disbursements in this action, including reasonable attorneys' fees (35 U.S.C. § 285);

29. IKOS and M.I.T. be awarded their costs, expenses, and disbursements in this action (Fed. R. Civ. P. 54(d));

30. IKOS and M.I.T. be awarded interest on the amount of damages found, including pre-judgment and post-judgment interest (35 U.S.C. §284); and

31. IKOS and M.I.T. be awarded such other and further relief as this Court may deem just and proper.

Jury Demand

Plaintiffs IKOS and M.I.T. demand trial by jury on all issues so triable.

DATED: January 9, 2002

SKJERVEN MORRILL MACPHERSON LLP

By ______________________________
Matthew J. Brigham
Attorneys for Plaintiffs
IKOS SYSTEMS, INC. and MASSACHUSETTS INSTITUTE OF TECHNOLOGY

833545 v1

EXHIBIT A

US005596742A

United States Patent [19]

Agarwal et al.

[11] **Patent Number:** **5,596,742**

[45] **Date of Patent:** **Jan. 21, 1997**

[54] **VIRTUAL INTERCONNECTIONS FOR RECONFIGURABLE LOGIC SYSTEMS**

[75] Inventors: **Anant Agarwal**, Framingham, Mass.; **Jonathan Babb**, Ringgold, Ga.; **Russell Tessier**, Cambridge, Mass.

[73] Assignee: **Massachusetts Institute of Technology**, Cambridge, Mass.

[21] Appl. No.: **42,151**

[22] Filed: **Apr. 2, 1993**

[51] **Int. Cl.^{6}** **G06F 17/50**

[52] **U.S. Cl.** **395/500**; 364/488; 364/489

[58] **Field of Search** 364/232.3, 949.9, 364/488, 489, 490, 491, 578; 395/500; 326/37, 38, 39, 41, 47, 101

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,495,590	1/1985	Mitchell, Jr.	364/716
4,506,341	3/1985	Kalter et al.	364/786
4,697,241	9/1987	Lavi	364/488
4,901,259	2/1990	Watkins	364/578
5,109,353	4/1992	Sample et al.	362/578
5,283,900	2/1994	Frankel et al.	395/700
5,442,306	8/1995	Woo	326/39
5,444,394	8/1995	Watson et al.	326/45
5,452,231	9/1995	Butts et al.	364/489
5,473,266	12/1995	Ahanin et al.	326/41
5,475,830	12/1995	Chen et al.	395/500
5,483,178	1/1996	Costello et al.	326/41
5,485,103	1/1996	Pedersen et al.	326/41

FOREIGN PATENT DOCUMENTS

0410502A2	1/1991	European Pat. Off. .
90/04233	4/1990	WIPO .

OTHER PUBLICATIONS

Khan et al., "FPGA Architectures for ASIC Hardware Emulators," 1993 ASIC Conference & Exhibit, pp. 336–340.
Walters, "Reprogrammable Hardware Emulation for ASICs Makes Thorough Design Verification Practical," Compcon Spring '90, pp. 484–486.

Deiss, Stephen, R., "Connectionism without the Connections," *IEEE*, (1994), pp. 1217–1221.

Dominguez–Castro, R., et al., "Architectures and Building Blocks for CMOS VLSI Analog Neural Programmable Optimizers," *IEEE*, (1992), pp. 1525–1528.

Fornaciari, William, et al., "An Automatic VLSI Implementation of Hopfield ANNs,", *IEEE*, (1995), pp. 499–502.

Bailey, Jim, et al., "Why VLSI Implementations of Associative VLCNs Require Connection Multiplexing," pp. II –173 –II –180.

Dally, "Virtual–Channel Flow Control" *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, No. 2 (Mar. 1992) pp. 194–205.

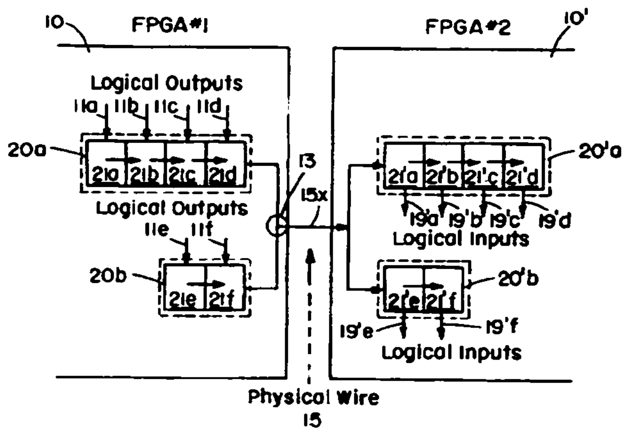
(List continued on next page.)

Primary Examiner—Kevin J. Teska
Assistant Examiner—Leigh Marie Garbowski
Attorney, Agent, or Firm—Hamilton, Brook, Smith & Reynolds, P.C.

[57] **ABSTRACT**

A compilation technique overcomes device pin limitations using virtual interconnections. Virtual interconnections overcome pin limitations by intelligently multiplexing each physical wire among multiple logical wires and pipelining these connections at the maximum clocking frequency. Virtual interconnections increase usable bandwidth and relax the absolute limits imposed on gate utilization in logic emulation systems employing Field Programmable Gate Arrays (FPGAs). A "softwire" compiler utilizes static routing and relies on minimal hardware support. The technique can be applied to any topology and FPGA device.

42 Claims, 6 Drawing Sheets



OTHER PUBLICATIONS

Burskey, "Using Antifuse Programming For Gate-Arrayed Density and Flexibility, An FPGAs Family Also Delivers Masked-Array Performance. FPGas Mirror Masked Gate--Array Architecture" *Electronic Design*, (Nov. 21, 1991) pp. 63-70.

Kermani et al., "Virtual Cut-Through: A New Computer Communication Switching Technique", *Computer Networks*, vol. 3, (Oct. 1979) pp. 267-286.

Murgai et al., "Logic Synthesis for Programmable Gate Arrays" *IEEE 27th ACM/IEEE Design Automation Conference*, (1990) pp. 620-625.

Singh et al, "Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed" *IEEE 1991 Custom Integrated Circuits Conference* (1991) pp. 6.1.1—6.1.6.

Witienberg, R. C., "Three Newcomers Stir Up Hardware Accelerator Market," no further information.

Bursky, D., "Fast simulator expands to mimic 4 million gates," *Electronic Design*, p. 23, (Dec. 1986).

Bertsekas, D., et al., *Data Networks*, pp. 8 -14, 91 -95 and 403 -405 (Prentice Hall 1987).

Rose, Jonathan, et al, "Architecture of Field-Programmable Gate Arrays: The Effect of logic Block Functionality on Area Efficiency," IEEE Journal of Solid State Circuits, No. 5, Oct. 1990, pp. 1217-1225.

Babb, Jonatha, et al., "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators," Proceedings IEEE Workshop on FPGAs for Custom Computing Machines, Apr. 5, 1993, pp. 142-151.

Hey, Anthony, "Supercomputing with Transputers -Past, Present and Future," *Computer Architecture News*, vol. 18, No. 3, Sep. 1990, pp. 479-489.

Agrawal, Om, "Field Programmable Gate Arrays (FPGAs) Provide ASIC System Designers Control Of Their Design Destiny," Electro Conference Record, vol. 15, May 1990, pp. 353-361.

Van Den Bout, D. E., et al., "IEEE Design and Test of Computers," *IEEE Computer Society*, pp. 21-30, Sep. 1992.

XILINX: The Programmable Gate Array Design Handbook, 1986, pp. 1-9 to 1-14.

Wei, Yen-Chuen, et al., "Multiple-Level Partitioning: An Application to the Very Large-Scale Hardware Simulator," *IEEE*, 26(5) : 706-716, (May 1991).

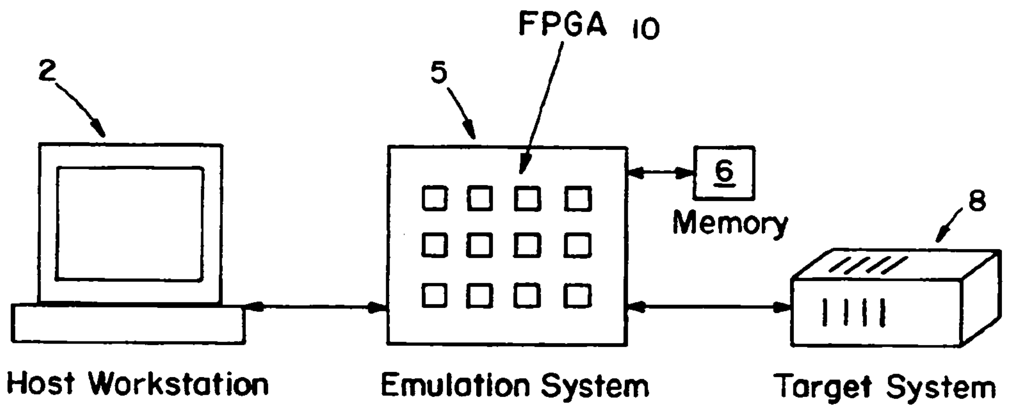


FIG. 1 (Prior Art)

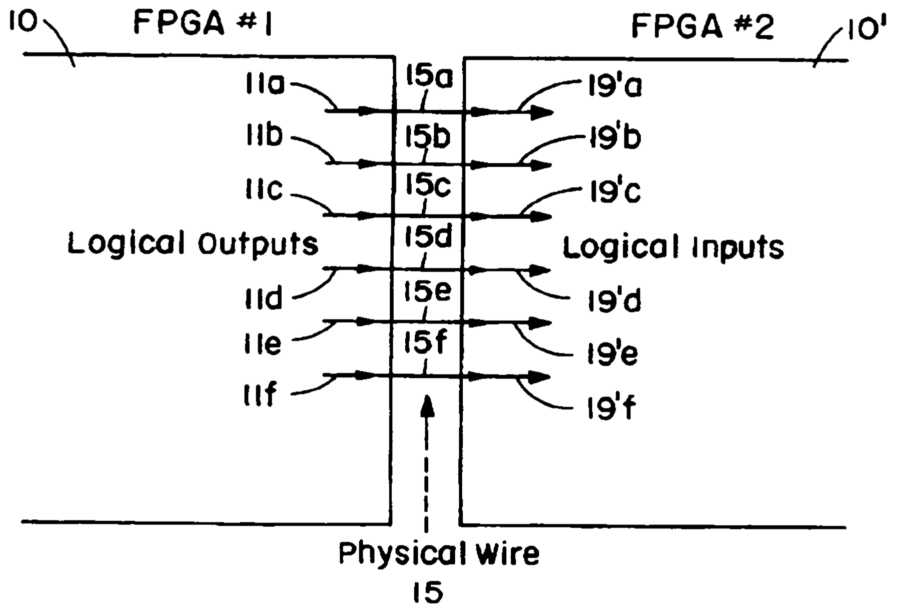


FIG. 2 (Prior Art)

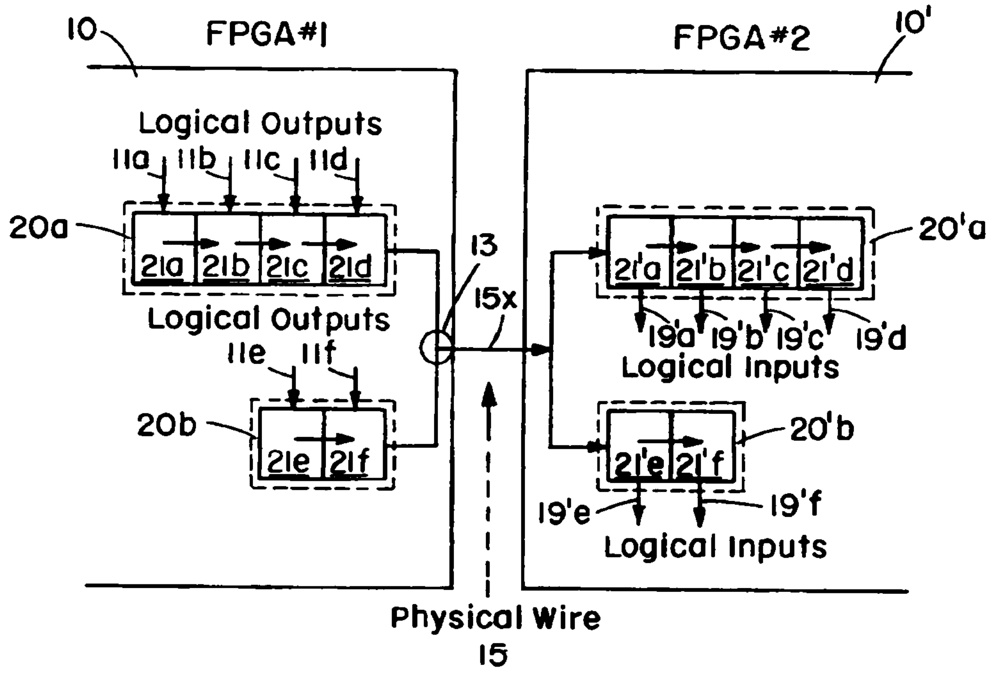


FIG. 3

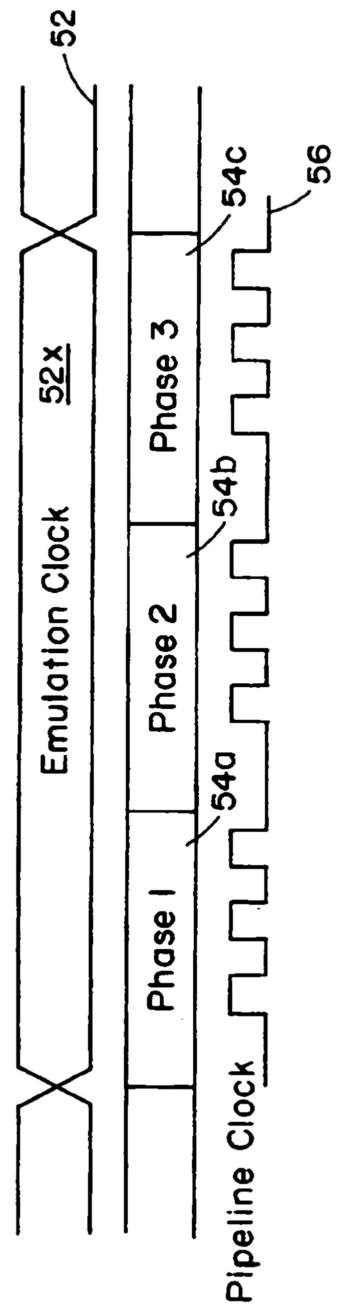

52
Emulation Clock
52x
Phase 1
Phase 2
Phase 3
54a
54b
54c
56
Pipeline Clock

FIG. 4

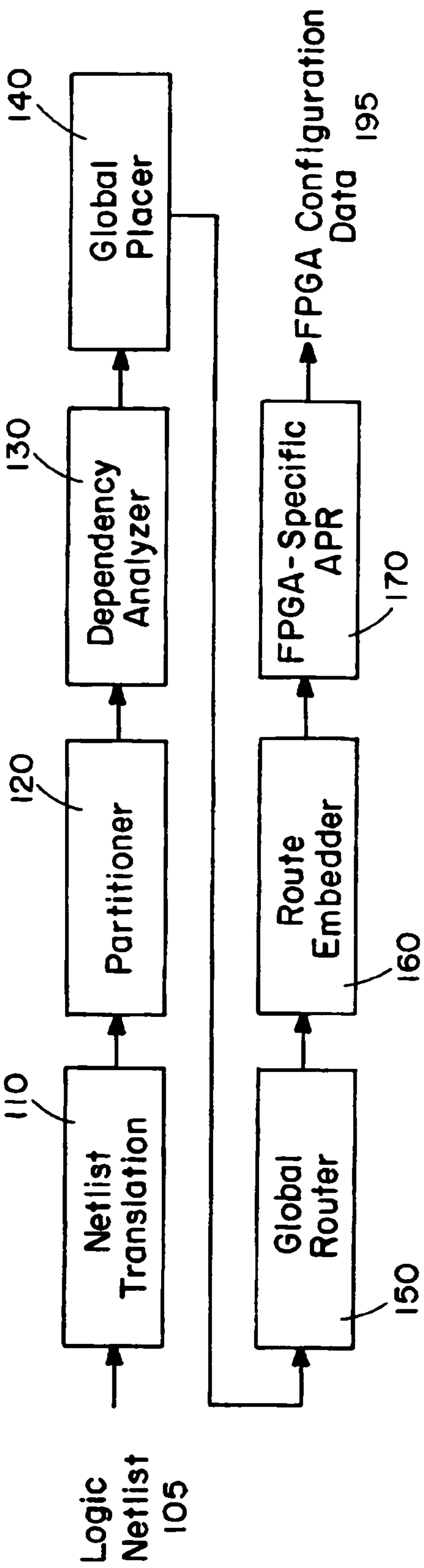
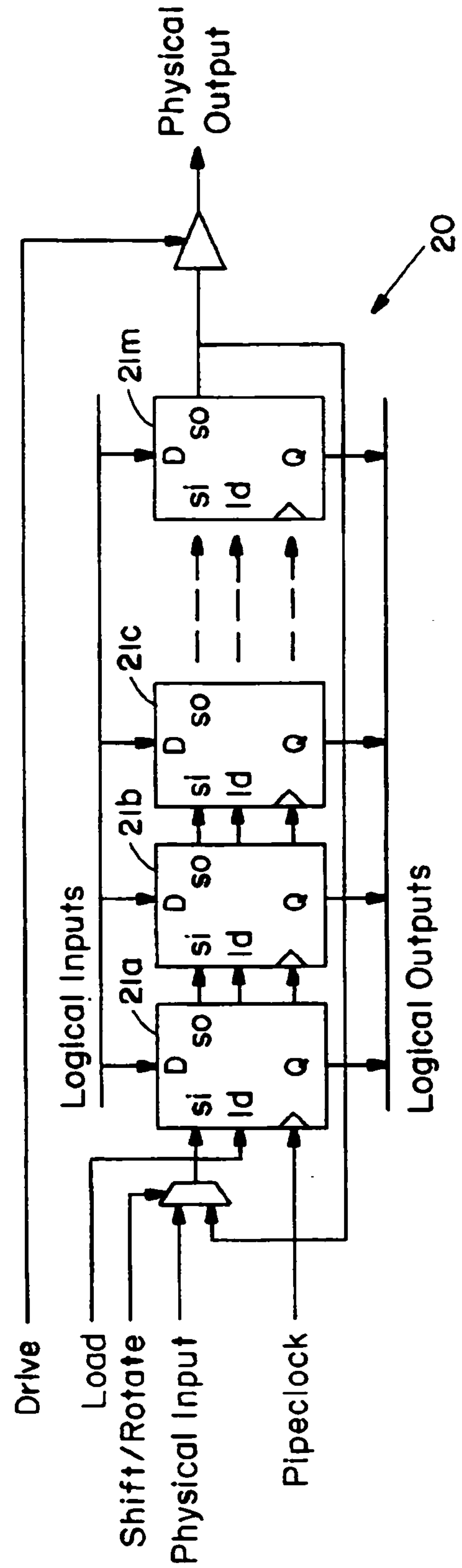
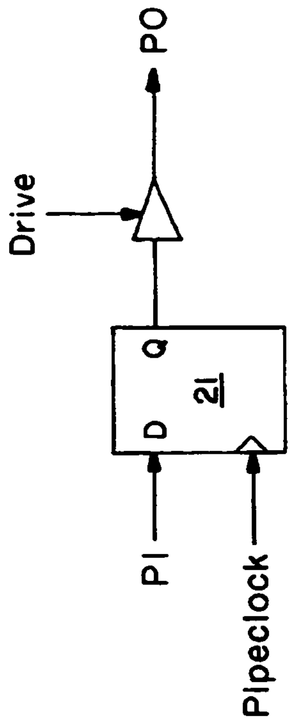

Logic Netlist 105
110
Netlist Translation
120
Partitioner
130
Dependency Analyzer
140
Global Placer
Global Router
150
Route Embedder
160
FPGA-Specific APR
170
FPGA Configuration Data
195

FIG. 5



Physical Output
20
21m
21c
21b
21a
si
D
so
ld
Q
Logical Inputs
Logical Outputs
Drive
Load
Shift/Rotate
Physical Input
Pipeclock

FIG. 6



PO
Drive
Q
D
21
PI
Pipeclock

FIG. 7

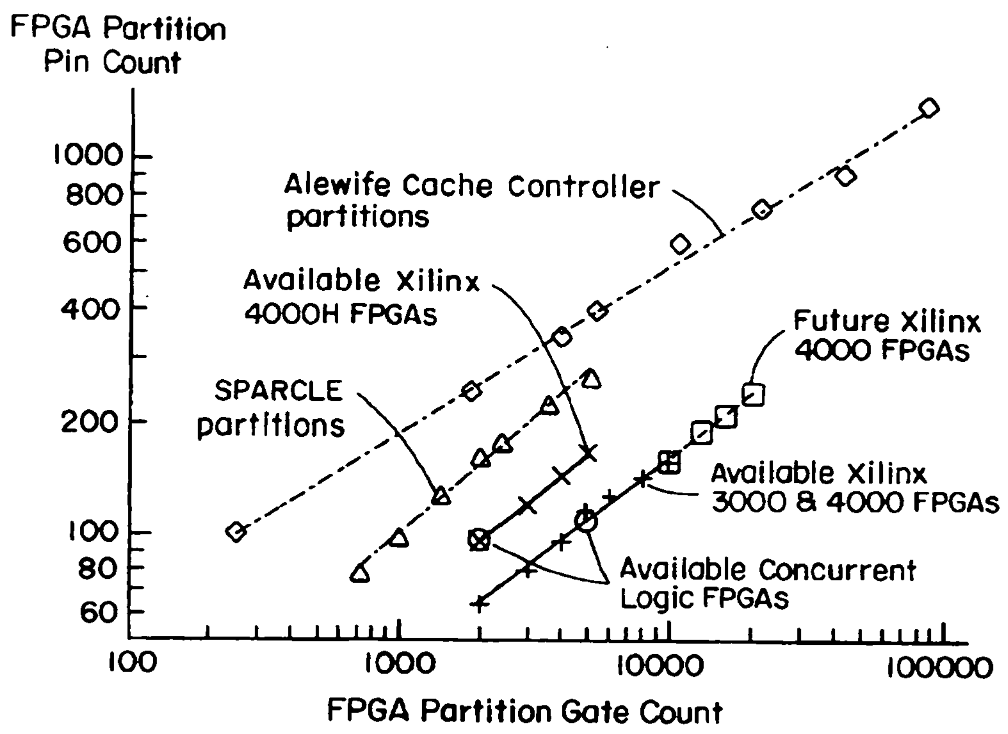

FPGA Partition Pin Count
1000
800
600
400
200
100
80
60
100
1000
10000
100000
FPGA Partition Gate Count
Alewife Cache Controller partitions
Available Xilinx 4000H FPGAs
SPARCLE partitions
Future Xilinx 4000 FPGAs
Available Xilinx 3000 & 4000 FPGAs
Available Concurrent Logic FPGAs

FIG. 8

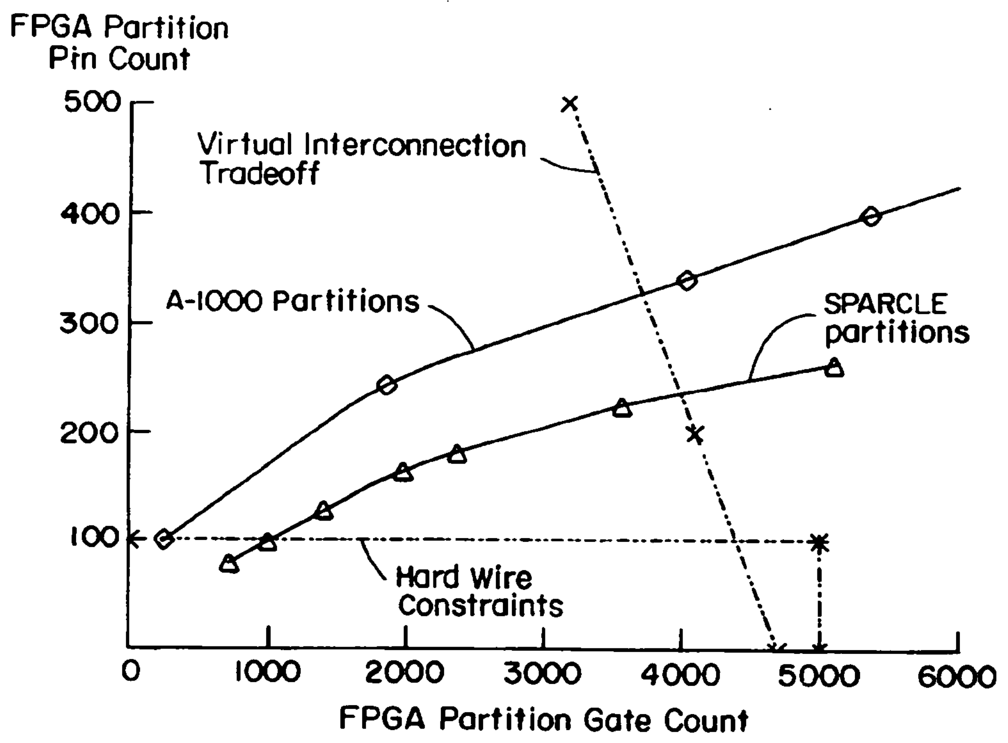

FPGA Partition Pin Count
500
400
300
200
100
0
1000
2000
3000
4000
5000
6000
FPGA Partition Gate Count
Virtual Interconnection Tradeoff
A-1000 Partitions
SPARCLE partitions
Hard Wire Constraints

FIG. 9

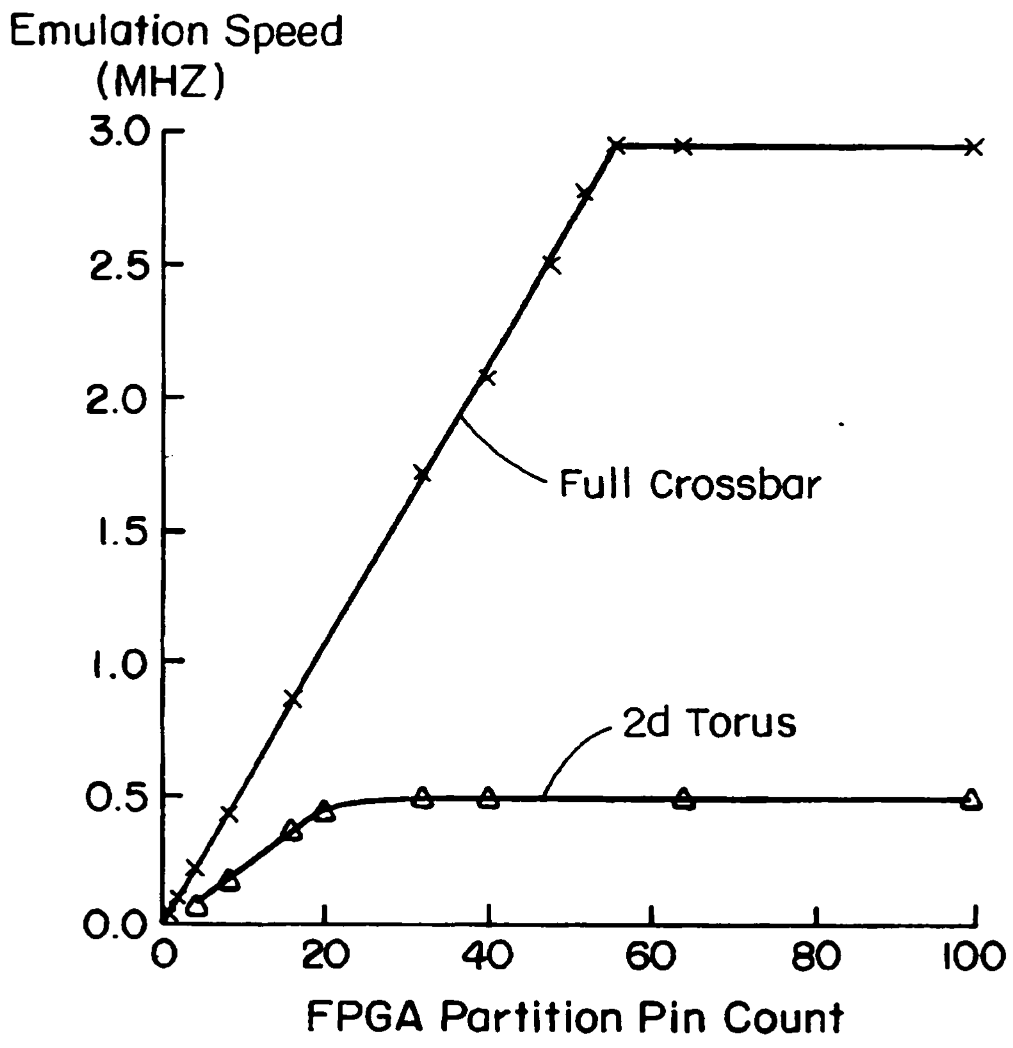

Emulation Speed
(MHZ)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
FPGA Partition Pin Count
Full Crossbar
2d Torus

FIG. 10

VIRTUAL INTERCONNECTIONS FOR RECONFIGURABLE LOGIC SYSTEMS

GOVERNMENT SUPPORT

The invention described herein was supported in whole or in part by Navy Contract No. N00014-91-J-1698 and Grant No. MIP-9012773 from the National Science Foundation (NSF).

BACKGROUND OF THE INVENTION

Field Programmable Gate Array (FPGA) based logic emulators are capable of emulating complex logic designs at clock speeds four to six orders of magnitude faster than even an accelerated software simulator. Once configured, an FPGA-based emulator is a heterogeneous network of special purpose processors, each FPGA processor being specifically designed to cooperatively execute a partition of the overall simulated circuit. As parallel processors, these emulators are characterized by their interconnection topology (network), target FPGA (processor), and supporting software (compiler). The interconnection topology describes the arrangement of FPGA devices and routing resources (i.e. full crossbar, two dimension mesh, etc). Important target FPGA properties include gate count (computational resources), pin count (communication resources), and mapping efficiency. Supporting software is extensive, combining netlist translators, logic optimizers, technology mappers, global and FPGA-specific partitioners, placers, and routers.

FPGA-based logic emulation systems have been developed for design complexity ranging from several thousand to several million gates. Typically, the software for these system is considered the most complex component. Emulation systems have been developed that interconnect FPGAs in a two-dimensional mesh and in a partial crossbar topology. In addition, a hierarchical approach to interconnection has been developed. Another approach uses a combination of nearest neighbor and crossbar interconnections. Logic partitions are typically hardwired to FPGAs following partition placement.

Statically routed networks can be used whenever communication can be predetermined. Static refers to the fact that all data movement can be determined and optimized at compile-time. This mechanism has been used in scheduling real-time communication in a multiprocessor environment. Other related uses of static routing include FPGA-based systolic arrays and in the very large simulation subsystem (VLSS), a massively parallel simulation engine which uses time-division multiplexing to stagger logic evaluation.

SUMMARY OF THE INVENTION

Existing FPGA-based logic emulators suffer from limited inter-chip communication bandwidth, resulting in low gate utilization (10 to 20 percent). This resource imbalance increases the number of chips needed to emulate a particular logic design and thereby decreases emulation speed, because signals must cross more chip boundaries, and increases system cost. Prior art emulators only use a fraction of potential communication bandwidth because the prior art emulators dedicate each FPGA pin (physical wire) to a single emulated signal (logical wire). These logical wires are not active simultaneously and are only switched at emulation clock speeds.

A preferred embodiment of the invention presents a compilation technique to overcome device pin limitations using virtual interconnections. This method can be applied to any topology and FPGA device, although some benefit substantially more than others. Although a preferred embodiment of the invention focuses on logic emulation, the technique of virtual interconnections is also applicable to other areas of reconfigurable logic.

Virtual interconnections overcome pin limitations by intelligently multiplexing each physical wire among multiple logical wires and pipelining these connections at the maximum clocking frequency of the FPGA. A virtual interconnection represents a connection from a logical output on one FPGA to a logical input on another FPGA. Virtual interconnections not only increase usable bandwidth, but also relax the absolute limits imposed on gate utilization. The resulting improvement in bandwidth reduces the need for global interconnect, allowing effective use of low dimension inter-chip connections (such as nearest-neighbor). In a preferred embodiment, a "softwire" compiler utilizes static routing and relies on minimal hardware support. Virtual interconnections can increase FPGA gate utilization beyond 80% without a significant slowdown in emulation speed.

In a preferred embodiment of the invention, a FPGA logic emulation system comprises a plurality of FPGA chips. Each chip has a number of pins for communicating signals between chips. There are also interchip connections between the FPGA pins. In addition, a software or hardware compiler programs each FPGA chip to emulate a partition of an emulated circuit with interconnections between partitions of the emulated circuit being provided through FPGA pins and interchip connections. A partition of the emulated circuit has a number of interconnections to other partitions that exceed the number of pins on the FPGA chip. The chip is programmed to communicate through virtual interconnections in a time-multiplexed fashion through the pins.

The FPGA chips may comprise gates that are programmed to serve as a multiplexer for communicating through the virtual interconnections. Alternatively, the FPGA chips may comprise hardwire multiplexers that are separate from the programmable gates. The pins of the FPGA chips may be directly connected to pins of other FPGA chips, where routing of signals between the chips is through intermediate FPGAs. The FPGA chips may also be programmed to operate in phases within an emulation clock cycle with interchip communications being performed within each phase.

The compiler may optimize partition selection and phase division of an emulated circuit based on interpartition dependencies.

Data may also be accessed from memory elements external to the FPGAs during each phase by multiplexing the data on the virtual interconnections.

In a preferred embodiment of the invention, the FPGA chips comprise an array of gates, shift registers, and several multiplexers. The gates are programmable to emulate a logic circuit. Each shift register receives plural outputs from the program gate array and communicates the outputs through a single pin in a multiplexed fashion. Some fraction of the gates in an FPGA chip may be programmed to serve as shift registers and multiplexer for communicating through virtual connections.

In a preferred embodiment of the invention, a compiler programs a FPGA logic emulation system using a means for partitioning an emulated logic circuit into partitions and means for programming each FPGA to emulate a partition of an emulated circuit. The partitions are to be programmed into individual FPGA chips. The compiler produces virtual interconnections between partitions of the emulated circuit that correspond to one or more common pins with signals along the virtual interconnections being time-multiplexed through the common pin.

The compiler may comprise a dependency analyzer and means for dividing an emulation clock into phases, the phase division being a function of partition dependencies and memory assignments. During the phases, program logic functions are performed and signals are transmitted between the FPGA chips. The compiler may also comprise a router for programming the FPGA chips to route signals between chips through intermediate chips. In particular, the routed signals are virtual interconnections.

Results from compiling two complex designs, the 18K gate SPARCLE microprocessor and the 86K gate Alewife Cache Compiler (A-1000), show that the use of virtual interconnections decreases FPGA chip count by a factor of 3 for SPARCLE and 10 for the A-1000, assuming a crossbar interconnect. With virtual interconnections, a two dimensional torus interconnect can be used for only a small increase in chip count (17 percent for the A-1000 and 0 percent for SPARCLE). Without virtual interconnections, the cost of a replacing the full crossbar with a torus interconnect is over 300 percent for SPARCLE, and virtually impossible for the A-1000. Emulation speeds are comparable with the no virtual interconnections case, ranging from 2 to 8 MHZ for SPARCLE and 1 to 3 MHZ for the A-1000. Neither design was bandwidth limited, but rather constrained by its critical path. With virtual interconnections, use of a lower dimension network reduces emulation speed proportional to the network diameter; a factor of 2 for SPARCLE and 6 for the A-1000 on a two dimensional torus.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other features of the invention, including various novel details of construction and combinations of parts, will now be more particularly described with reference to the accompanying drawings and pointed out in the claims. It will be understood that the particular virtual interconnection technique embodying the invention is shown by way of illustration only and not as a limitation of the invention. The principles and features of this invention may be employed in varied and numerous embodiments without departing from the scope of the invention.

FIG. 1 is a block diagram of a typical prior art logic emulation system.

FIG. 2 is a block diagram of a prior art hardwire interconnect system between Field Programmable Gate Arrays (FPGA) 10 of FIG. 1.

FIG. 3 is a block diagram of a virtual interconnection interconnect system between FPGAs 10 of FIG. 1.

FIG. 4 is a graphical representation of an emulation phase clocking scheme.

FIG. 5 is a flowchart of a preferred software compiler.

FIG. 6 is a block diagram of a preferred shift register or shift loop architecture.

FIG. 7 is a block diagram of the intermediate hop, single bit, pipeline stage of FIG. 6.

FIG. 8 is a graph illustrating pin count as a function of FPGA partition size.

FIG. 9 is a graph illustrating a determination of optimal partition size.

FIG. 10 is a graph illustrating emulation speed vs. pin count for a torus and a crossbar configuration.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

FIG. 1 is a block diagram of a typical prior art logic emulation system 5. The performance of the system 5 is achieved by partitioning a logic design, described by a netlist, across an interconnected array of FPGAs 10. This array is connected to a host workstation 2 which is capable of downloading design configurations, and is directly wired into the target system 8 for the logic design. Memory elements 6 may also be connected to the array of FPGAs 10. The netlist partition on each FPGA (hereinafter FPGA partition), configured directly into logic circuitry, can then be executed at hardware speeds.

In existing architectures, shown in FIG. 2, both the logic configuration and the network connectivity remain fixed for the duration of the emulation. FIGS. 2 shows an example of six logical wires 11*a–f*, 19'*a–f* allocated to six physical wires 15*a–f*. Each emulated gate is mapped to one FPGA equivalent gate and each emulated signal is allocated to one FPGA pin. Thus, for a partition to be feasible, the partition gate and pin requirements must be no greater that the available FPGA resources. This constraint yields the following possible scenarios for each FPGA partition:

1. Gate limited: no unused gates, but some unused pins.
2. Pin limited: no unused pins, but some unused gates.
3. Not limited: unused FPGA pins and gates.
4. Balanced: no unused pins or gates.

For mapping typical circuits onto available FPGA devices, partitions are predominately pin limited; all available gates cannot be utilized due to a lack of pin resources to support them. Low utilization of gate resources increases both the number of FPGAs 10 needed for emulation and the time required to emulate a particular design. Pin limits set a hard upper bound on the maximum usable gate count any FPGA gate size can provide. This discrepancy will only get worse as technology scales; trends (and geometry) indicate that available gate counts are increasing faster than available pin counts.

In a preferred embodiment of the invention, shown in FIG. 3, virtual interconnections are used to overcome pin limitations in FPGA-based logic emulators. FIG. 3 shows an example of six logical wires 11*a–f* sharing a single physical wire 15*x*. The physical wire 15*x* is multiplexed 13 between two pipelined shift loops 20*a*, 20*b*, which are discussed in detail below. Pipelining refers to signal streams in a particular phase and multiplexing refers to signals across phases. A virtual interconnection represents a connection between a logical output 11*a* on one FPGA 10 and a logical input 19'*a* on another FPGA 10'. Established via a pipelined, statically routed communication network, these virtual interconnections increase available off-chip communication bandwidth by multiplexing 13 the use of FPGA pin resources (physical wires) 15 among multiple emulation signals (logical wires).

Virtual interconnections effectively relax pin limitations. Although low pin counts may decrease emulation speed, there is not a hard pin constraint that must be enforced. Emulation speed can be increased if there is a large enough reduction in system size. The gate overhead of using virtual interconnections is low, comprising gates that are not utilized in the purely hardwired implementation. Furthermore, the flexibility of virtual interconnections allows the emulation architecture to be balanced for each logic design application.

One to one allocation of emulation signals (logical wires) 11, 19 to FPGA pins (physical wires) 15 does not exploit available off chip bandwidth because emulation clock frequencies are one or two orders of magnitude lower than the potential clocking frequency of the FPGA technology, and all logical wires 11, 19 are not active simultaneously.

By pipelining and multiplexing physical wires 15, virtual interconnections are created to increase usable bandwidth. By clocking physical wires 15 at the maximum frequency of the FPGA technology, several logical connections can share the same physical resource.

In a logic design, evaluation flows from system inputs to system outputs. In a synchronous design with no combinatorial loops, this flow can be represented as a directed acyclic graph. Thus, through intelligent dependency analysis of the underlying logic circuit, logical values between FPGA partitions need to only be transmitted once. Furthermore, because circuit communication is inherently static, communication patterns repeat in a predictable fashion.

In a preferred embodiment of the invention, virtual interconnections are supported with a "softwire" compiler. This compiler analyzes logic signal dependencies and statically schedules and routes FPGA communication. These results are then used to construct (in the FPGA technology) a statically routed network. This hardware consists of a sequencer and shift loops. The sequencer is a distributed finite state machine. The sequencer establishes virtual connections between FPGAs by strobing logical wires in a predetermined order into special shift registers 21, the shift loops 20. The shift loops 20 serve as multiplexers 13 and are described in detail below. Shift loops 20 are then alternately connected to physical wires 15 according to the predetermined schedule established by the sequencer.

The use of virtual interconnections is limited to synchronous logic. Any asynchronous signals must still be "hard-wired" to dedicated FPGA pins. This limitation is imposed by the inability to statically determine dependencies in asynchronous loops. Furthermore, each combinational loop (such as a flip-flop) in a synchronous design is completely contained in a single FPGA partition. For simplicity and clarity of description, it is assumed that the emulated logic has a single global clock.

In a preferred embodiment of the invention, virtual interconnections are implemented in the context of a complete emulation software system, independent of target FPGA device and interconnect topology. While this embodiment focuses primarily on software, the ultimate goal of the invention is a low-cost, reconfigurable emulation system.

In a preferred embodiment, the signals are routed through each FPGA. This embodiment avoids the use of a crossbar. By routing the signals through each FPGA, speed is increased because there are no long wires connecting the FPGAs to the crossbar.

FIG. 4 graphically represents an emulation phase clocking scheme. The emulation clock period 52x is the clock period of the logic design being emulated. This clock is broken into evaluation phases (54*a*, 54*b*, 54*c*) to accumulate multiplexing. Multiple phases are required because the combinational logic between flip-flops in the emulated design may be split across multiple FPGA partitions and multiplexing of vertical wires prevents direct pass of all signals through the partitions. The phases permit a single pin to send different logical signals on every phase. Within a phase 54, evaluation is accomplished within each partition, and the results are then communicated to other FPGA partitions. Although three phases are illustrated per emulation period, it will be understood that more or less phases can be employed.

At the beginning of the phase 54, logical outputs of each FPGA partition are determined by the logical inputs in input shift loops. At the end of the phase 54, outputs are then sent to other FPGA partitions with pipelined shift loops and intermediate hop stages. As illustrated in FIG. 4, these pipelines are clocked with a pipeline clock 56 at the maximum frequency of the FPGA. After all phases 54 within an emulation clock period 52x are complete, the emulation clock 52 is ticked to clock all flip-flops of the target circuit.

The input to the softwire compiler consists of a netlist 105 of the logic design to be emulated, target FPGA device characteristics, and interconnect topology. The compiler then produces a configuration bitstream that can be downloaded into the emulator. FIG. 5 is a flowchart of the compilation steps. Briefly, these steps include translation and mapping of the netlist to the target FPGA technology (step 110), partitioning the netlist (step 120), placing the partitions into interconnect topology (steps 130, 140), routing the inter-node communication paths (steps 150, 160), and finally FPGA-specific automated placement and routing (APR) (step 170).

The input netlist 105 to be emulated is usually generated with a hardware description language or schematic capture program. This netlist 105 must be translated and mapped (step 110) to a library of FPGA macros. It is important to perform this operation before partitioning so that partition gate counts accurately reflect the characteristics of the target FPGAs. Logic optimization tools can also be used at this point to optimize the netlist for the target architecture (considering the system as one large FPGA).

After mapping (step 110) the netlist to the target architecture, the netlist must be partitioned (step 120) into logic blocks that can fit into the target FPGA. With only hardwires, each partition must have both fewer gates and fewer pins than the target device. With virtual interconnections, the total gate count (logic gates and virtual wiring overhead) must be no greater than the target FPGA gate count. A preferred embodiment uses the Concept Silicon partitioner manufactured by INCA, Inc. This partitioner performs K-way partitioning with min-cut and clustering techniques to minimize partition pin counts.

Because a combinatorial signal may pass through several FPGA partitions during an emulated clock cycle, all signals will not be ready to schedule at the same time. A preferred embodiment solves this problem by only scheduling a partition output once all the inputs it depends upon are scheduled (step 130). An output depends on an input if a change in that input can change the output. To determine input to output dependencies, the logic netlist is analyzed, backtracing from partition outputs to determine which partition inputs they depend upon. In backtracing, it is assumed that all outputs depend on all inputs for gate library parts, and no outputs depend on any inputs for latch (or register) library parts. If there are no combinatorial loops that cross partition boundaries, this analysis produces a directed acyclic graph, the signal flow graph (SFG), to be used by the global router.

Following logic partitioning, individual FPGA partitions must be placed into specific FPGAs (step 140). An ideal placement minimizes system communication, thus requiring fewer virtual interconnection cycles to transfer information. A preferred embodiment first makes a random placement followed by cost-reduction swaps, and then further optimize with simulated annealing.

During global routing (150), each logical wire is scheduled to a phase, and assigned a pipeline time slot (corresponding to one cycle of the pipeline clock in that phase on a physical wire). Before scheduling, the criticality of each logical wire is determined (based on the signal flow graph produced by dependency analysis). In each phase, the router first determines the schedulable wires. A wire is schedulable if all wires it depends upon have been scheduled in previous phases. The router than uses shortest path analysis with a cost function based on pin utilization to route as many schedulable signals as possible, routing the most critical signals first. Any schedulable signals which cannot be routed are delayed to the next phase.

Once routing is completed, appropriately-sized shift loops and associated logic are added to each partition to complete the internal FPGA hardware description (step 160). At this point, there is one netlist for each FPGA. These netlists are then be processed with the vendor-specific FPGA place and route software (step 170) to produce configuration bit-streams (step 195).

Technically, there is no required hardware support for implementation of virtual interconnections (unless one considers re-designing an FPGA optimized for virtual wiring). The necessary "hardware" is compiled directly into configuration for the FPGA device. Thus, any existing FPGA-based logic emulation system can take advantage of virtual wiring. Virtual interconnections can be used to store and retrieve data from memory elements external to the FPGAs by multiplexing the data on the virtual interconnections during a phase. There are many possible ways to implement the hardware support for virtual interconnections. A preferred embodiment employs a simple and efficient implementation. The additional logic to support virtual interconnections can be composed entirely of shift loops and a small amount of phase control logic.

FIG. 6 is a block diagram of a preferred shift loop architecture. A shift loop 20 is a circular, loadable shift register with enabled shift in and shift out ports. Each shift register 21 is capable of performing one or more of the operations of load, store, shift, drive, or rotate. The Load operation strobes logical outputs into the shift loop. The Store operation drives logical inputs from the shift loop. The Shift operation shifts data from a physical input into the shift loop. The Drive operation drives a physical output with the last bit of the shift loop. The Rotate operation rotates bits in the shift loop. In a preferred embodiment, all outputs loaded into a shift loop 20 must have the same final destination FPGA. As described above, a logical output can be strobed once all corresponding depend inputs have been stored. The purpose of rotation is to preserve inputs which have reached their final destination and to eliminate the need for empty gaps in the pipeline when shift loop lengths do not exactly match phase cycle counts. In this way, a signal may be rotated from the shift loop output back to the shift loop input to wait for an appropriate phase. Note that in this implementation the store operation cannot be disabled.

Shift loops 20 can be re-scheduled to perform multiple output operations. However, because the internal latches being emulated depend on the logical inputs, inputs need to be stored until the tick of the emulation clock.

For networks where multiple hops are required (i.e. a mesh), one bit shift registers 21 that always shift and sometimes drive are used for intermediate stages. FIG. 7 is a block diagram of the intermediate hop pipeline stage. These stages are chained together, one per FPGA hop, to build a pipeline connecting the output shift loop on the source FPGA 10 with the input shift loop on the destination FPGA 10'.

The phase control logic is the basic run-time kernel in a preferred embodiment. This kernel is a sequencer that controls the phase enable and strobe (or load) lines, the pipeline clock, and the emulation clock. The phase enable lines are used to enable shift loop to FPGA pin connections. The phase strobe lines strobe the shift on the correct phases. This logic is generated with a state machine specifically optimized for a given phase specification.

EXPERIMENTAL RESULTS

The system compiler described above was implemented by developing a dependency analyzer, global placer, global router, and using the InCA partitioner. Except for the partitioner, which can take hours to optimize a complex design, running times on a SPARC 2 workstation were usually 1 to 15 minutes for each stage.

To evaluate the costs and benefits of virtual interconnections, two complex designs were compiled, SPARCLE and the A-1000. SPARCLE is an 18K gate SPARC microprocessor enhanced with multiprocessing features. The Alewife compiler and memory management unit (A-1000) is an 86K gate cache compiler for the Alewife Multiprocessor, a distributed shared memory machine being designed at the Massachusetts Institute of Technology. For target FPGAs, the Xilinx 3000 and 4000 series (including the new 4000H series) and the Concurrent Logic Cli6000 series were considered. This analysis does not include the final FPGA-specific APR stage; a 50 percent APR mapping efficiency for both architectures is assumed.

In the following analysis, the FPGA gate costs of virtual interconnections based on the Concurrent Logic CLi6000 series FPGA were estimated. The phase control logic was assumed to be 300 gates (after mapping). Virtual interconnection overhead can be measured in terms of shift loops. In the Cli6000, a bit stage shift register takes 1 of 3136 cells in the 5K gate part (C_s=3 mapped gates). Thus, total required shift register bits for a partition is then equal to the number of inputs. When routing in a mesh or torus, intermediate hops cost 1 bit per hop. The gate overhead is then C_s×S, where C_s is the cost of a shift register bit, and S is the number of bits. S is determined by the number of logical inputs, V_i, and M_p, the number of times a physical wire p is multiplexed (this takes into account the shift loop tristate driver and the intermediate hop bits). Gate overhead is then approximately:

$$Gate_{vw}=C_s\times(V_i+\Sigma_p M_p).$$

Storage of logical outputs is not counted because logical outputs can be overlapped with logical inputs.

Before compiling the two test designs, their communication requirements were compared to the available FPGA technologies. For this comparison, each design was partitioned for various gate counts and the pin requirements were measured. FIG. 8 shows the resulting curves, plotted on a log-log scale. Note that the partition gate count is scaled to represent mapping inefficiency.

Both design curves and the technology curves fit Rent's Rule, a rule of thumb used for estimating communication requirement in random logic. Rent's Rule can be stated as:

$$pins_2/pins_1=(gates_2/gate_1)^b,$$

where $pins_2$, $gates_2$ refer to a partition, and $pins_1$, $gates_1$ refer to a sub-partition, and b is constant between 0.4 and 0.7. Table 1 shows the resulting constants. For the technology curve, a constant of 0.5 roughly corresponds to the area versus perimeter for the FPGA die. The lower the constant, the more locality there is within the circuit. Thus, the A-1000 has more locality than SPARCLE, although it has more total communication requirements. As FIG. 8 illustrates, both SPARCLE and the A-1000 will be pin-limited for any choice of FPGA size. In hardwired designs with pin-limited partition sizes, usable gate count is determined solely by available pin resources. For example, a 5000 gate FPGA with 100 pins can only utilize 1000 SPARCLE gates or 250 A-1000 gates.

TABLE 1

Rent's Rule Parameter (slope of log-log curve)

FPGA Technology	SPARCLE	A-1000
0.50	0.06	0.44

Next, both designs were compiled for a two dimensional torus and a full crossbar interconnect of 5000 gate, 100 pin FPGAs, 50 percent mapping efficiency. Table 2 shows the results for both hard wires and virtual interconnections. Compiling the A-1000 to a torus, hardwires only, was not practical with the partitioning software. The gate utilizations obtained for the hardwired cases agree with reports in the literature on designs of similar complexity. To understand the tradeoffs involved, the hardwires pin/gate constraint and the virtual interconnections pin/gate tradeoff curve were plotted against the partition curves for the two designs (FIG. 9). The intersection of the partition curves and the wire curves gives the optimal partition and sizes. This graph shows how virtual interconnections add the flexibility of trading gate resources for pin resources.

TABLE 2

Number of 5K Gates, 100 Pin FPG
As Required for Logic Emulation

	Hardwires Only		Virtual interconnections Wires Only	
Design	2-D Torus	Full Crossbar	2-D Torus	Full Crossbar
Sparcle (18K gates)	>100 (<7%)	31 (23%)	9 (80%)	9 (80%)
A-1000 (86K gates)	Not Practical	>400 (<10%)	49 (71%)	42 (83%)

Number of FPGAs (Average Usable Gate Utilization)

Emulation clock cycle time T_E is determined by:

1. Communication delay per hop, t_c;
2. Length of longest path in dependency graph L;
3. Total FPGA gate delay along longest path T_L;
4. Sum of pipeline cycles across all phases, n;
5. Network diameter, D (D=1 for crossbar); and
6. Average network distance, k_d (k_d=1 for crossbar).

The total number of phases and pipeline cycles in each phase are directly related to physical wire contention and the combinatorial path that passes through the largest number of partitions. If the emulation is latency dominated, then the optimal number of phases is L, and the pipeline cycles per phase should be no greater than D, giving:

$$n=L\times D.$$

If the emulation is bandwidth dominated, then the total pipeline cycles (summed over all phases) is at least:

$$n=MAX_p[(Vi_p/Pi_p)]$$

where Vi_p and Pi_p are the number of virtual and physical wires for FPGA partition p. If there are hot spots in the network (not possible with a crossbar), the bandwidth dominated delay will be higher. Emulation speeds for SPARCLE and the A-1000 were both latency dominated.

Based on CLi6000 specifications, it was assumed that T_L=250 ns and t_c=20 ns (based on a 50 MHZ clock). A computation-only delay component, and a communication-only delay component were considered. This dichotomy is used to give a lower and upper bound on emulation speed.

The computation-only delay component is given by:

$$T_P=T_L+t_c\times n,$$

where n=0 for the hardwired case.

The communication-only delay component is given by:

$$T_c=t_c\times L.$$

Table 3 shows the resulting emulation speeds for virtual and hardwires for the crossbar topology. The emulation clock range given is based on the sum and minimum of the two components (lower and upper bounds). When the use of virtual interconnections allows a design to be partitioned across fewer FPGAs, L is decreased, decreasing T_c. However, the pipeline stages will increase T_p by t_c per pipeline cycle.

TABLE 3

Emulation Clock Speed Comparison

		Hardwire Only	Virtual Wire Only
SPARCLE	Longest Path	9 hops	6 hops
	Computation Only Delay	250 ns	370 ns
	Communication Only Delay	180 ns	120 ns
	Emulation Clock Range	2.3–5.6 MHZ	2.0–8.3 MHz
A-1000	Longest Path	27 hops	17 hops
	Computation Only Delay	250 ns	590 ns
	Communication Only Delay	540 ns	340 ns
	Emulation Clock Range	1.3–4.0 MHz	1.1–2.9 MHz

In Table 3, the virtual interconnection emulation clock was determined solely by the length of the longest path; the communication was limited by latency, not bandwidth. To determine what happens when the design becomes bandwidth limited, the pin count was varied and the resulting emulation clock (based on T_c) was recorded for both a crossbar and torus topology. FIG. 10 shows the results for the A-1000. The knee of the curve is where the latency switches from bandwidth dominated to latency dominated. The torus is slower because it has a larger diameter, D. However, the torus moves out of the latency dominated region sooner because it exploits locality; several short wires can be routed during the time of a single long wire. Note that this analysis assumes the crossbar can be clocked as fast as the torus; the increase in emulation speed obtained with the crossbar is lower if t_c is adjusted accordingly.

With virtual interconnections, neither designs was bandwidth limited, but rather limited by its respective critical paths. As shown in FIG. 10, the A-1000 needs only about 20 pins per FPGA to run at the maximum emulation frequency. While this allows the use of lower pin count (and thus cheaper) FPGAs, another option is to trade this surplus bandwidth for speed. This tradeoff is accomplished by hardwiring logical wires at both ends of the critical paths. Critical wires can be hardwired until there is no more surplus bandwidth, thus fully utilizing both gate and pin resources. For designs on the 100 pin FPGAs, hardwiring reduces the longest critical path from 6 to 3 for SPARCLE and from 17 to 15 for the A-1000.

Virtual interconnections allow maximum utilization of FPGA gate resources at emulation speeds competitive with existing hardwired techniques. This technique is independent of topology. Virtual interconnections allow the use of less complex topologies, such as a torus instead of a crossbar, in cases where such a topology was not practical otherwise.

Using timing and/or locality sensitive partitioning with virtual interconnections has potential for reducing the required number of routing sub-cycles. Communication bandwidth can be further increased with pipeline compaction, a technique for overlapping the start and end of long virtual paths with shorter paths traveling in the same direction. A more robust implementation of virtual interconnections replaces the global barrier imposed by routing phases with a finer granularity of communication scheduling, possible overlapping computation and communication as well.

Using the information gained from dependency analysis, one can now predict which portions of the design are active during which parts of the emulation clock cycle. If the FPGA device supports fast partial reconfiguration, this information can be used to implement virtual logic via invocation of hardware subroutines. An even more ambitious direction is event-driven emulation—only send signals which change, only activate (configure) logic when it is needed.

EQUIVALENTS

Those skilled in the art will know, or be able to ascertain using no more than routine experimentation, many equivalents to the specific embodiments of the invention described herein.

These and all other equivalents are intended to be encompassed by the following claims.

The invention claimed is:

1. A reconfigurable electronic system comprising:

a plurality of reprogrammable logic modules, each logic module having a plurality of pins for communicating signals between logic modules;

inter-module connections between pins of different logic modules; and

a configurer to configure each logic module to define a partition of a specified target circuit with interconnections between the partitions of the target circuit being provided through pins and inter-module connections, a partition of the configured target circuit having a number of interconnections to other partitions that exceeds the number of pins on the logic module and the logic module being configured to communicate through virtual interconnections in a time-multiplexed fashion through at least one pin, the inter-module communications including interconnections which extend through intermediate reconfigurable logic modules.

2. A system as claimed in claim 1 wherein the configurer configures a logic module to form a multiplexer for communicating through virtual interconnections.

3. A system as claimed in claim 1 wherein pins of logic modules are directly connected to pins of other logic modules and routing of signals between the logic modules is through intermediate logic modules.

4. A system as claimed in claim 1 wherein the logic modules comprise hardwired multiplexers.

5. A system as claimed in claim 1 wherein the logic modules are configured to operate in phases within a target clock period with inter-module communications being performed within each phase.

6. A system as claimed in claim 5 wherein the configurer optimizes logic module selection and phase division of the target circuit based on interpartition dependencies.

7. A system as claimed in claim 6 wherein the target clock period dictates the maximum rate at which signal lines of the target circuit change value and wherein each target clock period comprises a plurality of system clock periods which dictate the maximum rate at which signals in the electronic system change value.

8. The system as claimed in claim 7 wherein each system clock period is scheduled to either perform a computation or carry a communication signal during a particular target clock period.

9. The system as claimed in claim 6 wherein a physical intermodule pin carries a plurality of target system signals during a target system clock period, each target system signal being carried during a system clock period.

10. A system as claimed in claim 1 wherein data is accessed from memory elements external to the logic modules.

11. The system as claimed in claim 10 wherein memory data is multiplexed on virtual interconnections.

12. A system as claimed in claim 1 wherein the logic modules are Field Programmable Gate Arrays (FPGAs).

13. A system as claimed in claim 1 wherein the system is an emulation system for emulating the target circuit.

14. A system as claimed in claim 1 wherein each logic module comprises an array of interconnected programmable logic cells.

15. A system as claimed in claim 1 wherein each module is a single chip.

16. A system as claimed in claim 1 wherein logic modules are configured to include pins dedicated to individual signals.

17. A system as claimed in claim 16 wherein an individual signal is on a critical signal path.

18. A system as claimed in claim 16, including asynchronous logic hardwired to dedicated pins of the logic modules.

19. A system as claimed in claim 16 wherein each dedicated pin is a surplus pin not configured by the configurer as a virtual interconnection.

20. A system as claimed in claim 1 wherein the configurer comprises a partitioner that partitions the target logic circuit, each partition being configured into a respective logic module.

21. A system as claimed in claim 20 further comprising a dependency analyzer and a divider, a target clock period being divided into phases during which program logic functions are performed and signals are transmitted between the logic modules, the phase division being a function of partition dependencies and memory assignments.

22. A system as claimed in claim 20 further comprising a router that configures the logic modules to route signals between logic modules through intermediate logic modules.

23. A system as claimed in claim 1 wherein the configurer provides pipeline compaction by overlapping a first virtual path with a plurality of second virtual paths traveling in the same direction.

24. A system as claimed in claim 1 wherein the configurer configures virtual logic by reconfiguring a portion of the logic module during periods of inactivity for the portion of the logic module.

25. A system as claimed in claim 1 wherein the configuration of the logic modules is event driven such that pins only communicate signals which have changed in value.

26. A logic system as claimed in claim 1 wherein at least one intermodule logic module includes a register corresponding to a signal being routed.

27. A compiler for programming a reconfigurable electronic system comprising:

a partitioner that partitions a target logic circuit into partitions to be configured into individual logic modules;

a configurer to configure each logic module to create a partition of the target circuit with virtual interconnections between partitions of the target circuit corresponding to at least one common pin with signals along the virtual interconnections being time-multiplexed through common pins; and

a router to configure the logic modules to route signals between logic modules through intermediate logic modules.

28. A compiler as claimed in claim 27 further comprising a dependency analyzer and a divider that divides a target clock period into phases during which program logic functions are performed and signals are transmitted between the logic modules, the phase division being a function of partition dependencies and memory assignments.

29. A compiler as claimed in claim 27 wherein the logic modules are Field Programmable Gate Arrays (FPGAs).

30. A compiler as claimed in claim 27 wherein the system is a logic emulation system and the target circuit is being emulated by the logic emulation system.

31. A compiler as claimed in claim 27 wherein the configurer configures intermediate logic modules to form at least one topology from the group consisting of crossbar, mesh and torus.

32. A method of compiling a reconfigurable electronic system, comprising the steps of:

partitioning a target circuit into a plurality of partitions, each partition to be configured into an individual logic module having a plurality of pins;

configuring each logic module to create a partition of the target circuit with virtual interconnections between partitions corresponding to at least one common pin with signals along the virtual interconnections being time-multiplexed through the at least one common pin; and

configuring the logic modules to route signals between logic modules through intermediate logic modules.

33. A method as claimed in claim 32 further comprising the step of dividing a first clock period which dictates the maximum rate at which signal lines within the target circuit change value into phases during which program logic functions are performed and signals are transmitted between logic modules.

34. A reconfigurable electronic system comprising:

a plurality of reprogrammable logic modules, each logic module having a plurality of pins for communicating signals between logic modules;

inter-module connections between pins of different logic modules; and

a configurer to configure each logic module to define a partition of a specified target circuit with interconnections between the partitions of the target circuit being provided through pins and inter-module connections, a partition of the configured target circuit having a number of interconnections to other partitions that exceeds the number of pins on the logic module and the logic module being configured to communicate through virtual interconnections in a time-multiplexed fashion through at least one pin, the electronic system including dedicated pins for providing a predetermined signal.

35. A system as claimed in claim 34 wherein the predetermined signal is on a critical signal path.

36. A system as claimed in claim 34, including asynchronous logic hardwired to dedicated pins of the logic modules.

37. A system as claimed in claim 34 wherein each dedicated pin is a surplus pin not configured by the configurer as a virtual interconnection.

38. A system as claimed in claim 34 wherein the logic modules are configured to operate in phases within a target clock period with inter-module communications being performed within each phase.

39. A system as claimed in claim 34 wherein data is accessed from memory elements external to the logic modules.

40. A logic system as claimed in claim 34 wherein the logic modules are Field Programmable Gate Arrays (FPGAs).

41. A system as claimed in claim 34 wherein the system is an emulation system for emulating the target circuit.

42. A system as claimed in claim 34 wherein each module is a single chip.

* * * * *

EXHIBIT B

US005649176A

United States Patent [19]
Selvidge et al.

[11] Patent Number: 5,649,176
[45] Date of Patent: Jul. 15, 1997

[54] TRANSITION ANALYSIS AND CIRCUIT RESYNTHESIS METHOD AND DEVICE FOR DIGITAL CIRCUIT MODELING

[75] Inventors: Charles W. Selvidge, Charlestown; Matthew L. Dahl, Cambridge, both of Mass.

[73] Assignee: Virtual Machine Works, Inc., Cambridge, Mass.

[21] Appl. No.: 513,605

[22] Filed: Aug. 10, 1995

[51] Int. Cl.^{6} G06F 1/12
[52] U.S. Cl. 395/551; 364/489
[58] Field of Search 395/551, 500; 364/488, 489, 490, 491

[56] References Cited

U.S. PATENT DOCUMENTS

4,450,560	5/1984	Conner	371/25
4,697,241	9/1987	Lavi	364/488
5,180,937	1/1993	Laird et al.	327/276
5,420,544	5/1995	Ishibashi	331/11
5,428,626	6/1995	Frisch et al.	364/488 X

FOREIGN PATENT DOCUMENTS

0 453 171 A2	10/1991	European Pat. Off.	G06F 1/04
2 180 382	3/1987	United Kingdom	H03K 19/00

OTHER PUBLICATIONS

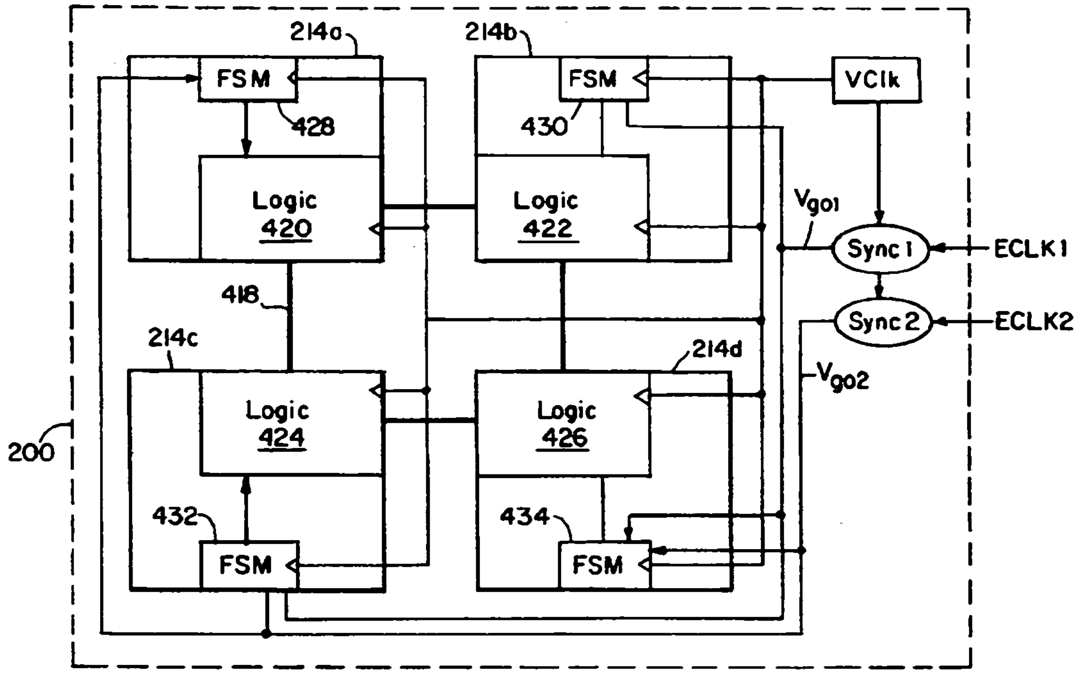
Laird, D., et al., "Delay Compensator," *LSI Logic Corp.*, pp. 1–8, (Aug. 1990).

Primary Examiner—Thomas M. Heckler
Attorney, Agent, or Firm—Hamilton, Brook, Smith & Reynolds, P.C.

[57] ABSTRACT

A method of configuring a configurable logic system, including a single or multi-FPGA network, is disclosed in which an internal clock signal is defined that has a higher frequency than timing signals the system receives from the environment in which it is operating. The frequency can be at least ten times higher than a frequency of the environmental timing signals. The logic system is configured to have a controller that coordinates operation of its logic operation in response to the internal clock signal and environmental timing signals. Specifically, the controller is a finite state machine that provides control signals to sequential logic elements such as flip-flops. The logic elements are clocked by the internal clock signal. In the past, emulation or simulation devices, for example, operated in response to timing signals from the environment. A new internal clock signal, invisible to the environment, rather than the timing signals is used to control the internal operations of the devices. Additionally, a specific set of transformations are disclosed that enable the conversion of a digital circuit design with an arbitrary clocking methodology into a single clock synchronous circuit.

50 Claims, 14 Drawing Sheets



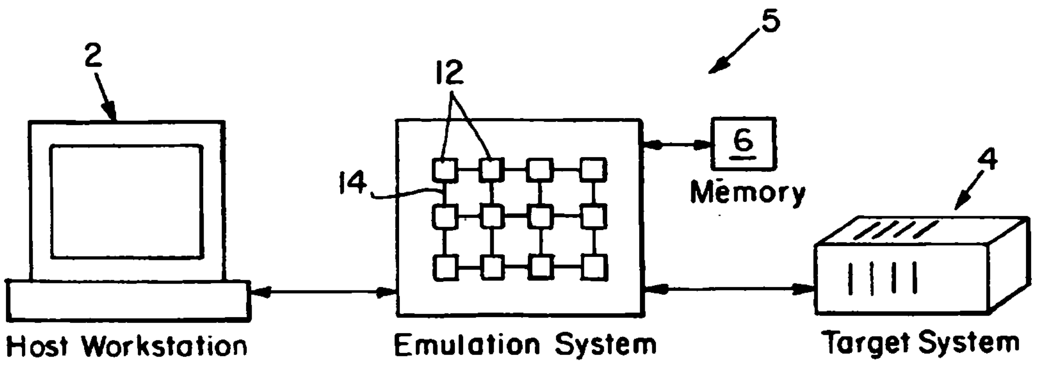


FIG. 1
(Prior Art)

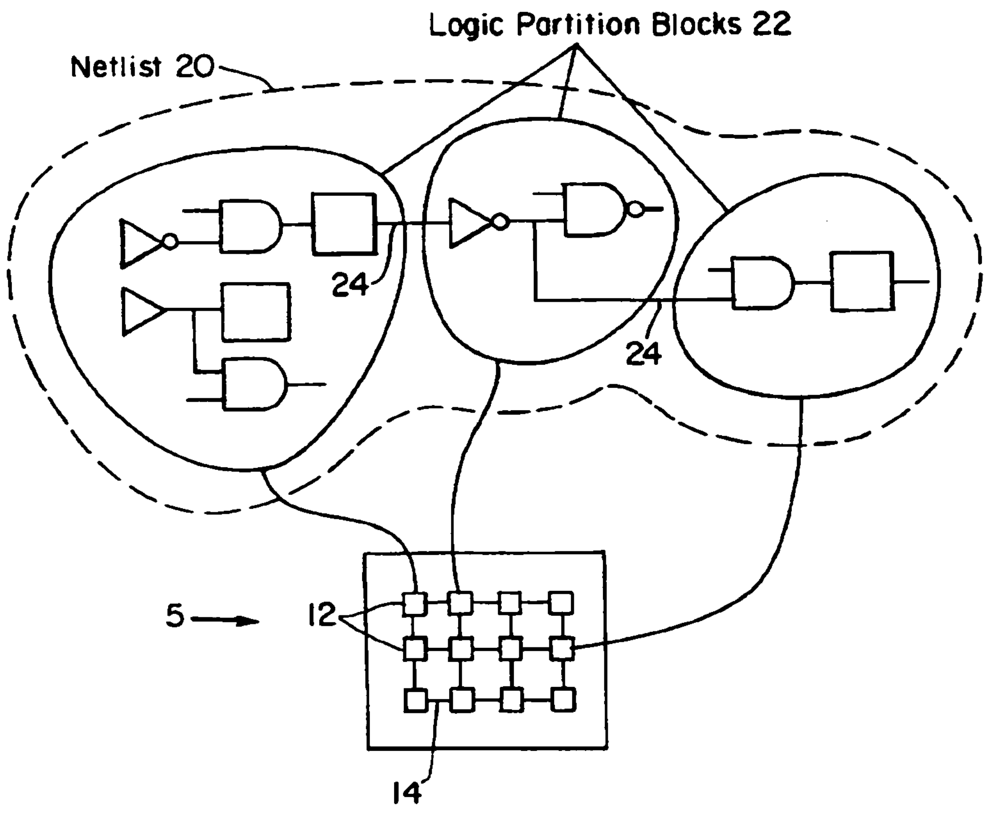


FIG. 2

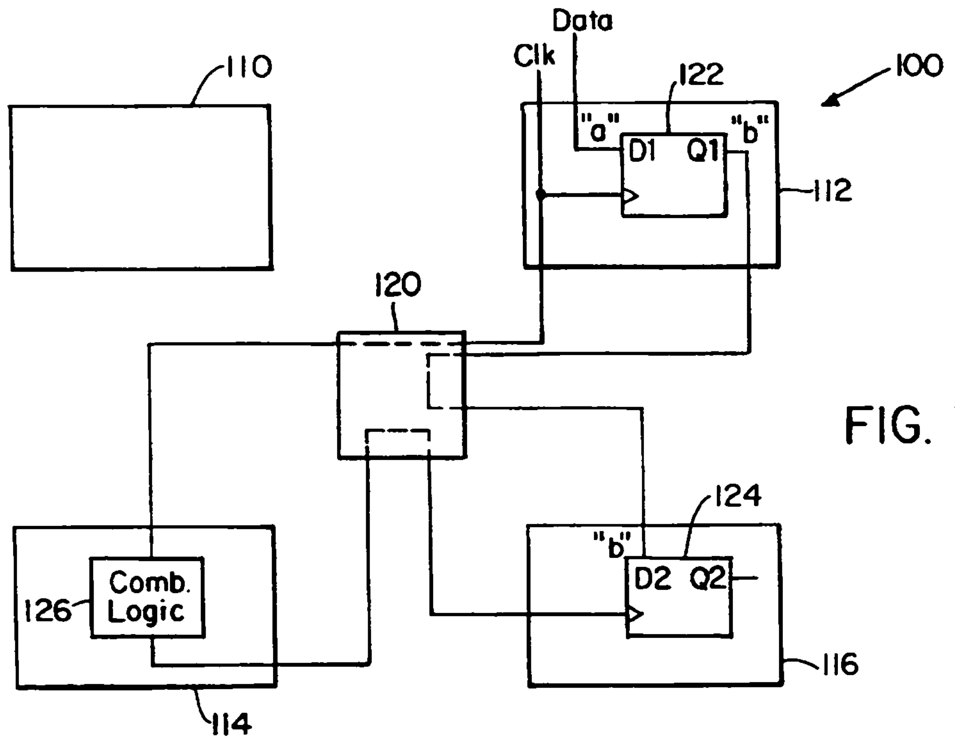

Data
Clk
110
122
100
"a"
"b"
D1
Q1
112
120
124
"b"
D2
Q2
126
Comb. Logic
114
116

FIG. 3

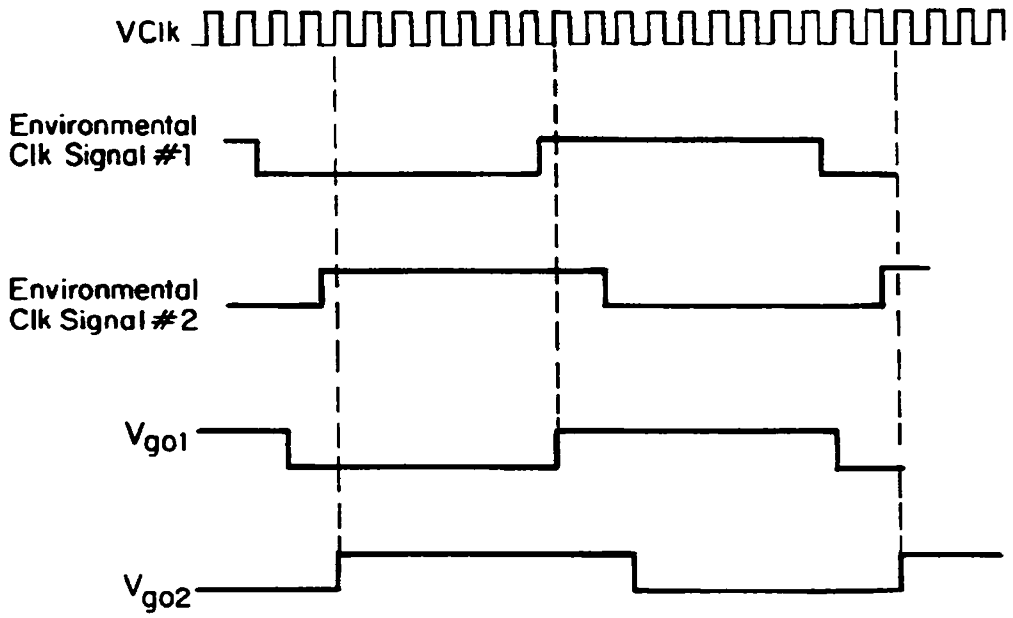

VClk
Environmental Clk Signal #1
Environmental Clk Signal #2
V_{go1}
V_{go2}

FIG. 4B

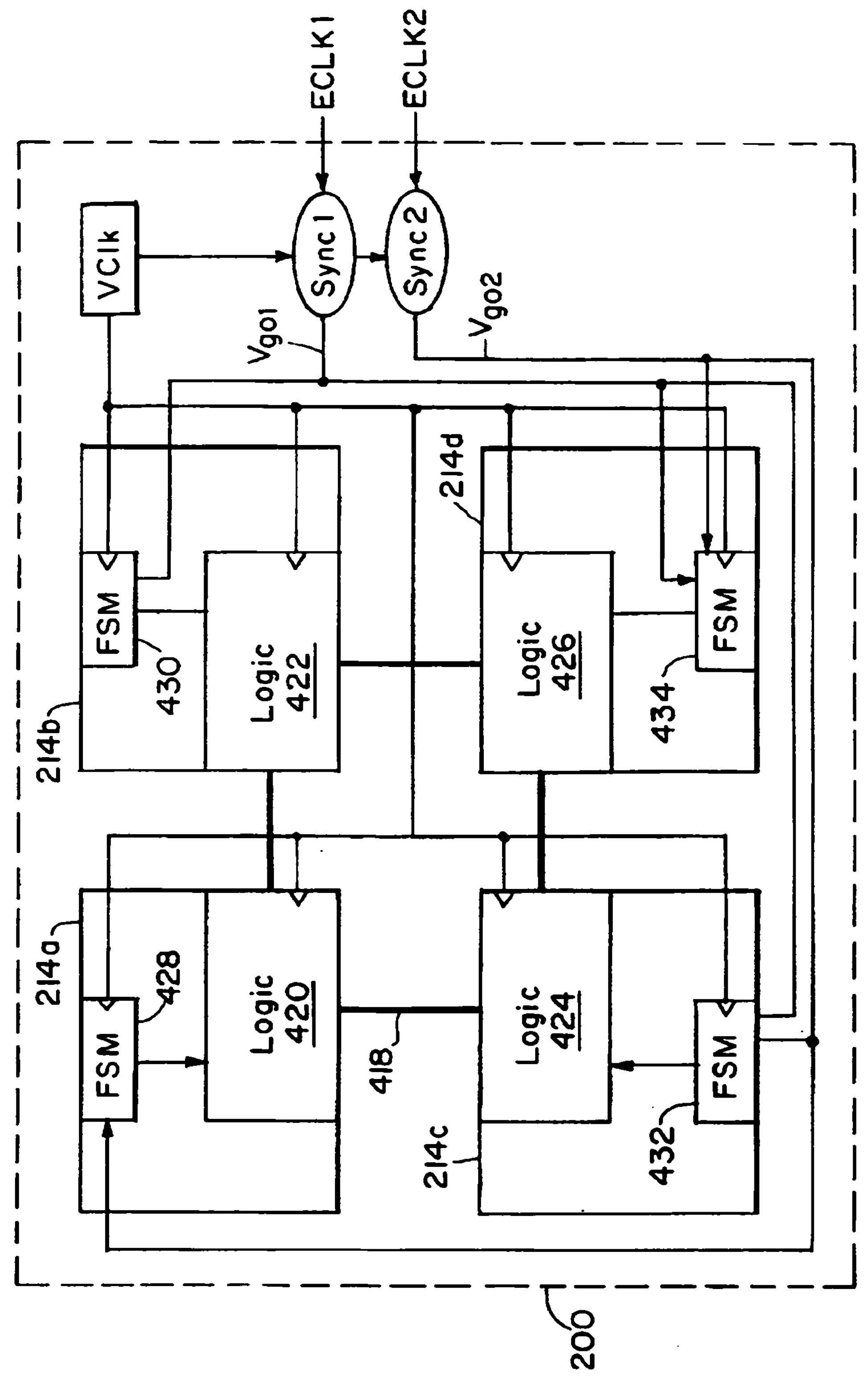

ECLK1
ECLK2
VClk
Sync 1
Sync 2
V_{go1}
V_{go2}
FSM
430
Logic 422
214b
214d
Logic 426
FSM
434
214a
FSM
428
Logic 420
418
Logic 424
FSM
432
214c
200

FIG. 4A

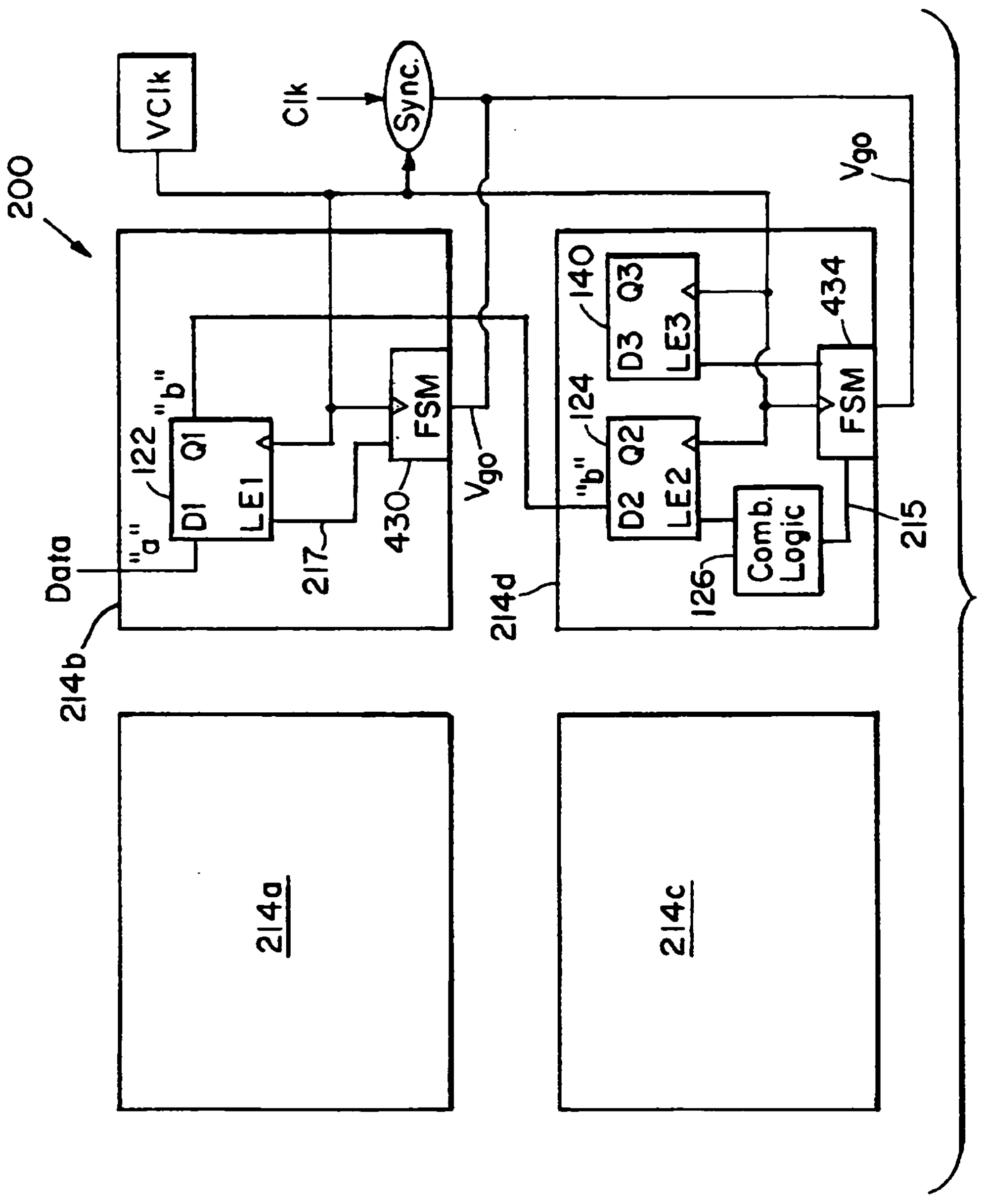
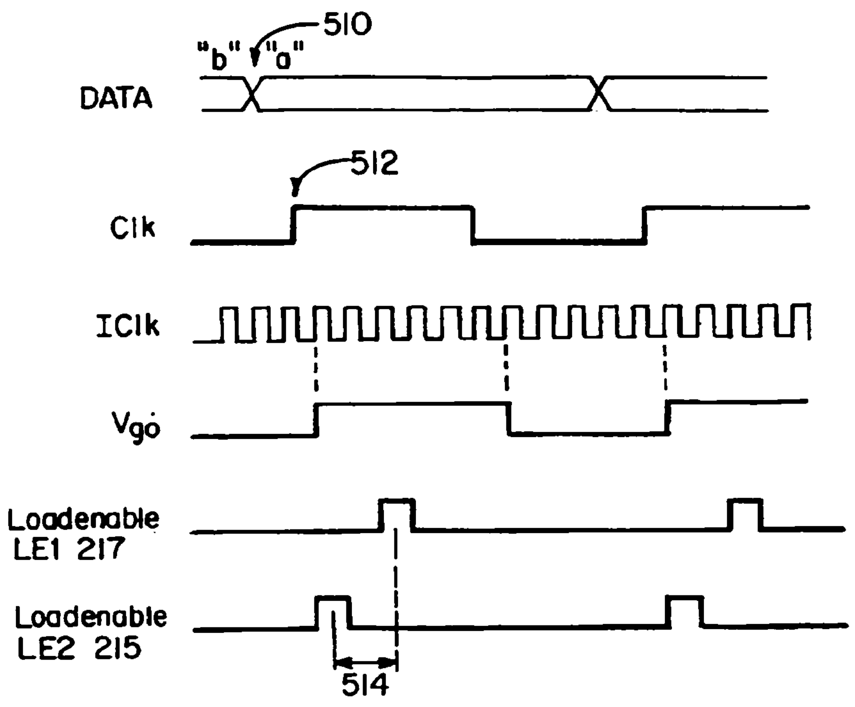
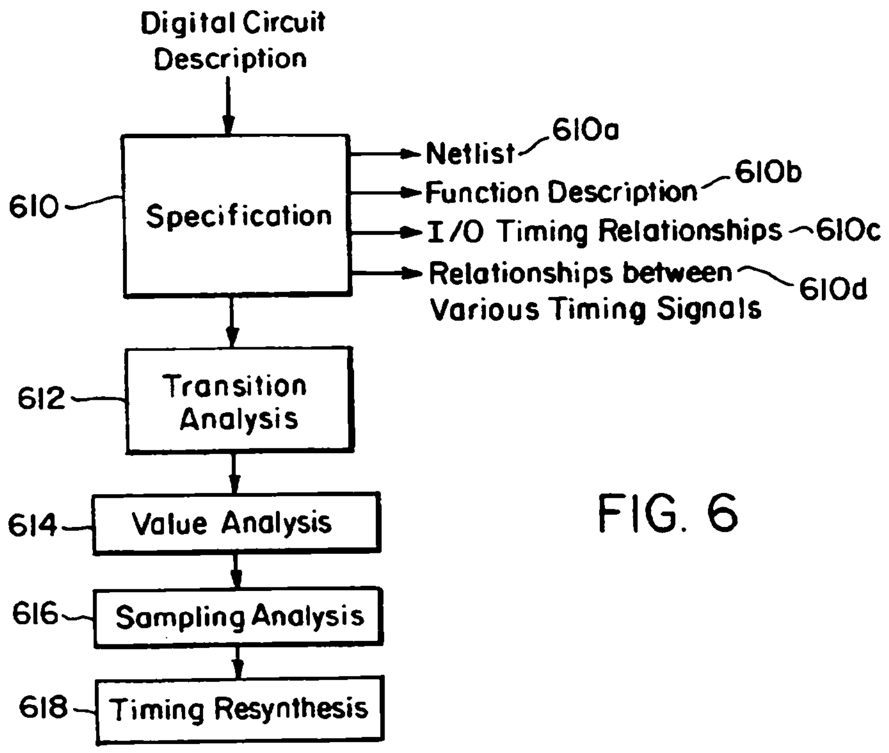

200
VClk
Clk
Sync.
Vgo
Data
"a"
"b"
D1
Q1
LE1
122
217
FSM
430
Vgo
214b
214d
"b"
D2
Q2
LE2
124
140
D3
Q3
LE3
434
FSM
Comb. Logic
126
215
214a
214c

FIG. 5A



510
"b"
"a"
DATA
512
Clk
IClk
Vgo
Loadenable
LE1 217
Loadenable
LE2 215
514

FIG. 5B



Digital Circuit Description
610
Specification
Netlist
610a
Function Description
610b
I/O Timing Relationships
610c
Relationships between Various Timing Signals
610d
612
Transition Analysis
614
Value Analysis
616
Sampling Analysis
618
Timing Resynthesis

FIG. 6

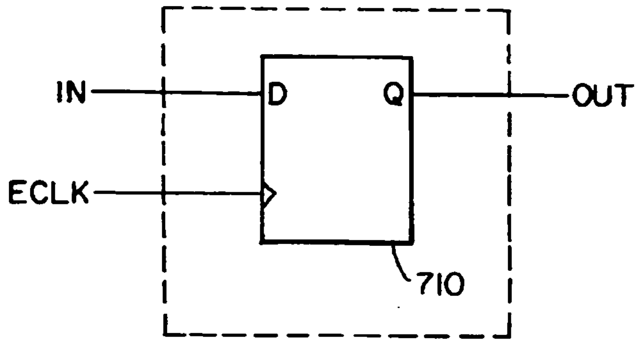

IN
ECLK
D
Q
OUT
710

FIG. 7A

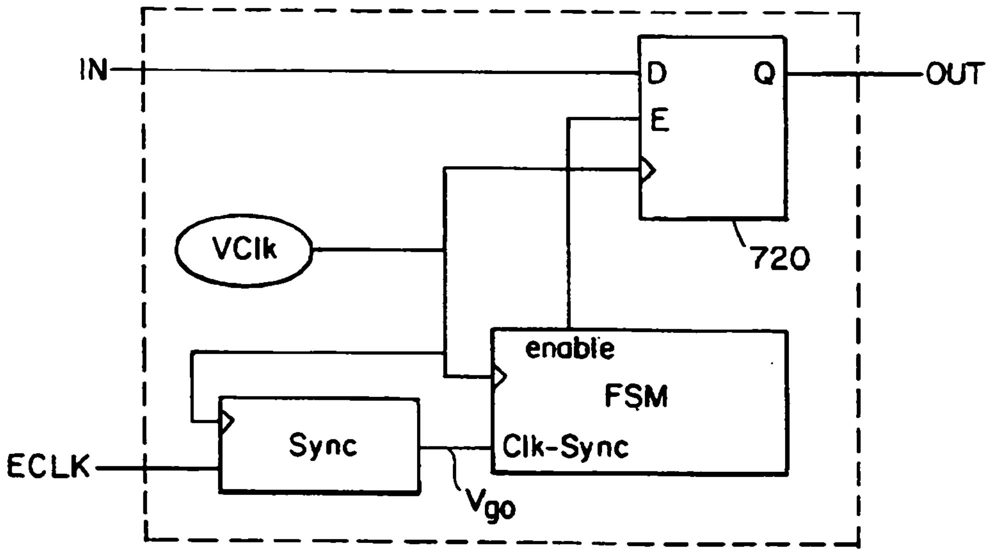
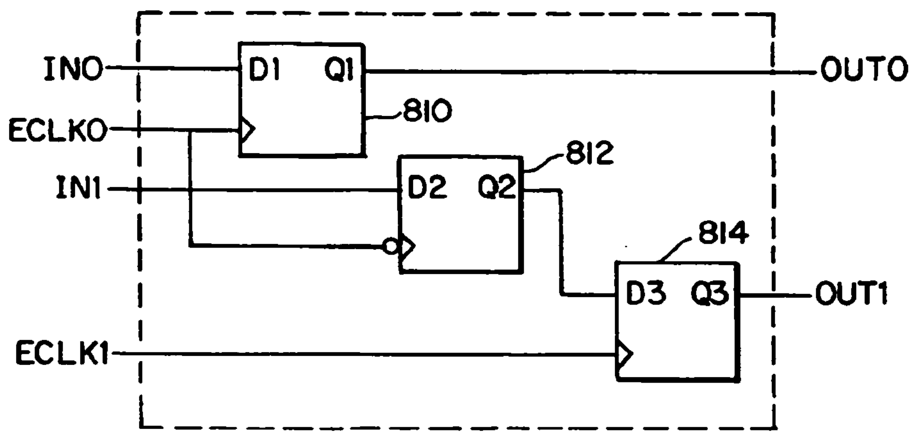
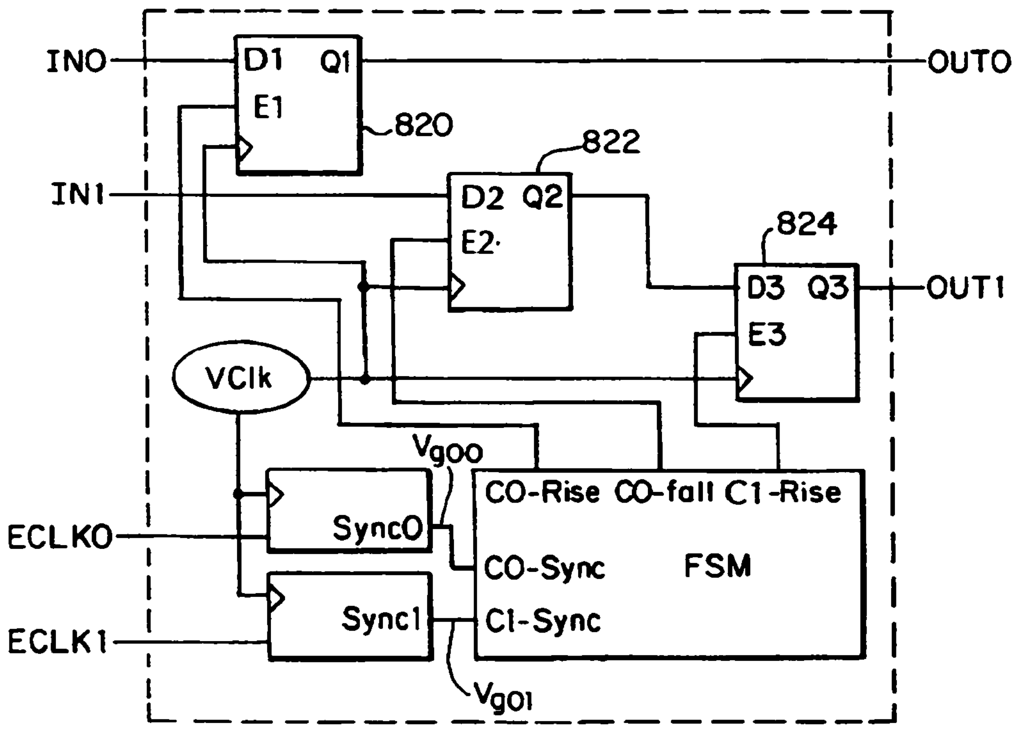

IN
D
Q
OUT
E
VClk
720
enable
FSM
Sync
Clk-Sync
ECLK
Vgo

FIG. 7B



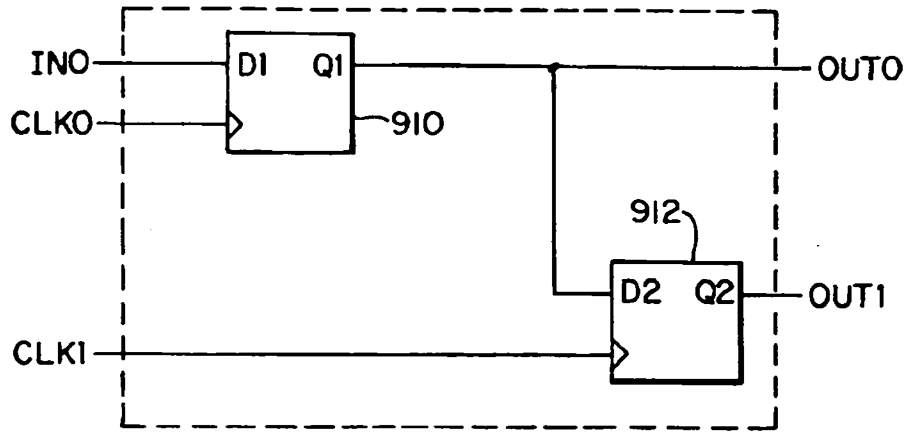
IN0
ECLK0
IN1
ECLK1
D1
Q1
810
D2
Q2
812
814
D3
Q3
OUT0
OUT1

FIG. 8A



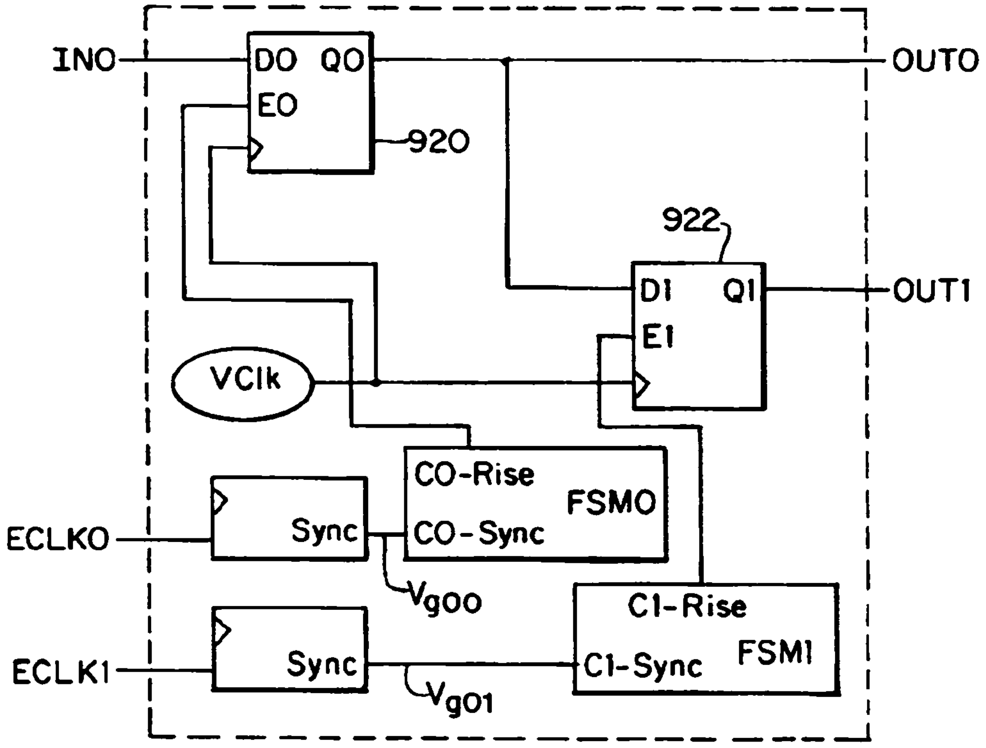
IN0
IN1
D1
Q1
E1
820
D2
Q2
E2
822
824
D3
Q3
E3
OUT0
OUT1
VClk
V_{g00}
Sync0
Sync1
ECLK0
ECLK1
C0-Rise
C0-fall
C1-Rise
C0-Sync
C1-Sync
FSM
V_{g01}

FIG. 8B



IN0
CLK0
CLK1
D1
Q1
910
912
D2
Q2
OUT0
OUT1

FIG. 9A



IN0
D0
Q0
E0
920
922
D1
Q1
E1
OUT0
OUT1
VClk
C0-Rise
FSM0
C0-Sync
ECLK0
Sync
Vg00
C1-Rise
C1-Sync
FSM1
ECLK1
Sync
Vg01

FIG. 9B

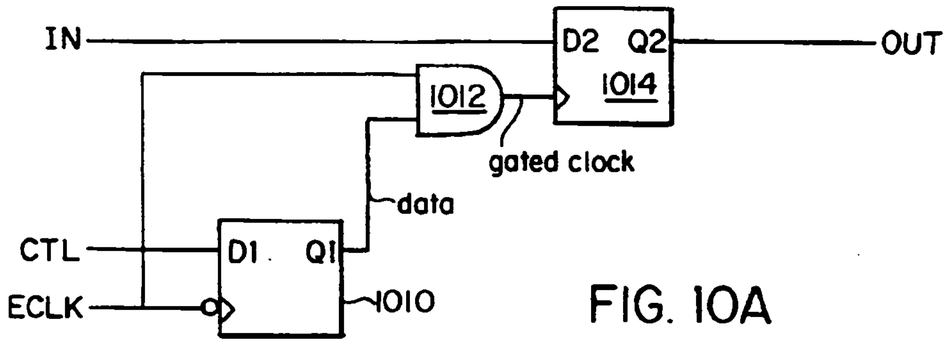


FIG. 10A

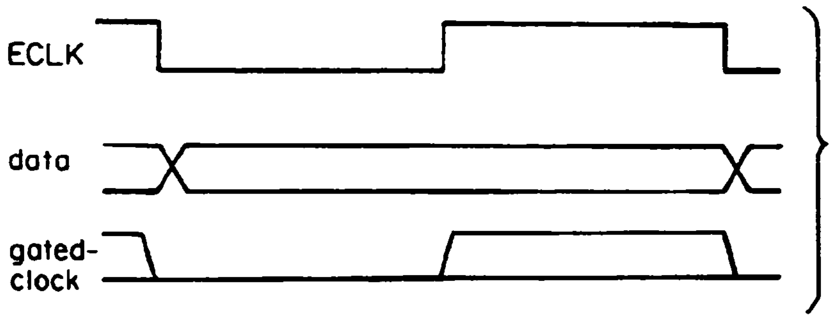


FIG. 10B

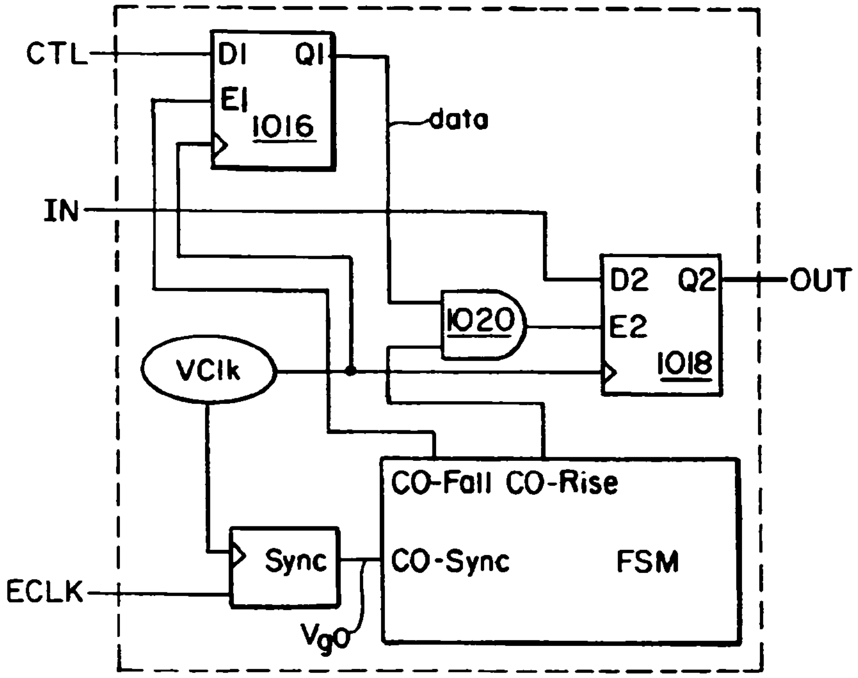


FIG. 10C

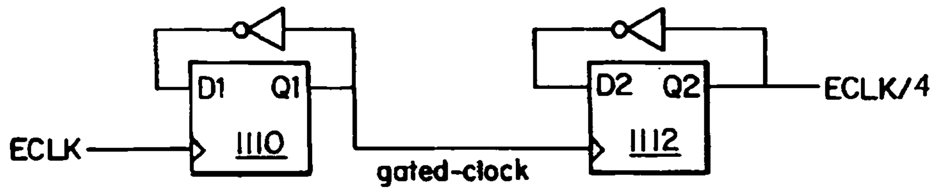


FIG. 11A

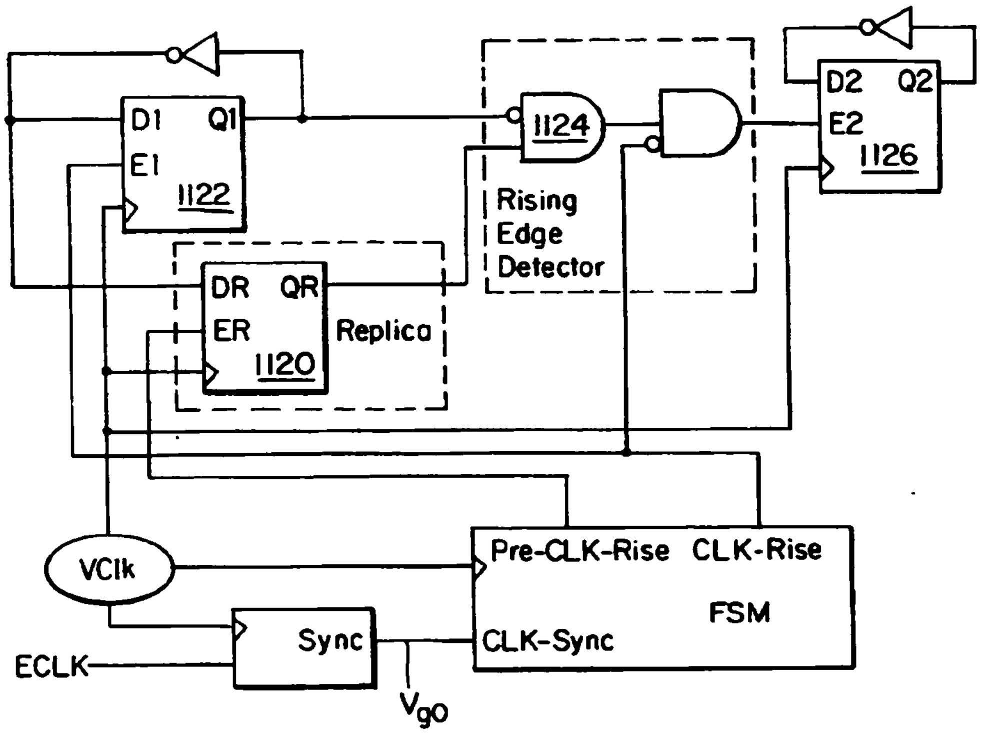


FIG. 11B

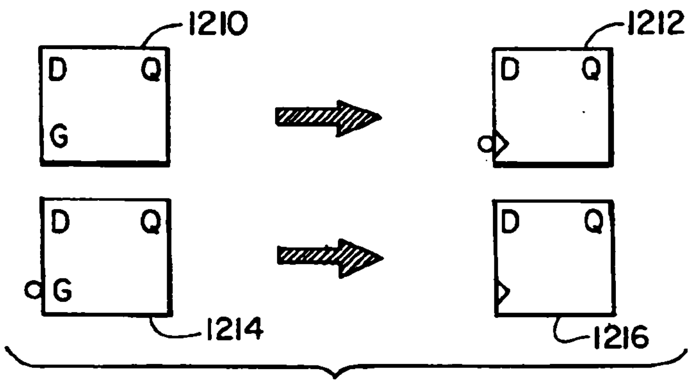


FIG. 12

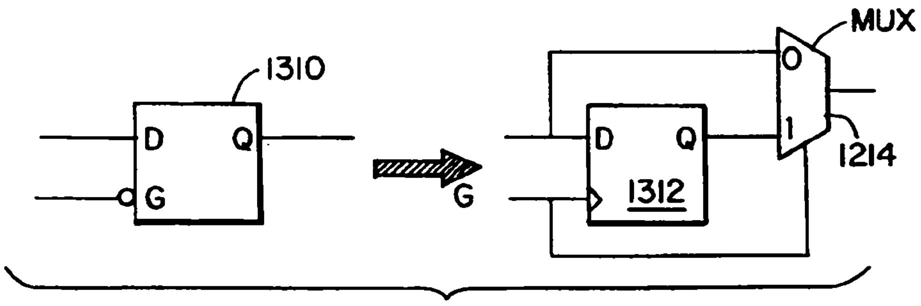


FIG. 13

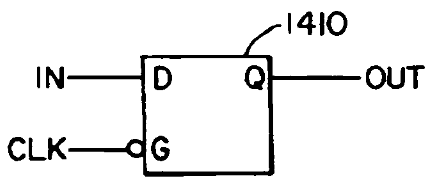


FIG. 14A

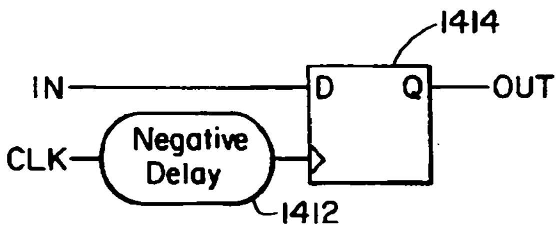


FIG. 14B

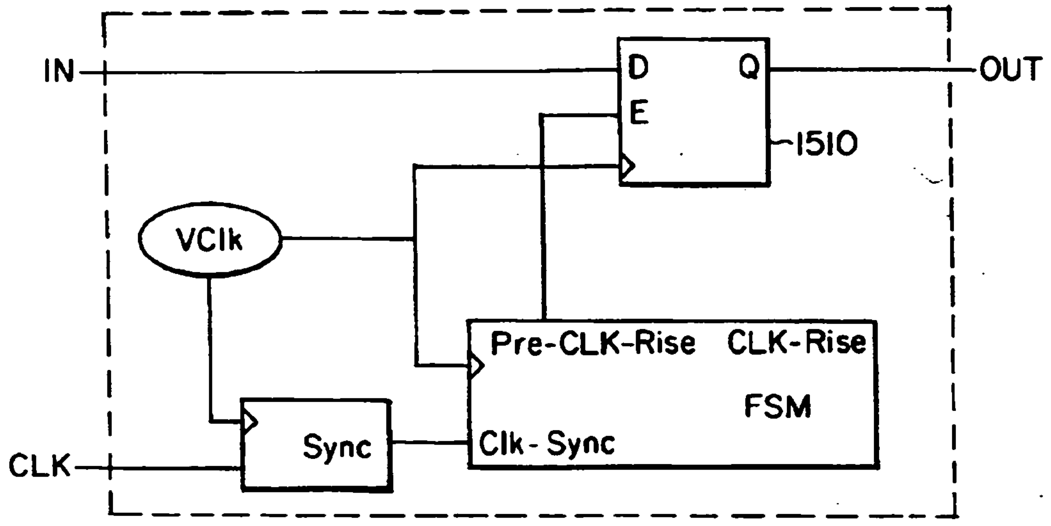


FIG. 15

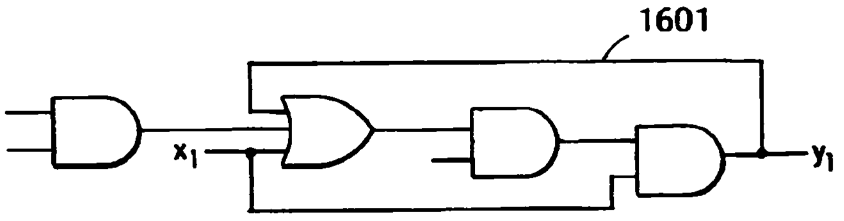


FIG. 16A

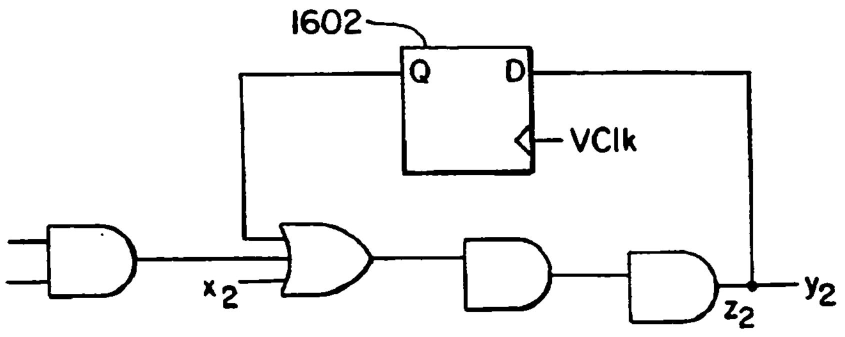


FIG. 16B

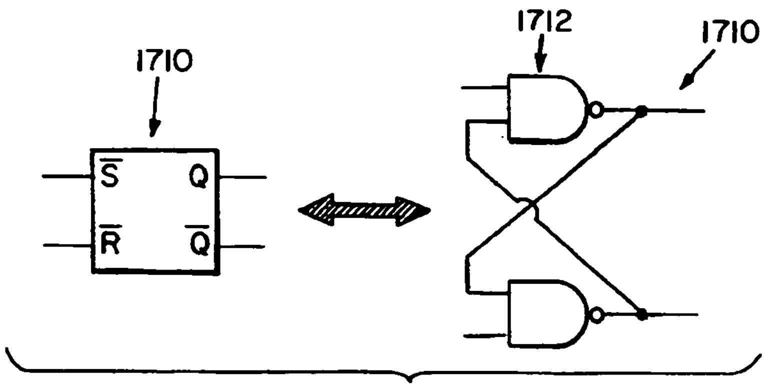


FIG. 17A

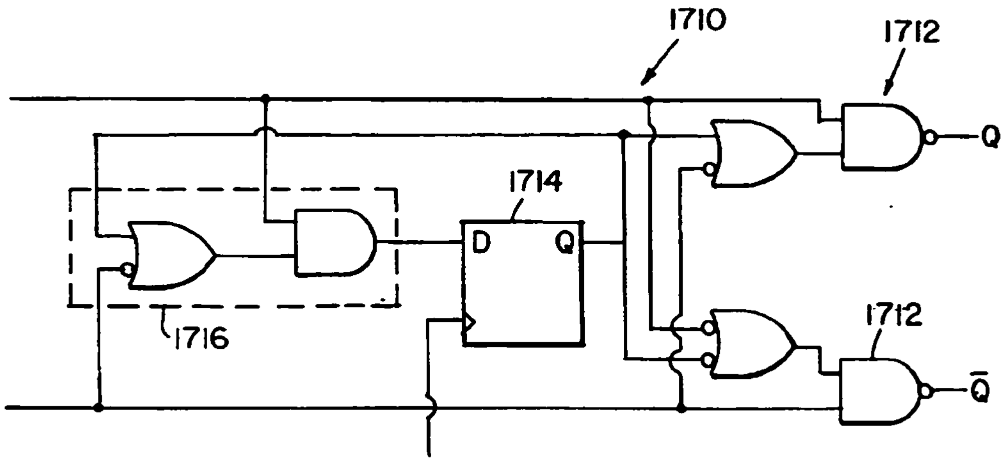


FIG. 17B

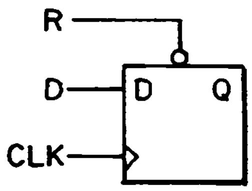


FIG. 18A

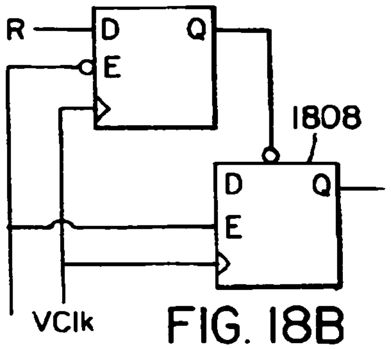


FIG. 18B

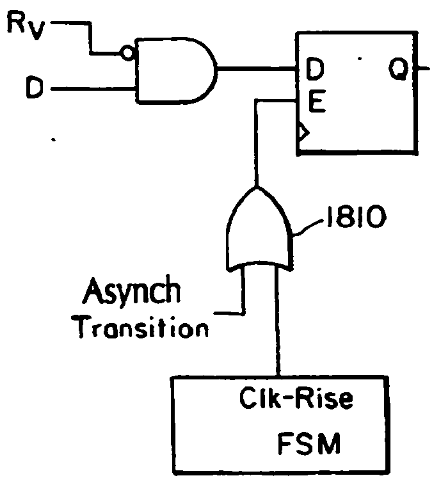


FIG. 18C

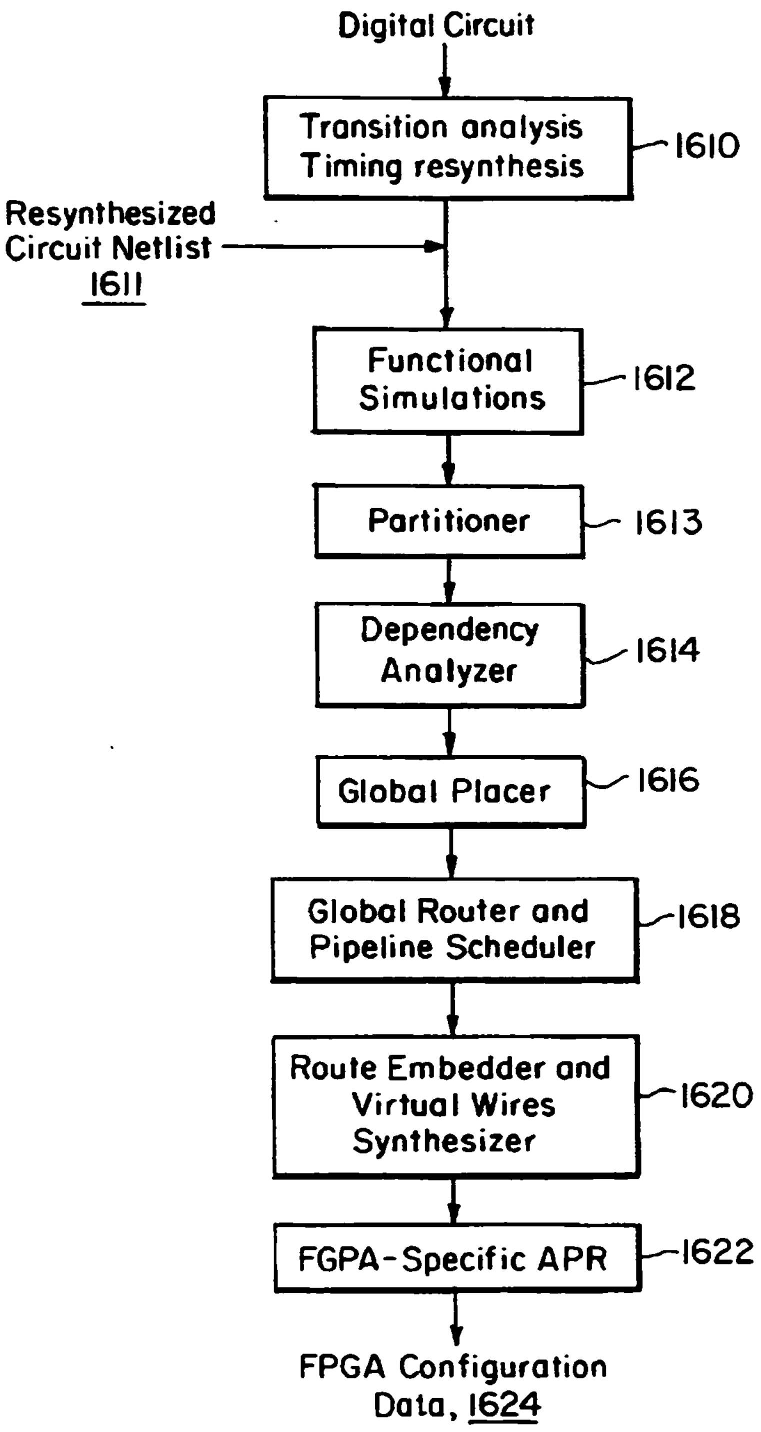

Digital Circuit
Transition analysis Timing resynthesis
1610
Resynthesized Circuit Netlist 1611
Functional Simulations
1612
Partitioner
1613
Dependency Analyzer
1614
Global Placer
1616
Global Router and Pipeline Scheduler
1618
Route Embedder and Virtual Wires Synthesizer
1620
FGPA-Specific APR
1622
FPGA Configuration Data, 1624

FIG. 19

TRANSITION ANALYSIS AND CIRCUIT RESYNTHESIS METHOD AND DEVICE FOR DIGITAL CIRCUIT MODELING

BACKGROUND OF THE INVENTION

Configurable logic devices are a general class of electronic devices that can be easily configured to perform a desired logic operation or calculation. One example is Mask Programmed Gate Arrays (MPGA). These devices offer density and performance. Poor turn around time coupled with only one-time configurability tend to diminish their ubiquitous use. Reconfigurable logic devices or programmable logic devices (such as Field Programmable Gate Arrays (FPGA)) offer lower levels of integration but are reconfigurable, i.e., the same device may be programmed many times to perform different logic operations. Most importantly, the devices can be programmed to create gate array prototypes instantaneously, allowing complete dynamic reconfigurability, something that MPGAs can not provide.

System designers commonly use reconfigurable logic devices such as FPGAs to test logic designs prior to manufacture or fabrication in an effort to expose design flaws. Usually, these tests take the form of emulations in which a reconfigurable logic devices models the logic design, such as a microprocessor, in order to confirm the proper operation of the logic design along with possibly its compatibility with an environment or system in which it is intended to operate.

In the case of testing a proposed microprocessor logic design, a netlist describing the internal architecture of the microprocessor is compiled and then loaded into a particular reconfigurable logic device by some type of configuring device such as a host workstation. If the reconfigurable logic device is a single or array of FPGAs, the loading step is as easy as down-loading a file describing the compiled netlist to the FPGAs using the host workstation or other computer. The programmed configurable logic device is then tested in the environment of a motherboard by confirming that its response to inputs agrees with the design criteria.

Alternatively, reconfigurable logic devices also find application as hardware accelerators for simulators. Rather than testing a logic design by programming a reconfigurable device to "behave" as the logic device in the intended environment for the logic design, e.g., the motherboard, a simulation involves modeling the logic design on a workstation. In this environment, the reconfigurable logic device performs gate evaluations for portions of the model in order to relieve the workstation of this task and thereby decreases the time required for the simulation.

Recently, most of the attention in complex logic design modeling has been directed toward FPGAs. The lower integration of the FPGAs has been overcome by forming heterogeneous networks of special purpose FPGA processors connected to exchange signals via some type of interconnect. The network of the FPGAs is heterogeneous not necessarily in the sense that it is composed of an array of different devices but that the devices have been individually configured to cooperatively execute different sections, or partitions, of the overall logic design. These networks rely on static routing at compile-time to organize the propagation of logic signals through the FPGA network. Static refers to the fact that all data or logic signal movement can be determined and optimized during compiling.

When a logic design intended for eventual MPGA fabrication is mapped to an FPGA, hold time errors are a problem that can arise, particularly in these complex configurable

logic device networks. A digital logic design that has been loaded into the configurable logic devices receives timing signals, such as clock signals, and data signals from the environment in which it operates. Typically, these timing signals coordinate the operation of storage or sequential logic components such as flip-flops or latches. These storage devices control the propagation of combinational signals, which are originally derived from the environmental data signals, through the logic devices.

Hold time problems commonly arise where a timing signal is intended to clock a particular storage element to signal that a value at the element's input terminal should be held or stored. As long as the timing signal arrives at the storage element while the value is valid, correct operation is preserved. Hold time violations occur when the timing signal is delayed beyond a time for which the value is valid, leading to the loss of the value. This effect results in the destruction of information and generally leads to the improper operation of the logic design.

Identification and mitigation of hold time problems presents many challenges. First, while the presence of a hold time problem can be recognized by the improper operation of the logic design, identifying the specific location within the logic design of the hold time problem is a challenge. This requires sophisticated approximations of the propagation delays of timing signals and combinational signals through the logic design. Once a likely location of a hold time problem has been identified, the typical approach is somewhat ad hoc. Delay is added on the path of the combinational signals to match the timing signal delays. This added delay, however, comes at its own cost. First, the operational speed of the design must now take into account this new delay. Also, new hold time problems can now arise because of the changed clock speed. In short, hold time problems are both difficult to identify and then difficult to rectify.

Other problems arise when a logic design intended for ultimate MPGA fabrication, for example, is realized in FPGAs. Latches, for instance, are often implemented in MPGAs. FPGA, however, do not offer a corresponding element.

SUMMARY OF THE INVENTION

The present invention seeks to overcome the hold time problem by imposing a new timing discipline on a given digital circuit design through a resynthesis process that yields a new but equivalent circuit. The resynthesis process also transforms logic devices and timing structures to those that are better suited to FPGA implementation. This new timing discipline is insensitive to unpredictable delays in the logic devices and eliminates hold time problems. It also allows efficient implementation of latches, multiple clocks, and gated clocks. By means of the resynthesis, the equivalent circuit relies on a new higher frequency internal clock (or virtual clock) that is distributed with minimal skew. The internal clock signal controls the clocking of all or substantially all the storage elements, e.g. flip-flops, in the equivalent circuit, in effect discretizing time and space into manageable pieces. The user's clocks are treated in the same manner as user data signals.

In contrast with conventional approaches, the present invention does not allow continuous inter-FPGA signal flow. Instead, all signal flow is synchronized to the internal clock so that signals flow between flip-flops through intermediate FPGAs in discrete hops. The internal clock provides a time base for the circuit's operation.

In general, according to one aspect, the invention features a method of configuring a configurable or programmable logic system. Generally, such logic systems include single or multi-FPGA network, although the invention can be applied to other types of configurable devices. Particular to the invention, the logic system is provided with an internal clock signal that typically has a higher frequency, by a factor of at least four, than timing signals the system receives from the environment in which it is operating. The logic system is configured to have a controller that coordinates operation of the logic in response to the internal clock signal and the environmental timing signals. In the past, while emulation or simulation devices, for example, operated in response to timing signals from the environment, a new internal clock signal, invisible to the environment, was not used to control the internal operations of the devices.

In specific embodiments, a synchronizer is incorporated to essentially generate a synchronized version of the environmental timing signal. This synchronized version behaves much like other data signals from the environment. This synchronizer feeds the resulting sampled environmental clock signals to a finite state machine, which generates control signals. The logic operations are then coordinated by application of these control signals to sequential logic elements.

In more detail, the logic system is configured to have both combinational logic, e.g. logic gates, and sequential logic, e.g. flip-flops, to perform the logic operations. The control signals function as load enable signals to the sequential logic. The internal clock signal is received at the clock terminals of that logic. Just like the original digital circuit design, each sequential logic element operates in response to the environmental timing signals. Now, however, these timing signal control the load enable of the elements, not the clocking. It is the internal clock signal that now clocks the elements. As a result, the resynthesized circuit operates synchronously with a single clock signal regardless of the clocking scheme of the original digital circuit.

In general, according to another aspect, the invention features a method for converting a digital circuit design into a new circuit that is substantially functionally equivalent to the digital circuit design. First, the internal clock signal is defined, then sequential logic elements of the digital circuit design are resynthesized to operate in response to the internal clock signal in the new circuit rather than simply the environmental timing signals.

In specific embodiments, flip-flops of the digital circuit design, which are clocked by the environmental timing signal, are resynthesized to be clocked by the internal clock signal and load enabled in response to the environmental timing signals. Finite state machines are used to actually generate control signals that load enable each flip-flop. The load enable signals are sometimes also generated from a logic combination of finite state machine signals and logic gates.

In other embodiments, latches in the digital circuit design, which were gated by the environmental timing signals, are resynthesized to be flip-flops or latches in future FPGA designs in the new circuit that are clocked by the new internal or virtual clock signal. These new flip-flops are load enabled in response to the environmental timing signals.

In general, according to still another aspect, the invention features a logic system for generating output signals to an environment in response to at least one environmental timing signal and environmental data signals provided from the environment. This logic system has its own internal clock and at least one configurable logic device. The internal architecture of the configurable device includes logic for

generating the output signals in response to the environmental data signals and a controller, specifically a finite state machine, for coordinating operation of the logic in response to the internal clock signal and the environmental timing signal.

Specifically, the logic includes sequential and combinational logic elements. The sequential logic elements are clocked by the internal clock signal and load enabled in response to the environmental timing signals.

The above and other features of the invention including various novel details of construction and combinations of parts, and other advantages, will now be more particularly described with reference to the accompanying drawings and pointed out in the claims. It will be understood that the particular method and device embodying the invention is shown by way of illustration and not as a limitation of the invention. The principles and features of this invention may be employed in various and numerous embodiments without the departing from the scope of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

In the accompanying drawings, like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale and in some cases have been simplified. Emphasis is instead placed upon illustrating the principles of the invention. Of the drawings.

FIG. 1 is a schematic diagram showing a prior art emulation system and its interaction with an environment and a host workstation;

FIG. 2 shows a method for impressing a logic design on the emulation system;

FIG. 3 is a schematic diagram of a configurable logic system that comprises four configurable logic devices—a portion of the internal logic structure of these devices has been shown to illustrate the origins of hold time violations;

FIG. 4A is a schematic diagram of the logic system of the present invention showing the internal organization of the configurable logic devices and the global control of the logic devices by the internal or virtual clock;

FIG. 4B is a timing diagram showing the timing relationships between the internal or virtual clock signal, environmental timing signals, and control signals generated by the logic system;

FIG. 5A is a schematic diagram of a logic system of the present invention that comprises four configurable logic devices, the internal structure of these devices is the functional equivalent of the structure shown in FIG. 3 except that the circuit has been resynthesized according to the principles of the present invention;

FIG. 5B is a diagram showing the timing relationship between the signals generated in the device of FIG. 5A;

FIG. 6 illustrates a method by which a digital circuit description having an arbitrary clocking methodology is resynthesized into a functionally equivalent circuit that is synchronous with a single internal clock;

FIGS. 7A and 7B illustrate a timing resynthesis circuit transformation in which an edge-triggered flip-flop is converted into a load-enable type flip-flop;

FIGS. 8A and 8B illustrate a timing resynthesis circuit transformation in which a plurality of edge triggered flip-flops clocked by two phase-locked clock signals are converted into load enable flip-flops that are synchronous with the internal clock signal;

FIGS. 9A and 9B illustrate a timing resynthesis circuit transformation in which two edge triggered flip-flops clocked by two arbitrary clock signals are transformed into load enabled flip-flops that operate synchronously with the internal clock signal;

FIGS. 10A, 10B, and 10C illustrate a timing resynthesis circuit transformation in which two edge-triggered flip-flops, one of which is clocked by a gated clock, are transformed into two load-enable flip-flops that operate synchronously with the internal clock signal, FIG. 10B is a timing diagram showing the signal values over time in the circuit;

FIG. 11A and 11B illustrate a timing resynthesis circuit transformation in which a complex gated clock structure, with a second flip-flop being clocked by a gated clock, is converted into a circuit containing three flip-flops and an edge detector, all of the flip-flops operating off of the internal clock signal in the new circuit;

FIG. 12 illustrates circuit transformations in which gated latches are converted into edge-triggered flip-flops on the assumption that the latches are never sampled when open, i.e., latch output is not registered into another storage element when they are open;

FIG. 13 illustrates a timing resynthesis circuit transformation in which a gated latch is converted into an edge-triggered flip-flop and a multiplexor;

FIGS. 14A and 14B illustrate a timing resynthesis circuit transformation in which a latch is converted to an edge-triggered flip-flop with a negative delay at the clock input terminal to avoid glitches;

FIG. 15 illustrates a timing resynthesis circuit transformation of the negative delay flip-flop of FIG. 14B into a flip-flop that operates synchronously with the internal clock signal;

FIGS. 16A and 16B illustrate a timing resythesis circuit transformation in which a flip-flop is inserted in a combinational loop to render the circuit synchronous with the virtual clock;

FIGS. 17A and 17B illustrate a timing resynthesis circuit transformation in which an RS flip-flop is transformed into a device that is synchronous with the virtual clock;

FIG. 18A, 18B, and 18C illustrate a timing resynthesis circuit transformation for handling asynchronous preset and clears of state elements; and

FIG. 19 illustrates the steps performed by a compiler that resynthesizes the digital circuit design and converts it into FPGA configuration data that is loaded into the logic system 200.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Turning now to the drawings, FIG. 1 is a schematic diagram showing an emulation system 5 of the prior art. The emulation system 5 operates in an environment such as a target system 4 from which it receives environmental timing signals and environmental data signals and responsive to these signals generates output data signals to the environment. A configuring device 2 such as a host workstation is provided to load configuration data into the emulation system 5.

The emulation system 5 is usually constructed from individual configurable logic devices 12, specifically FPGA chips are common. The configurable logic devices 12 are connected to each other via an interconnect 14. Memory elements 6 are also optionally provided and are accessible by the configurable logic devices 12 through the interconnect 14.

The host workstation 2 downloads the configuration data that will dictate the internal configuration of the logic

devices 12. The configuration data is compiled from a digital circuit description that includes the desired manner in which the emulation system 5 is intended to interact with the environment or target system 4. Typically, the target system 4 is a larger electronic system for which some component such as a microprocessor is being designed. The description applies to this microprocessor and the emulation system 5 loaded with the configuration data confirms compatibility between the microprocessor design and the target system 4. Alternatively, the target system 4 can be a device for which the logic system satisfies some processing requirements. Further, the emulation system 5 can be used for simulations in a software or FPGA based logic simulation.

FIG. 2 illustrates how the logic design is distributed among the logic devices 12 of the logic system 5. A netlist 20 describing the logic connectivity of the logic design is separated into logic partition blocks 22. The complexity of the blocks 22 is manipulated so that each can be realized in a single FPGA chip 12. The logic signal connections that must bridge the partition blocks 24, global links, are provided by the interconnect 14. Obviously, the exemplary netlist 20 has been substantially simplified for the purposes of this illustration.

FIG. 3 illustrates the origins of hold time problems in conventional logic designs. The description is presented in the specific context of a configurable system 100, such as an emulation system, comprising four configurable logic devices 110–116, such as FPGAs, which are interconnected via a crossbar 120 interconnect. A portion of the internal logic of these devices is shown to illustrate the distribution of a gated clock and the potential problems from the delay of the clock.

The second logic device 112 has been programmed with a partition of the intended logic design that includes an edge-triggered D-type flip-flop 122. This flip-flop 122 receives a data signal DATA at an input terminal D1 and is clocked by a clock signal CLK, both of which are from the environment in which the system 100 is intended to operate. The output terminal Q1 of the first flip-flop is connected to a second flip-flop 124 in the fourth logic device 116 through the crossbar 120. This second flip-flop 124 is also clocked by the clock signal, albeit a gated version that reaches the second flip-flop 124 through the crossbar 120, through combinational logic 126 on a third configurable logic device 114 and through the crossbar 120 a second time before it reaches the clock input of the second flip-flop 124.

Ideally, the rising edge of the clock signal should arrive at both the first flip-flop 122 and the second flip-flop 124 at precisely the same time. As a result of this operation, the logic value "b" held at the output terminal Q1 of the first flip-flop 122 and appearing at the input terminal D2 of the second flip-flop 124 will be latched to the output terminal Q2 of the second flip-flop 124 as the data input is latched by flip-flop 122. The output terminals Q1 and Q2 of the flip-flops 122, 124 will now hold the new output values "a" and "b". This operation represents correct synchronous behavior.

The more realistic scenario, especially when gated clocks are used, is that the clock signal CLK will not reach both of the flip-flops 122 and 124 at the same instant in time. This realistic assumption is especially valid in the illustrated example in which the clock signal CLK must pass through the combinational logic 126 on the third configurable logic device 114 before it reaches the second flip-flop 124 on the fourth configurable logic device 116. In this example, assume the clock signal CLK reaches the first flip-flop 122 in the second configurable logic device 112 and clocks the value at that flip-flop's input terminal D1 to the output Q1. At some point, the output Q1 of the first flip-flop is now holding the new value "a" and this new value begins to propagate toward the input D2 of the second flip-flop 124. The rising edge of the clock signal CLK has not propagated to the second flip-flop 124 on the fourth configurable logic device 116, however. Instead, a race of sorts is established between the rising edge of the clock signal CLK and the new value "a" to the second flip-flop 124. If the new value "a" reaches the input terminal D2 of the second flip-flop before the rising edge of the clock signal CLK, the old value "b" will be over-written. This is incorrect behavior since the information contained in "b" is lost. For correct operation of the circuit, it was required that signal "b" at the input terminal D2 of the second flip-flop 124 be held valid for a brief period of time after the arrival of the clock edge to satisfy a hold time requirement. Unfortunately, unpredictable routing and logic delays postpone the clock edge beyond the validity period for the input signal "b".

In environments where delays can not be predicted precisely, hold time violations are a serious problem that can not be rectified merely by stretching the length of the clock period. Often, there is a need for careful delay tuning in traditional systems, either manually or automatically, in which analog delays are added to signal paths in the logic. The delays usually require further decreases in the operational speed of the target system. This lengthens the periods of the environmental timing signals and gives the emulation system more time to perform the logic calculations. These changes, however, create their own timing problems, and further erode the overall speed, ease-of-use, and predictability of the system.

FIG. 4A is a schematic diagram showing the internal architecture of the logic system 200 which has been configured according to the principles of the present invention. This logic system 200 comprises a plurality of configurable logic devices 214a–214d. This, however, is not a strict necessity for the invention. Instead, the logic system 200 could also be constructed from a single logic device or alternatively from more than the four logic devices actually shown. The logic devices are shown as being connected by a Manhattan style interconnect 418. Again, the interconnect is non-critical, modified Manhattan-style, crossbars or hierarchial interconnects are other possible and equivalent alternatives.

The internal logic architecture of each configurable logic device 214a–214d comprises a finite state machine 428–434 and logic 420–426. An internal or virtual clock VClk generates an internal or virtual clock signal that is distributed through the interconnect 418 to each logic device 214a–214d, and specifically, the logic 420–426 and finite state machines 428–434. Generally, the logic 420–426 performs the logic operations and state transitions associated with the logic design that was developed from the digital circuit description. The finite state machines 428–434 control the sequential operations of the logic in response to the signal from the virtual clock VClk.

The logic system 200 operates synchronously with the single internal clock signal VClk. Therefore, a first synchronizer SYNC1 and a second synchronizer SYNC2 are provided to essentially generate synchronous versions of timing signals from the environment. In the illustrated example, they receive environmental timing signals BClk1 and BClk2, respectively. The synchronizers SYNC1 and SYNC2 also receive the internal clock signal VClk. Each of the synchronizers SYNC1 and SYNC2 generates a synchronizing control signal V_{GO1}, V_{GO2} in response to an edge of the respective environmental timing signal BClk1 and BClk2, upon the next transition of the internal clock VClk. Thus, these control signals are synchronous with the internal clock.

FIG. 4B shows an exemplary timing diagram of the virtual clock signal VClk compared with a first environmental clock signal BClk1 and a second environmental clock signal BClk2. As shown, typically, the virtual clock VClk is substantially faster than any of the environmental clocks, at least four times faster but usually faster by a factor of ten to twenty. As a general rule, the temporal resolution of the virtual clock, i.e., the cycle time or period of the virtual clock, should be smaller than the time difference between any pair of environmental timing signal edges.

In the example, the environmental clocks BClk1 and BClk2 are rising edge-active. The signals V_{GO1} and V_{GO2} from the first synchronizer SYNC1 and the second synchronizer SYNC2, respectively, are versions of the environmental clock which are synchronized to the internal clock VClk in duration. The transitions occur after the rising edges of the environmental clocks BClk1 and BClk2, upon the next or a later rising edge of the internal clock. For example, the second synchronizing signal V_{GO2} is active in response to the receipt of the second environmental clock signal BClk2 upon the next rising edge of the internal clock VClk.

Returning to FIG. 4A, in typical simulation or emulation configurable systems and the present invention, logic of the configurable devices include a number of interconnected combinational components that perform the boolean functions dictated by the digital circuit design. An example of such components are logic gates. Other logic is configured as sequential components. Sequential components have an output that is a function of the input and state and are clocked by a timing signal. An example of such sequential components would be a flip-flop. In the typical configurable systems, the environmental clock signals are provided to the logic in each configurable logic device to control sequential components in the logic. This architecture is a product of the emulated digital circuit design in which similar components were also clocked by these timing signals. The present invention, however, is configured so that each one of these sequential components in the logic sections 420–426 is clocked by the internal or virtual clock signal VClk. This control is schematically shown by the distribution of the internal clock signal VClk to each of the logic sections 420–426 of the configurable devices 410–416. As described below, the internal clock is the sole clock applied to the sequential components in the logic sections 420–426 and this clock is preferably never gated.

Finite state machines 428–434 receive both the internal clock signal VClk and also the synchronizing signals V_{GO1} V_{GO2} from the synchronizers SYNC1 and SYNC2. The finite state machines 428–434 of each of the configurable logic devices 410–416 generate control signals to the logic sections 420–426. These signals control the operation of the sequential logic components. Usually, the control signals are received at load enable terminals. As a result, the inherent functionality of the original digital circuit design is maintained. The sequential components of the logic are operated in response to environmental timing signals by virtue of the fact that loading occurs in response to the synchronized versions of the timing signals, i.e. V_{GO1} V_{GO2}. Synchronous operation is imposed, however, since the sequential components are actually clocked by the single internal clock signal VClk throughout the logic system 200. In contrast, the typical simulation or emulation configurable systems would clock the sequential components with the same environmental clock signals as in the original digital circuit description.

It should be noted that separate finite state machines are not required for each configurable logic device. Alternatively, a single finite state machine having the combined functionality of finite state machines 428–434 could be implemented. For example, one configurable device could be entirely dedicated to this combined finite state machine. Generally, however, at least one finite state machine on each device chip is preferred. The high cost of interconnect bandwidth compared to on-chip bandwidth makes it desirable to distribute only the synchronizing signals V_{CO1} V_{CO2} to each chip, and generate the multiple control signals on-chip to preserve the interconnect for other signal transmission.

FIG. 5A shows a portion of a logic circuit that has been programmed into the logic system 200 according to the present invention. This logic circuit is a resynthesized version of the logic circuit shown in FIG. 3. That is, the logic circuit of FIG. 5A and of FIG. 3 have many of the same characteristics. Both comprise flip-flops 122 and 124. The flip-flop 122 has an output terminal Q1 which connects to the input terminal D2 of flip-flop 124. Further, the combinational logic 126 is found in both circuits.

The logic circuit of FIG. 5A differs from FIG. 3 first in that each of the flip-flops 122 and 124 are load-enable type flip-flops and clocked by a single internal clock VClk. Also, the environmental clock signal Clk is not distributed per se to both of the flip-flops 122 and 124 as in the circuit of FIG. 3. Instead, a synchronized version of the clock signal V_{CO} is distributed to a finite state machine 430 of the second configurable logic device 214*b* and is also distributed to a finite state machine 434 of the fourth configurable logic device 214*d*. The finite state machine 430 then provides a control signal to a load enable terminal LE1 of flip-flop 122 and finite state machine 434 provides a control signal to the load enable terminal LE2 of flip-flop 124 through the combinational logic 126.

FIG. 5B is a timing diagram showing the timing of the signals in the circuit of FIG. 5A. That is, at time 510, new data is provided at the input terminal D1 of flip-flop 522. Then, at some later time, 512, the clock signal Clk is provided to enable the flip-flop 122 to clock in this new data. The second flip-flop 124 is also intended to respond to the environmental clock signal Clk by capturing the previous output of flip-flop 122 before that flip-flop is updated with the new data. Recall that the problem in the logic circuit of FIG. 3 was that the clock signal to the second flip-flop 124 was gated by the combinational logic 126 which delayed that clock signal beyond time at which the output "b" from the output terminal Q1 of the flip-flop 122 was valid. In the present invention, the environmental clock signal Clk is received at the synchronizer SYNC. This synchronizer also receives the virtual clock signal VClk. The output of the synchronizer V_{CO} is essentially the version of the environmental clock signal that is synchronized to the internal clock signal. Specifically, the new signal V_{CO} has rising and falling edges that correspond to the rising edges of the internal clock signal VClk.

The finite state machines 430 and 434 are individually designed to control the flip-flops in the respective configurable logic 214*b* and 214*d* to function as required for correct synchronous operation. Specifically, finite state machine 434 generates a control signal 215 which propagates through the combinational logic 126 to the load enable terminal LE2 of the flip-flop 124. This propagation of

control signal 215 from finite state machine, through combinational logic 126, to LE2 occurs in a single virtual clock cycle. The generation of control signal 215 precedes the generation of control signal 217 by the finite state machine 430 by a time of two periods (for example) of the internal clock VClk. This two cycle difference, 514, assumes that flip-flop 124 is enabled before flip-flop 122 is enabled, thereby latching "b", and thus providing correct operation. As a result, both flip-flop 122 and flip-flop 124 are load enabled in a sequence that guarantees that a new value in flip-flop 122 does not reach flip-flop 124 before flip-flop 124 is enabled. In fact, if the compiler has scheduled "b" to arrive at D2 on some cycle, x, later than 217, then the compiler can cause control signal 215 to be available on that cycle x, or later. In the above instance, the correct circuit semantics is preserved even though control signal 215 arrives after control signal 217. The key is that 215 must enable flip-flop 124 in a virtual cycle in which "b" is at D2.

Further, the precise control of storage elements afforded by the present invention allows set up and hold times into the target system to be dictated. In FIG. 5A, output Q2 of flip-flop 124 is linked to a target system via a third flip-flop 140. The flip-flop 140 is load enabled under the control of finite state machine 434 and clocked by the virtual clock. Thus, by properly constructing this finite state machine 434, the time for which flip-flop 140 holds a value at terminal Q3 is controllable to the temporal resolution of a cycle or period of the virtual clock signal.

This aspect of the invention enables the user to test best case and worst case situations for signal transmission to the target system and thereby ensure that the target system properly captures these signals. In a similar vein, this control also allows the user to control the precise time of sampling signals from the target system by properly connected storage devices.

FIG. 6 illustrates a method by which a digital circuit design with an arbitrary clocking methodology and state elements is transformed into a new circuit that is synchronous with the internal clock signal but is a functional equivalent of the original digital circuit. The state elements of the new circuit are exclusively edge triggered flip-flops.

The first step is specification 610. This is a process by which the digital circuit design along with all of the inherent timing methodology information required to precisely define the circuit functionality is identified. This information is expressed in four pieces, a first piece of which is the gate-level circuit netlist 610*a*. This specifies the components from which the digital circuit is constructed and the precise interconnectivity of the components.

The second part 610*b* of the specification step 610 is the generation of a functional description of each component in the digital circuit at the logic level. For combinatorial components, this is a specification of each output as a boolean function of one or more inputs. For example, the specification of a three input OR gate—inputs A, B, and C and an output O—is O=A+B+C. For sequential components, this entails the specification of outputs as a boolean function of the inputs and state. The specification of the new state as a boolean function of the inputs and state is also required for the sequential components along with the specification of when state transitions occur as a function of either boolean inputs or directed input transitions. A directed input transition is a rising or falling edge of an input signal, usually a timing signal from the environment in which the logic system 200 is intended to ultimately function. For example, the specification of a rising edge-triggered flip-flop—inputs

D, CLK, of output Q, and state S— is Q=S, S=D, and state transition when CLK rises.

Another part of the specification step is the description of the timing relationships of the inputs to the logic system step 610c. This includes environment timing signals and environmental input signals and the relationship to the output signals generated by the logic system 200 to the environment. Input signals to the logic system 200 can be divided into two classes: timing signals and environmental data signals. The timing signals are generally environmental clock signals, but can also be asynchronous resets and any other form of asynchronous signal that combinatorially reach inputs of state elements involved in the functions triggering state transitions. In contrast, environmental data signals include environmental output signals and output signals to the environment that do not combinatorially reach transition controlling inputs of state elements. The timing relationship also specify the timing of environmental data signals relative to a timing signal.

The specification step must also include the specification of the relative timing relationships for all timing signals step 610*d*. These relationships can be one of three types:

A basket of timing signals can be phase-locked. Two signals of equal frequency are phase-locked if there is a known phase relationship between each edge of one signal and each edge of the other signal. For example, the first environmental clock signal and the second environmental clock signal illustrated in FIG. 4 would be phase-locked signals. Additionally, two signals of integrally related frequency are phase-locked if there is a known phase relationship, relative to the slower signal, between any edge of the slower signal and each edge of the faster signal. Two signals of rationally related frequency are phase-locked if they each are phase-locked to the same slower signal.

Another type of timing relationship is non-simultaneous. Two signals are non-simultaneous if a directed transition in one signal guarantees that no directed transition will occur in the other within a window around the transition of some specified finite duration. If two signals are non-simultaneous and also not phase-locked, this implies that one signal is turned off while the other is on and vice versa. For example, two non-simultaneous signals might be two signals that indicate the mutually exclusive state of some component in the environment. The first signal would indicate if the component was in a first condition and the second timing signal would indicate if the component were in a second condition and the first and second condition could never happen at the same time.

Finally, the last type of relationship is asynchronous. Two signals are asynchronous if the knowledge about a directed transition of one of the signals imparts no information as to occurrence of a transition in the other signal.

It should be recognized that phase-locked is a transitive relationship so that there will be collections of one or more clocks that are mutually phase-locked with respect to each other. Such collection of phase-locked clocks is referred to as a domain. Relationship between domains are either non-simultaneous or asynchronous. The timing signals must be decomposed into a collection of phase-locked domains, and the relationship between pairs of the resulting domains, either synchronous or non-simultaneous, must be specified.

The ordering of the edges of timing signals within each domain are also specified. For example, first CLK1 rises, then CLK2 rises, then CLK2 falls and then CLK1 falls.

A transition analysis step 612, value analysis step 614, and sampling analysis step 616 are used to determine when,

relative to the times at which transitions occur on timing signals, signals within a digital circuit change value, and where possible, what these values are. Also determined is when the values of particular signals are sampled by state elements within the circuit as a separate analysis.

In the transition analysis step 612, a discrete time range is established for each clock domain including one time point for each edge of a clock within the domain. All edges within the domain are ordered and the ordering of time points corresponds to this ordering of edges.

In the value analysis step 614, the steady state characteristics of every wire in the digital circuit is determined for each discrete time range. Within a discrete time range, any wire within the digital circuit can either be known to be O, known to be 1, known not to rise, known not to fall or known not to change, or a combination of not falling and not rising. A conservative estimate of the behavior of an output of a logic component can be deduced from the behavior of its inputs. Information about environmental timing signals and environmental data signals can be used to define their behavior. Based on the transition and value information of the inputs to the logic system corresponding information can be deduced for the outputs of each component. A relaxation algorithm is used, in which output values of a given component are recomputed any time an input changes. If the outputs in turn change, this information is propagated to all the places the output connects, since these represent more inputs which have changed. The process continues until no further changes occur.

A second relaxation process, similar to that for transition and value analysis, is used in the sampling step 616. Sampling information reflects the fact that at some point in time, the value carried on a wire may be sampled by a state element, either within the logic system 200 or by the environment. Timing information for output data signals to the environment provides an external boundary condition for this relaxation process. Additionally, once transition analysis has occurred, it is possible to characterize when all internal state elements potentially make transitions and thus when they may sample internal wires. Just as with transition and value propagation, the result is a relationship between inputs and outputs of a component. For sampling analysis, it is possible to deduce the sampling behavior of inputs of a component from the sampling information for its outputs. The relaxation process for computing sampling information thus propagates in the opposite direction from that of transition information, but otherwise similarly starts with boundary information and propagates changes until no further changes occur.

At the termination of transition 612 and sampling 616 steps it is possible to characterize precisely which timing edges can result in transitions and/or sampling for each wire within the digital circuit. Signals which are combinationally derived from timing signals with known values often also carry knowledge about their precise values during some or all of the discrete time range. They similarly often are known to only be able to make one form of directed transition, either rising or falling, at some particular discrete time point. This information is relevant to understanding the behavior of edge-triggered state elements.

The final resynthesis step 618 involves the application of a number of circuit transformations to the original digital circuit design which have a number of effects. First, the internal clock VClk is introduced into the logic design 200 of the digital circuit. The internal clock signal is the main clock of the logic system 200. Further, in effect, all of the original environmental timing signals of the digital circuit are converted into data signals in the logic system 200. Finally, all of the state elements in the digital circuit are converted to use the internal clock signal as their clock, leaving the internal clock as the only clock signal of the transformed system. The state elements of the original digital circuit design are converted preferably into edge-triggered flip-flops and finite state machines, which generate control signals to the load enable terminals of the flip-flops. The information developed in the transition analysis step 612, value analysis step 614, and sampling analysis step 616 is used to define the operation of the finite state machines as it relates to the control of the flip-flops in response to the internal clock signal and the environmental timing signals. The finite state machines send load enable signals to the flip-flops when it is known that data inputs are correct based upon a routing and scheduling algorithm described in the U.S. patent application Ser. No. 08/344,723 filed Nov. 23, 1994 and entitled "Pipe-Lined Static Router and Scheduler for Configurable Logic System Performing Simultaneous Communications and Computations", incorporated herein by this reference. The scheduling algorithm essentially produces a load enable signal on a virtual clock cycle that is given by the maximum of the sum of data, value available time, and routing delays for each signal that can affect data input.

Single Flip-Flop Timing Resynthesis

FIG. 7A shows a simple edge-triggered flip-flop 710 which was a state element in the original digital circuit. Specifically, the edge-triggered flip-flop 710 receives some input signal at its input terminal D and some timing signal, such as an environmental clock signal BCLK at its clock input terminal. In response to a rising edge received into this clock terminal, the value held at the input terminal D is placed at the output terminal Q.

The timing resynthesis step converts this simple edge-triggered flip-flop 710 to the circuit shown in FIG. 7B. The new flip-flop is a load-enabled flip-flop and is clocked by the internal clock signal VClk. The enable signal of the converted flip-flop is generated by a finite state machine FSM. Specifically, the finite state machine monitors a synchronized version of the clock signal V_{C0} and asserts the enable signal to the enable input terminal E of the converted flip-flop 720 for exactly one cycle of the internal clock VClk in response to synchronizing signal V_{C0} transitions from 0 to 1. The finite state machine is programmed so that the enable signal is asserted on an internal clock signal cycle when the input IN is valid accounting for delays in the circuit that arise out of a need to route the signal IN on several VClk cycles from the place it is generated to its destination at the input of flip-flop 720. In a virtual wire systems signals are routed among multiple FPGAs on specific internal clock VClk cycles. The synchronizing signal V_{C0} is generated by a synchronizer SYNC in response to receiving the environmental timing signal BClk on the next or a following transition of the internal clock signal VClk. As a result, the circuit is functionally equivalent to the original circuit shown in FIG. 7A since the generation of the enable signal occurs in response to the environmental clock signal BClk each time a transition occurs. The circuit, however, is synchronous with the internal clock VClk.

In a digital circuit comprising combinational logic and a collection of flip-flops, all of which trigger off the same edge of a single clock, the basic timing resynthesis transformation, shown in FIG. 7B and described above, can be extended. All flip-flops are converted to load-enabled flip-flops and have their clock inputs connected to the

internal clock VClk. The load enable terminal E of each flip-flop is connected to enable signals generated by a shared finite state machine in an identical manner as illustrated above. The FSM can be distinct for each FPGA. The enables for each flip-flop will be produced to account for routing delays associated with each signal input to the flip-flops.

Timing Resynthesis for Domains for Multiple Clocks

FIG. 8A shows a circuit comprising three flip-flops 810–814 that are clocked by two environmental clock signals BClk0 and BClk1. For the purposes of this description, both environmental clock signals BClk0 and BClk1 are assumed to be phase-locked with respect to each other.

The transformed circuit is shown in FIG. 8B. It should be noted that the basic methodology of the transform is the same as described in relation to FIGS. 7A and 7B. The finite state machine FSM and the clock sampling circuitry SYNC1 and SYNC2 have been extended. As before, each flip-flop of the transformed circuit has been replaced with a load-enabled positive-edge triggered flip-flop 820–824 in the transformed circuit. The first environmental clock signal BClk0 and the second environmental clock signal BClk1 are synchronized to the internal clock by the first synchronizer SYNC0 and the second synchronizer SYNC1. The synchronizing signals V_{C00} and V_{C01} are generated by the synchronizers SYNC0 and SYNC1 to the finite state machine FSM. The finite state machine FSM watches for the synchronizing signals V_{C00} and V_{C01} and then produces a distinct load enable pulse C0-Rise, C0-Fall, C1-Rise for each timing edge on which the clocks BClk0 and BClk1 of the flip-flops 820–824 operate. The ordering of these load enable pulses is prespecified within a domain where there is a unique ordering of the edges of all phase-locked clocks. This unique ordering of clocks is specified by the user of the system. As with the single clock case shown in FIG. 7B, each of the enable pulses C0-Rise, C0-Fall, and C1-Rise is asserted for exactly one period of the internal clock VClk upon detection of the corresponding clock edge in FIG. 8B.

Multiple Clock Domains Resynthesis

FIG. 9A shows a collection of flip-flops 910–912 from the digital circuit having multiple clock domains. That is, the first clock signal CLK0 and the second clock signal CLK1 do not have a phase-locked relationship to each other, rather the clocks are asynchronous with respect to each other.

FIG. 9B shows the transformed circuit. A different finite state machine FSM0 and FSM1 is assigned to each domain. Specifically, a first finite state machine FSM0 is synchronized to the first environmental clock BClk0 to generate the load enable signal to the load enable terminal E0 of the first flip-flop 920. The second finite state machine FSM1 generates a load enable signal to E1 of the second flip-flop 922 in response to the second environmental clock signal BClk1. It should be noted, however, that although FSM0 and FSM1 operate independently of each other, each of whose sequences are initiated by separate signals V_{g00} and V_{g01}, and that although the first flip-flop 920 and the second flip-flop 922 work independently of each other, i.e., load enabled by different clock signals BClk0 and BClk1, the resulting system is a single-clock synchronous system with the internal clock VClk.

The relationship between the behavior of the first finite state machine FSM0 and the second finite state machine FSM1 of the two clock signal domains is related to the relationship between the domains themselves. When the two domains are asynchronous, the first finite state machine FSM0 and the second finite state machine FSM1 may operate simultaneously or non-simultaneously. When the two domains are non-overlapping, the first finite state machine FSM0 and the second finite state machine FSM1 never operate simultaneously since the edges within the domains are separated in time.

The simultaneity of operation of finite state machines that are asynchronous with respect to each other leaves two circuits which can not readily be transformed by timing resynthesis. A state element which can undergo transitions as a result of an edge produced from a combination of signals in asynchronously related domains can not be resynthesized. Such condition can rise if two asynchronous clocks are gated together and fed into the clock input of a flip-flop or if a state element with multiple clocks and/or asynchronous presets or clears is used as transition triggering inputs from distinct asynchronously related clock domains. Due to the non-simultaneous events and non-overlapping domains, the situations above are not problematic in the non-overlapping situation.

Gated Clock Transformations

Clock gating in the digital circuit provides additional control over the behavior of state elements by using combinational logic to compute the input to clock terminals. The timing resynthesis process transforms gated clock structures into functionally equivalent circuitry which has no clock gating. Generally, gated clock structures can be divided into two classes: simple gated clocks and complex gated clocks. The basis for this distinction lies in the behavior of the gated clocks as deduced from timing analysis. Previously, the terms timing signal and data signal were defined in the context of inputs and outputs to the digital circuit. A gated clock is a combinational function of both timing signals and data signals. The gated clock transition then controls the input of a state element. Data signals can either be external input data signals from the environment or internally generated data signals.

A simple gated clock has two properties:

1) at any discrete time it is possible for a simple gated clock to make a transition in at most one direction, stated differently, there is no discrete time at which the simple gated clock may sometime rise and sometime fall; and
2) only timing signals change at those discrete times at which state elements can change state.

A complex gated clock violates one of these two properties.

Simple Gated Clock Transformation

FIG. 10A shows a circuit that exhibits a simple gated clock behavior. FIG. 10B is a timing diagram showing transitions in the data signal and the gated clock signal as a function of the environmental clock signal EClk. Specifically, upon the falling edge of the environmental clock BClk, the gating flip-flop 1010 latches the control signal CTL received at its input D1 at its output terminal Q1. This is the data signal. The AND gate 1012 receives both the data signal and the environmental clock BClk. As a result, only when the environmental clock BClk goes high, does the gated-clock signal go high on the assumption that the data signal is also a logic high. Upon the rising edge of the gated clock, the second flip-flop 1014 places the input signal IN received at its D2 terminal to its output terminal Q2.

FIG. 10C shows the transformed circuit. Here, a finite state machine FSM receives a signal V_{GO} from the synchronizer SYNC upon receipt of the environmental clock BClk. The finite state machine FSM produces two output signals: C0-Fall which is active upon the falling edge of the environmental clock signal, and C0-Rise which is active upon the rising of the environmental clock signal BClk.

The transformed circuit functions as follows. On the first period of the internal clock VClk after the falling edge of the

environmental clock signal BClk, the first flip-flop 1016 places the value of the control signal received at its input terminal D1 to its output terminal Q1 upon the clocking of the internal clock signal VClk. This output of the first flip-flop 1016 appearing at terminal Q1 corresponds to the data signal in the original circuit. This data signal is then combined in an AND gate 1020 with the signal C0-Rise from the finite state machine FSM that is indicative of the rising edge of the environmental clock signal BClk. The output of the AND gate goes to the load enable terminal E2 of a second flip-flop 1018 which receives signal IN at its input terminal D2. Again, upon the receipt of this load enable and upon the next cycle of the internal clock VClk, the second flip-flop moves the value at its input terminal D2 to its output terminal Q2.

Complex Gated Clock Transformations

In the case of complex gated clock behavior, the factoring technique used for simple gated clock transformations is inadequate. Because data and clocks change simultaneously and/or the direction of a transition is not guaranteed, both the value of a gated clock prior to the transition time and the value of the gated clock after the transition time are needed. Using these two values, it can be determined whether a signal transition that should trigger a state change has occurred. One way to produce the post-transition value of data signals is to replicate the logic computing the signal and also replicate any flip-flops containing values from which the signal is computed and which may change state as a result of the transition. These replica flip-flops can be enabled with an early version of the control signal, thus causing them to take on a new state prior to the main transition. By this mechanism, pre- and post-transition values for signals needed for gated clocks can be produced.

An alternative way to get the two required values for the gated clock signal is to add a flip-flop to record the pre-transition state of the gated-clock and delay in time the update of the state element dependent on the gated clock. These two techniques have different overhead costs and the latter is only applicable if the output of the state element receiving the gated clock is not sampled at the time of the transition. The former always works but the latter generally has lower overhead when applicable.

FIG. 11A shows two cascaded edge-triggered flip-flops 1100 and 1112. This configuration is generally known as a frequency divider. The environmental clock signal BClk is received at the clock terminal of the first flip-flop 1110; and at the output Q2 of the second flip-flop 1112, a new clock signal is generated that has one-fourth the frequency of BClk. The divider of FIG. 11A operates as follows: In an initial state in which the output terminal Q1 of the first flip-flop 1110 is a 0 and the input terminal D1 of the flip-flop 1110 is a 1, receipt of the rising edge of the environmental timing signal BClk changes Q1 to a 1 and D1 converts to a 0. The conversion of Q1 from 0 to 1 functions as a gated clock to the clock input terminal of the second flip-flop 1112. The second flip-flop 1112 functions similarly, but since it is only clocked when Q1 of the first flip-flop 1110 changes from 0 to 1, but not 1 to 0, it changes with one-fourth the frequency of BClk.

FIG. 11B shows the transformed circuit of FIG. 11A. Here, a replica flip-flop 1120 has been added that essentially mimics the operation of the first flip-flop 1122. The replica flip-flop 1120, however, receives a pre-Clk-Rise control signal from the finite state machine FSM. More specifically, the finite state machine FSM responds to the synchronizing signal V_{GO} and the internal clock VClk and produces a pre-CLK-rise signal that is active just prior to the CLK-Rise signal, CLK-Rise being active in response to the rising edge of the environmental timing signal BClk. Assume the output terminal Q1 of the first flip-flop 1122 is initially at a 0 and the input terminal D1 of first flip-flop 1122 is a 1, the replica flip-flop 1120 is initially at a 0. Upon receipt of the pre-CLK-rise signal at the replica flip-flop load enable terminal ER, the output terminal QR of the replica flip-flop 1120 makes a transition from a 0 to a 1. Since Q1 is low and QR is high, an AND gate 1124 functioning as an edge detector generates a high signal. When the CLK-rise control signal from the finite state machine FSM is active in response to receipt of the rising edge of the environmental clock signal BClk, the output terminal Q1 of the first flip-flop 1122 is converted from a 0 to a 1. The enable terminal E2 of flip-flop 1126 also is high, causing the flip-flop to change state. On the next falling transition of Q1, the AND gate 1124 will produce 0 and flip-flop 1126 will not change state. Since the replica flip-flop 1120 provides a zero to the rising edge detector whenever the zero is present at the input terminal of the first flip-flop, the rising edge detector is enabled only every other transition of Q1.

Latch Resynthesis

Generally, latches are distinguished from flip-flops in that flip-flops are edge-triggered. That is, in response to receiving either a rising or falling edge of a clock signal, the flip-flop changes state. In contradistinction, a latch has two states. In an open state, the input signal received at a D terminal is simply transferred to an output terminal Q. In short, in an open condition, the output follows the input like a simple wire. When the latch is closed, the state of the output terminal Q is maintained or held independent of the input value at terminal D. A semantic characterization of such a latch is as follows. For an input D, output Q, a gate G, and a state S, Q=S. S=D if G=1. The latch is open when G=1 and closed when G=0.

Beginning with the simplest case, if the output of a latch is never sampled when the latch is closed, G=0, the latch is really just a wire. Latches with this characteristic may be used to provide extra hold time for a signal. For this sample latch, this would be true, if the set of discrete times at which the output of the latch is sampled, is equal to or a proper subset of the set of discrete times at which the gate signal G is known to have a value of 1. In this situation, the latch can be removed and replaced with a wire connecting the input and output signals.

In contrast, if the output of the latch is never sampled when the latch is open, the latch is equivalent to a flip-flop. The only value produced by the latch which is ever sampled is a value of the input D on the gate signal edge when the latch transitions from open to closed. This condition is true if the set of discrete times at which the output of the latch is sampled, is equal to, or a proper subset of the discrete times at which the gate signal G is known to have a value of 0. In this situation, the latch can be removed and replaced with an edge-triggered flip-flop.

As shown in FIG. 12, latches that are open when their gate signal G is high 1210 are converted to negative-edge triggered flip-flops 1212. Latches that are open when their gate signal G is low 1214 are converted to positive edge triggered flip-flops 1216.

Once the transition from the latch to the edge triggered flip-flop has been made, these new edge-triggered flip-flops are then further resynthesized by the timing resynthesis techniques described in connection with FIGS. 6–11. Therefore, after this further processing, both positive and negative edge-triggered flip-flops will be flip-flops clocked by the internal clock VClk. The resynthesized flip-flops will have an enable signal that is generated by a finite state machine in response to the particular environmental clock signal that gated the original latch element.

Referring to FIG. 13, in the condition in which the output of a given latch 1310 is sampled both when the latch might be open and might be closed, that latch can be converted to a flip-flop 1312, plus a multiplexor 1314 as shown in FIG. 13. There, when the gate signal G is low, the multiplexor 1314 selects the input signal to the input terminal D of the flip-flop 1312. On the rising edge of the gating signal, however, the input to the D terminal is latched at the output terminal Q. Also, at this point, the gating signal selects the second input to the multiplexor 1214. As with the case in FIG. 12, the result of the transformation in FIG. 13 is subjected to further resynthesis.

The transform of FIG. 13 may exhibit timing problems if the multiplexor is implemented in a technology that exhibits hazards, or output glitches. Output glitches can and could result in set up and hold time problems of the sampling state element. This transformation can therefore only be used when the output is never sampled at discrete times at which the clock may exhibit an edge. If the output is sampled both when the latch might be opened and closed and some sampling occurs on the edge of the gate signal, a final transformation is employed. A new clock signal is created which is phase-locked to the original clock signal and precedes it.

As shown in FIGS. 14A and 14B, the latch 1410 of FIG. 14A is replaced by a flip-flop which receives the phase-advanced clock indicated by the negative delay 1412 as shown in FIG. 14B. The state transition of the new flip-flop 1414 precedes a state transition of any circuits sampling the original output Q of the original latch 1410. If the latch is also sampled when it is open by signals occurring prior to the sampling edge, one of the prior techniques can be employed, either latch to wire or latch to flip-flop and multiplexor transforms of FIG. 13.

As shown in FIG. 14B, the negative delay 1412 represents a time-advanced copy of the clock CLK which is used to clock the flip-flop 1414. While negative-delays are unphysical, this structure can be processed by the timing resynthesis process with a distinct control signal generated by a finite state machine.

FIG. 15 shows a finite state machine FSM generating a pre-CLK-rise control signal one or more cycles of the internal clock VClk prior to the generation of the control signal, CLK-Rise. The control signal CLK-rise is generated in response to the rising edge of the environmental timing signal BClk. As a result, the input signal appearing at the D terminal of the flip-flop 1510 is transferred to the output terminal prior to the rising edge of the environmental clock signal BClk as signaled by the Clk-rise control signal. Subsequent elements can be then load enabled from the CLK-Rise signal generated by the finite state machine FSM. Here again, if the latch of the original digital circuit is sampled both when the latch is opened and closed, a multiplexor can be placed at the output Q of the flip-flop 1510.

Combinational Loop Transformations

Combinational loops with an even number of logic inversions around the loop are an implicit state element. An example is shown in FIG. 16A, this implicit state can be transformed into an explicit state element which is clocked by the virtual clock VClk by simply choosing a wire 1601 in the loop and inserting a flip-flop 1602 which is clocked by the virtual clock VClk as shown in FIG. 16B.

The addition of the flip-flop 2602 changes the timing characteristics of the loop. Additional virtual clock cycles are required for the values in the loop to settle into their final states.

Assume in FIG. 16B that all input values to the loop are ready by some virtual cycle V. In the absence of the flip-flop 1602, all outputs will become correct and stable after some delay period. With the flip-flop 1602, it is necessary to wait until the loop stabilizes and then wait for an additional virtual clock period during which the flip-flop value may change and subsequently change the loop outputs. Thus the outputs of the loop cannot be sampled until virtual cycle V+1.

If combinational cycles are nested, each can be broken by the insertion of a flip-flop as above. Nested loops may require up to 2^N clock cycles to settle, where N is the depth of the loop nesting and thus the number of flip-flops needed to break all loops.

RS Latch Transformations

RS latches 1710 are asynchronous state elements built from cross-coupled NOR or NAND gates 1712, as illustrated in FIG. 17A.

RS latches 1710 can be transformed based on the transformation for combinational cycles illustrated in FIGS. 16A and 16B. An alternative approach illustrated in FIG. 17B eliminates the combinational cycles associated with RS latches while also avoiding the extended settling time associated with the general combinational cycle transformation of FIG. 16B.

The circuit is FIG. 17B forces the outputs Q and $\bar{Q}$ of the RS latch 1710 combinatorially to their values for all input patterns except the one in which the latch maintains its state. For this pattern, the added flip-flop 1714 produces appropriate values on the outputs. Logic 1716 is provided to set the flip-flop 1714 into an appropriate state, based on the values of the inputs whenever an input pattern dictates a state change. When the latch 1710 is maintaining its state, the outputs will be stable so no propagation is required. Thus the outputs of the transformation are available with only a combinatorial delay.

A symmetrical transformation can be applied to latches produced from cross-coupled NOR gates.

Asynchronous Presets and Clears

Asynchronous presets and clears of state elements shown in FIG. 18A can be transformed in one of two ways. Each transformation relies on the fact that preset and clear signals R are always synchronized to the virtual clock, either because they are internally generated by circuitry which is transformed to be synchronous to the virtual clock or because they are external asynchronous signals which are explicitly synchronized using synchronizer circuitry.

The first transformation, shown in FIG. 18B, makes use of an asynchronous preset or clear on flip-flop 1808 in the FPGA, if such exists. The enable signal E which enables the resynthesized state element to undergo state changes is used to suppress/defer transitions on the preset or clear input R to eliminate race conditions arising from simultaneously clocking and clearing or presenting a state element.

The second transformation shown in FIG. 18C converts an asynchronous preset or clear R_V which has already been synchronized to the clock into a synchronous preset or clear. The enable signal E to the resynthesized state element must be modified to be enabled at any time at which a preset or clear transition might occur by gate 1810.

Returning to FIG. 6, the above described transformations of the timing resynthesis step 618 in combination of with the specification step 610, transition analysis 612, value analysis 614 and sampling analysis 616 enable conversion of a digital circuit description having some arbitrary clocking methodology to a single clock synchronous circuit. The result is a circuit which the state elements are edge-triggered flip-flops.

To generate the logic system 200 having the internal architecture shown in FIG. 4, this resynthesized circuit must now be compiled for and loaded into the configurable logic devices 410–416 by the host workstation 222.

FIG. 19 shows the complete compilation process performed by the host workstation 222 to translate the digital circuit description into the configuration data received by the configurable devices 214. More specifically, the input to a compiler running on the host workstation 222 is the digital circuit description in step 1610. This description is used to generate the resynthesized circuit as described above. The result is a logic netlist of the resynthesized circuit 1611. This includes the new circuit elements and the new VClk.

In step 1612, functional simulations of the transformed circuit can be performed. This step ensures that the resynthesized circuit netlist is the functional equivalent of the original digital circuit. It should be noted that the transformed circuit is also more amenable to computer-based simulations. All relevant timing information specifying the behavior of the timing signals including the timing relationship to each other is built into the resynthesized circuit yet the resynthesized circuit is synchronous with a single clock. Therefore, the resynthesized circuit could alternatively be used as the circuit specification for a computer simulation rather than the hardware based simulation on the configurable logic system. The resynthesized circuit is then partitioned 1613 into the logic partition blocks that can fit into the individual FPGAs of the array, see FIG. 2.

In the preferred embodiment of the present invention, techniques described in U.S. patent application Ser. No. 08/042,151, filed on Apr. 2, 1993, entitled Virtual Wires for Reconfigurable Logic System, which is incorporated herein by this reference, are implemented to better utilize pin resources by multiplexing global link transmission on the pins of the FPGAs across the interconnect. Additionally, as described in incorporated U.S. patent application Ser. No. 08/344,723, filed on Nov. 23, 1994, entitled Pipe-Lined Static Router and Scheduler for Configurable Logic System Performing Simultaneous Communication and Computation, signal routing is scheduled so that logic computation and global link transmission through the interconnect happen simultaneously.

Specifically, because a combinatorial signal may pass through several FPGA partitions as global links during an emulated clock cycle, all signals will not be ready to schedule at the same time. This is best solved by performing a dependency analysis, step 1614 on global links that leave a logic partition block. To determine dependencies, the partition circuit is analyzed by backtracing from partition outputs, either output global links or output signals to the target system, to determine on which partition inputs, either input links or input signals from the target system, the outputs depend. In backtracing, it is assumed that all outputs depend on all inputs for gate library parts, and no outputs depend on any inputs for latch or register library parts. If there are no combinatorial loops that cross partition boundaries, this analysis produces a directed acyclic graph, used by a global router. If there are combinatorial loops, then the loops can be hardwired or implemented in a single FPGA. Loops can also be broken by inserting a flip-flop into the loop and allowing enough virtual cycles for signal values to settle to a stable state in the flip-flop.

Individual FPGA partitions must be placed into specific FPGAs (step 1616). An ideal placement minimizes system communication, requiring fewer virtual wire cycles to transfer information. A preferred embodiment first makes a random placement followed by cost-reduction swaps and then optimizes with simulated annealing. During global routing (step 1618), each global link is scheduled to be transferred across the interconnect during a particular period of the pipe-line clock. This step is discussed more completely in the incorporated U.S. patent application Ser. No. 08/344,723, Pipe-Lined Static Router and Scheduler for Configurable Logic System Performing Simultaneous Communication and Computation.

Once global routing is completed, appropriately-sized multiplexors or shift loops, pipeline registers, and associated logic such as the finite state machines that control both the design circuit elements and the multiplexors and pipeline registers are added to each partition to complete the internal configuration of each FPGA chip 22 (steps 1620). See specifically, incorporated U.S. patent application Ser. No. 08/042,151, Virtual Wires for Reconfigurable Logic System. At this point, there is one netlist for each configurable logic device 214 or FPGA chip. These FPGA netlists are then processed in the vender-specific FPGA place-and-route software (step 1622) to produce configuration bit streams (step 1624). Technically, there is no additional hardware support for the multiplexing logic which time-multiplex the global links through the interconnect: the array of configurable logic is itself configured to provide the support. The necessary "hardware" is compiled directly into the configuration of the FPGA chip 214. Some hardware support in the form of special logic for synchronizers to synchronize the external clocks to the internal VClk is recommended.

While this invention has been particularly shown and describe with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined by the appended claims. For example, it is not a strict necessity that the internal clock signal VClk be distributed directly to the sequential logic elements. Preferably it reaches each element at substantially the same time. In some larger networks, therefore, some delay may be preferable to delay tune the circuit for propagation delays.

We claim:

1. A method of configuring a configurable logic system to operate in an environment, the logic system generating output signals to the environment in response to at least one environmental timing signal and environmental data signals provided from the environment, the method comprising:

defining an internal clock signal;

configuring the logic system to perform logic operations for generating the output signals in response to the environmental data signals and the internal clock signal; and

configuring the logic system to have a controller for coordinating operation of the logic operations in response to the internal clock signal and the environmental timing signal.

2. A method of configuring as described in claim 1, further comprising configuring the controller to comprise a synchronizer for sampling the environmental timing signal in response to the internal clock signal.

3. A method of configuring as described in claim 2, further comprising configuring the controller to further comprise a finite state machine for generating control signals to control the logic operations in response to the sampled environmental timing signal.

4. A method of configuring as described in claim 1, further comprising configuring the logic system to have combinational logic and sequential logic to perform the logic operations.

5. A method of configuring as described in claim 4, further comprising configuring the controller to comprise a finite state machine for generating control signals to the sequential logic in response to the environmental timing signal and the internal clock signal.

6. A method of configuring as described in claim 5, further comprising configuring the sequential logic to comprise flip-flops receiving the internal clock signal at a clock input and the control signals at a latch enable input.

7. A method of configuring as described in claim 1, wherein the logic system comprises at least one field programmable gate array.

8. A method of configuring as described in claim 1, wherein the logic system comprises a plurality of configurable logic devices electrically connected via an interconnect for transmitting signals between the chips.

9. A method of configuring as described in claim 8, wherein the interconnect comprises cross bar chips.

10. A method as configuring as described in claim 8, wherein the interconnect utilizes a direct-connect topology.

11. A method of configuring as described in claim 10, wherein the interconnect includes buses.

12. A method of configuring as described in claim 1, further comprising configuring the controller to dictate setup and hold times of signals to the environment.

13. A method of configuring as described in claim 1, further comprising configuring the controller to dictate sampling times of the environmental data signals.

14. A method for converting a digital circuit design into a new circuit that is substantially functionally equivalent to the digital circuit design, the digital circuit design and the new circuit being adapted to operate in an environment in response to at least one environmental timing signal and environmental data signals and providing output data signals to the environment, the method comprising:

defining an internal clock signal; and

resynthesizing sequential logic elements in the digital circuit design that are clocked by the environmental timing signal to sequential logic elements in the new circuit that are clocked by the internal clock signal.

15. A method as claimed in claim 14, wherein the resynthesized sequential logic elements of the new circuit are load enabled in response to the environmental timing signal.

16. A method as claimed in claim 14, wherein the internal clock signal has a substantially higher frequency than the environmental timing signal.

17. A method as claimed in claim 14, wherein a frequency of the internal clock signal is at least four times higher than a frequency of the environmental timing signal.

18. A method as claimed in claim 14, further comprising resynthesizing flip-flops in the digital circuit design that are clocked by the environmental timing signal to flip-flops in the new circuit that are clocked by the internal clock signal.

19. A method as claimed in claim 14, further comprising resynthesizing flip-flops in the digital circuit design that are clocked by the environmental timing signal to flip-flops in the new circuit that are clocked by the internal clock signal and load enabled in response to the environmental timing signal.

20. A method as claimed in claim 14, further comprising resynthesizing flip-flops in the digital circuit design that are clocked by the environmental timing signal to flip-flops in the new circuit that are clocked by the internal clock signal and load enabled by control signals generated by finite state machines operating in response to the environmental timing signal.

21. A method as claimed in claim 14, further comprising resynthesizing latches in the digital circuit design that are gated by the environmental timing signal to flip-flops in the new circuit that are clocked by the internal clock signal.

22. A method as claimed in claim 14, further comprising resynthesizing latches in the digital circuit design that are gated by the environmental timing signal to flip-flops in the new circuit that are clocked by the internal clock signal and load enabled in response to the environmental timing signal.

23. A method as claimed in claim 14, further comprising resynthesizing latches in the digital circuit design that are gated by the environmental timing signal to flip-flops in the new circuit that are clocked by the internal clock signal and load enabled by control signals generated by finite state machines operating in response to the environmental timing signal.

24. A method as claimed in claim 14, further comprising performing a simulation of the new circuit.

25. A method as claimed in claim 14, further comprising resynthesizing latches in the digital circuit design that are gated by the environmental timing signal to cascade-connected flip-flops and multiplexers, the multiplexers receiving select signals derived from the environmental timing signal.

26. A method as claimed in claim 25, wherein the select signals received by the multiplexers are generated by a finite state machine controller.

27. A logic system for generating output signals to an environment in response to at least one environmental timing signal and environmental data signals provided from the environment, the logic system comprising:

an internal clock for generating an internal clock signal for the logic system;

logic means for generating the output signals in response to the environmental data signals; and

controller means for coordinating operation of the logic means in response to the internal clock signal and the environmental timing signal.

28. A logic system for generating output signals to an environment in response to at least one environmental timing signal and environmental data signals provided from the environment, the logic system comprising:

an internal clock for generating an internal clock signal for the logic system;

at least one configurable logic device including:

logic which generates the output signals in response to the environmental data signals and the internal clock signal; and

a controller which coordinates operation of the logic in response to the internal clock signal and the environmental timing signal.

29. A logic system as described in claim 28, wherein the controller comprises a synchronizer for sampling the environmental timing signal in response to the internal clock signal.

30. A logic system as described in claim 29, wherein the synchronizer is constructed from non-programmable logic.

31. A logic system as described in claim 29, wherein the controller further comprises a finite state machine for generating control signals to the combinational logic in response to the sampled environmental timing signal.

32. A logic system as described in claim 28, wherein the logic comprises combinational logic and sequential logic.

33. A logic system as described in claim 32, wherein the controller comprises a finite state machine for generating control signals to the sequential logic in response to the environmental timing signal and the internal clock signal.

34. A logic system as described in claim 33, wherein the sequential logic comprises flip-flops receiving the internal clock signal at a clock input and the control signals at a latch enable input.

35. A logic system as described in claim 28, wherein the at least one configurable logic device comprises at least one field programmable gate array.

36. A logic system as described in claim 28, further comprising an interconnect for transmitting signals between plural configurable logic devices.

37. A configurable logic system, comprising:

at least one configurable logic device;

an interconnect providing connections between the logic device and an environment to convey output data signals from the configurable logic device and at least one environmental timing signal and environmental data signals from the environment; and

a configurer for programming the configurable logic device to synchronize the environmental timing signal to an internal clock signal of the logic system.

38. A configurable logic system as described in claim 37, wherein the configurer converts a digital circuit design into a new circuit that is substantially functionally equivalent to the digital circuit design, and programs the at least one configurable logic device with the new circuit.

39. A configurable logic system as described in claim 38, wherein the configurer converts the digital circuit design into the new circuit by resynthesizing sequential logic elements in the digital circuit design that operate in response to the environmental timing signal to operate in response to the internal clock signal in the new circuit.

40. A configurable logic system as described in claim 37, wherein the configurer programs the configurable logic device to have logic and a controller for coordinating operation of the logic in response to the internal clock signal and the environmental timing signal.

41. A configurable logic system as described in claim 40, wherein the configurer programs the controller to include a synchronizer for sampling the environmental timing signal in response to the internal clock signal.

42. A configurable logic system as described in claim 41, wherein the configurer programs the controller to include a finite state machine for generating control signals to the logic in response to the sampled environmental timing signal.

43. A configurable logic system as described in claim 41, wherein the configurer programs the logic to include combinational logic and sequential logic.

44. A configurable logic system as described in claim 41, wherein the configurer programs the controller to include a finite state machine for generating control signals to the sequential logic in response to the environmental timing signal and the internal clock signal.

45. A configurable logic system as described in claim 37, wherein the configurer programs the configurable logic device to include flip-flops that are clocked by the internal clock signal and load enabled in response to the environmental timing signal.

46. A configurable logic system as described in claim 37, wherein the configurer programs the configurable logic device to include:

flip-flops that are clocked by the clock signal and load enabled by control signals; and

finite state machines generating the control signals in response to the environmental timing signal.

47. A configurable logic system as described in claim 37, wherein the environment is a cycle simulation.

48. A configurable logic system as described in claim 37, wherein the environment is a hardware system.

49. A configurable logic system as described in claim 48, wherein the configurable logic system is a logic emulator.

50. A configurable logic system as described in claim 37, wherein the logic system is a simulation accelerator.

* * * * *

EXHIBIT C

US005761484A

United States Patent [19]

Agarwal et al.

[11] Patent Number: 5,761,484

[45] Date of Patent: *Jun. 2, 1998

[54] VIRTUAL INTERCONNECTIONS FOR RECONFIGURABLE LOGIC SYSTEMS

[75] Inventors: Anant Agarwal, Framingham; Jonathan Babb; Russell Tessier, both of Cambridge, all of Mass.

[73] Assignee: Massachusetts Institute of Technology, Cambridge, Mass.

[*] Notice: The term of this patent shall not extend beyond the expiration date of Pat. No. 5,596,742.

[21] Appl. No.: 530,323

[22] PCT Filed: Apr. 1, 1994

[86] PCT No.: PCT/US94/03620

§ 371 Date: Sep. 28, 1995

§ 102(e) Date: Sep. 28, 1995

[87] PCT Pub. No.: WO94/23389

PCT Pub. Date: Oct. 13, 1994

[51] Int. Cl.^{6} G06F 17/50

[52] U.S. Cl. 395/500; 364/488; 364/489

[58] Field of Search 364/488, 489, 364/490, 491, 578; 395/500

[56] References Cited

U.S. PATENT DOCUMENTS

4,495,590	1/1985	Mitchell, Jr.	364/716
4,506,341	3/1985	Kalter et al.	364/786
4,697,241	9/1987	Lavi	364/488
4,901,259	2/1990	Watkins	364/578
5,109,353	4/1992	Sample et al.	364/578
5,283,900	2/1994	Frankel et al.	395/700
5,442,306	8/1995	Woo	326/39
5,444,394	8/1995	Watson et al.	326/45
5,452,231	9/1995	Butts et al.	364/489
5,473,266	12/1995	Ahanin et al.	326/41
5,475,830	12/1995	Chen et al.	395/500
5,483,178	1/1996	Costello et al.	326/41
5,485,103	1/1996	Pedersen et al.	326/41
5,513,338	4/1996	Alexander et al.	395/500
5,572,710	11/1996	Asano et al.	395/500
5,596,742	1/1997	Agarwal et al.	395/500

FOREIGN PATENT DOCUMENTS

0410502A2	1/1991	European Pat. Off. .
90/04233	4/1990	WIPO .

OTHER PUBLICATIONS

Murgai et al., "Logic Synthesis for Programmable Gate Arrays," *IEEE 27th ACM/IEEE Design Automation Conference*, (1990) pp. 620–625.

Singh et al. "Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed," *IEEE 1991 Custom Integrated Circuits Conference*, (1991), pp. 6.1.1–6.1.6.

Wirbenberg, R.C., "Three Newcomers Stir Up Hardware Accelerator Market," Electronic Design, Dec. 29, 1986, pp. 22–23.

Bursky, D., "Fast simulator expands to mimic 4 million gates," *Electronic Design*, pp. 1–5, (Dec. 1986).

Lavi, Y., "The Supersim—An Ultrafast Hardware Logic Simulator," IFIP Workshop on CAD Engines, Tokyo (Jun. 6–9, 1987), No Page #s.

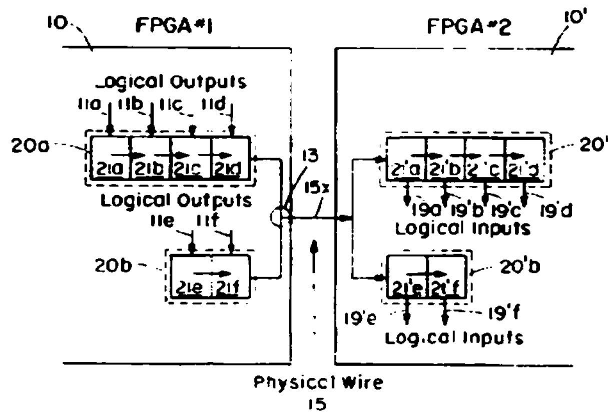
(List continued on next page.)

Primary Examiner—Jacques Louis-Jacques
Assistant Examiner—Leigh Marie Garbowski
Attorney, Agent, or Firm—Hamilton, Brook, Smith & Reynolds, P.C.

[57] ABSTRACT

A compilation technique overcomes device pin limitations using virtual interconnections. Virtual interconnections overcome pin limitations by intelligently multiplexing each physical wire among multiple logical wires and pipelining these connections at the maximum clocking frequency. Virtual interconnections increase usable bandwidth and relax the absolute limits imposed on gate utilization in logic emulation systems employing Field Programmable Gate Arrays (FPGAs). A "softwire" compiler utilizes static routing and relies on minimal hardware support. The technique can be applied to any topology and FPGA device.

39 Claims, 6 Drawing Sheets



OTHER PUBLICATIONS

Bertsekas, D., et al., *Data Networks*, pp. 8–14, 91–95 and 403–405 (Prentice Hall 1987).

Rose, J., et al., "Architecture of Field–Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuits*, No. 5, (Oct. 1990), pp. 1217–1225.

Babb, J., et al., "Virtual Wires: Overcoming Pin Limitations in FPGA–based Logic Emulators," Proceedings IEEE Workshop on FPGAs for Custom Computing Machines, (Apr. 5, 1993), pp. 142–151.

Hey, A., "Supercomputing with Transputers—Past, Present and Future," *Computer Architecture News*, vol. 18, No. 3, (Sep. 1990), pp. 479–489.

Agrawal, O., "Filed Programmable Gate Arrays (FPGAs) Provide ASIC System Designers Control of Their Design Destiny," Electro Conference Record, vol. 15, (May 1990), pp. 353–161.

Van Den Bout, D.E., et al., AnyBoard: An FPGA–Based Reconfigurable System *IEEE Computer Society*, (Sep. 1992), pp. 21–30.

XILINX: The Programmable Gate Array Design Handbook, (1986), pp. 1–9 to 1–14.

Wei, Yen–Chuen, et al., "Multiple–Level Partitioning: An Application to the Very Large–Scale Hardware Simulator," *IEEE*, vol. 26, No. 5, (May 1991), pp. 706–716.

Dally, "Virtual–Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, No. 2 (Mar. 1992), pp. 194–205.

Bursky, "Using Antifuse Programming For Gate–Arrayed Density and Flexibility, An FPGAs Family Also Delivers Masked–Array Performance, FPGAs Mirror Masked Gate–Array Architecture," *Electronic Design*, (Nov. 21, 1991), pp. 63–70.

Kermani et al., "Virtual Cut–Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, (Oct. 1979), pp. 267–286.

Khan, et al., "FPGA Architectures for ASIC Hardware Emulators," 1993 ASIC Conference and Exhibit, pp. 336–340.

Walters, "Reprogrammable Hardware Emulation for ASICs Makes Thorough Design Verification Practical," Compcon Spring '90, pp. 484–486.

Deiss, Stephen, R., "Connectionism without the Connections," *IEEE*, (1984), pp. 1217–1221.

Dominguez–Castro, R., et al., "Architectures and Building Blocks for CMOS VLSI Analog Neural Programmable Optimizers," *IEEE*, (1992), pp. 1525–1528.

Fornaciari, William, et al., "An Automatic VLSI Implementation of Hopfield ANNs," *IEEE*, (1995), pp. 499–502.

Bailey, Jim, et al., "Why VLSI Implementations of Associative VLCNs Require Connection Multiplexing," pp. II–173–II–180, No Date.

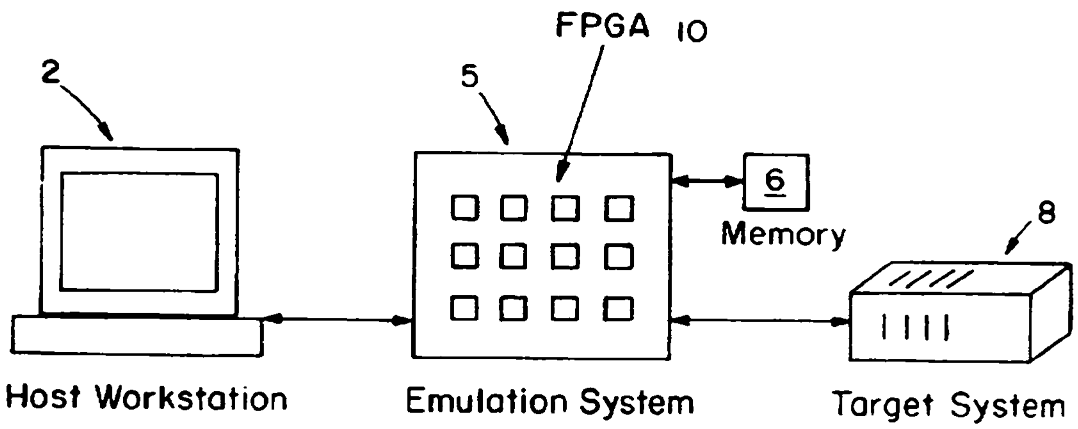


FIG. 1 (Prior Art)

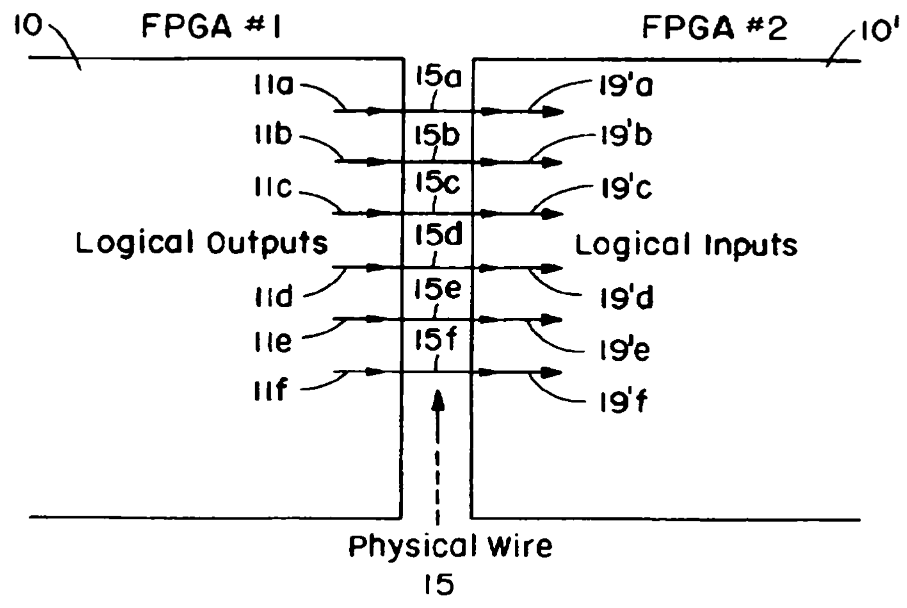


FIG. 2 (Prior Art)

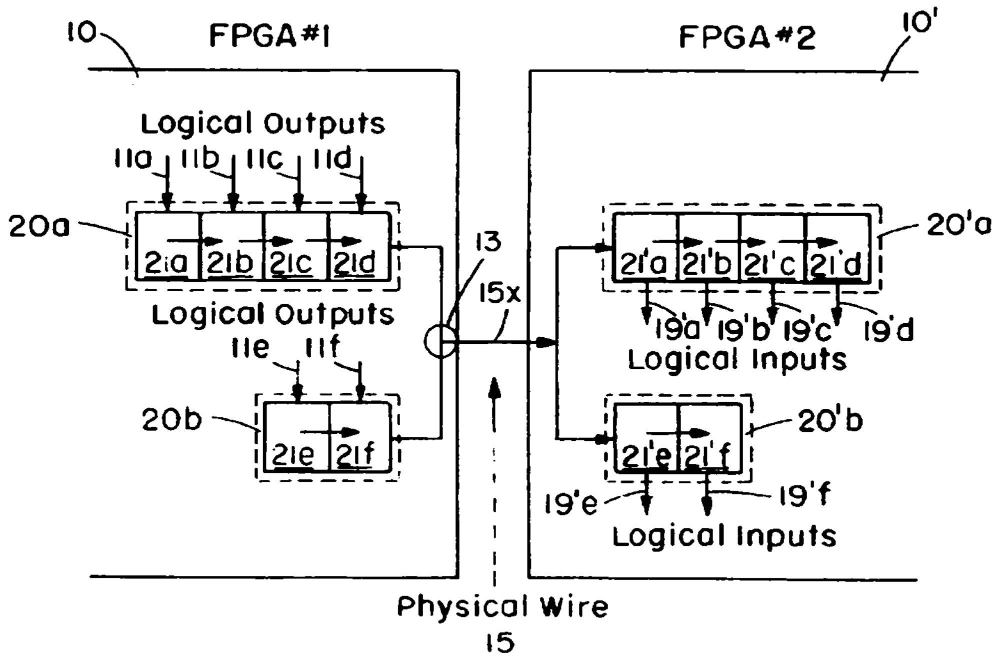


FIG. 3

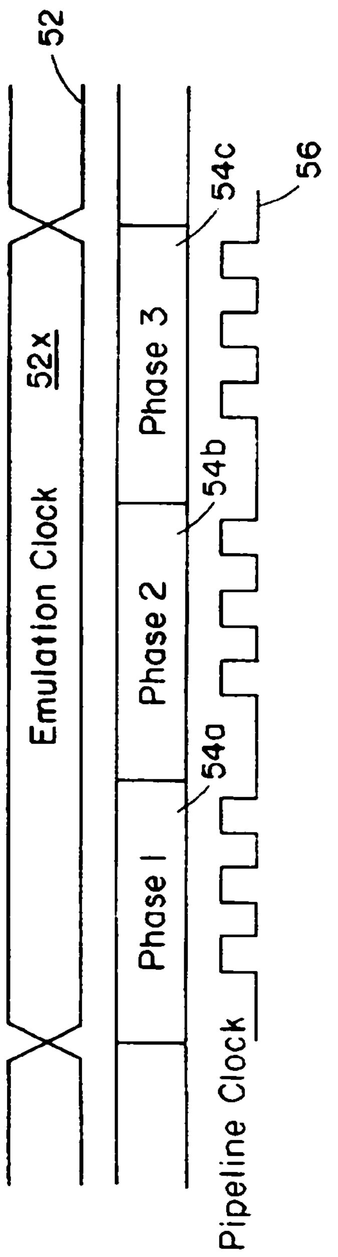

52
Emulation Clock
52x
Phase 1
Phase 2
Phase 3
54a
54b
54c
56
Pipeline Clock

FIG. 4

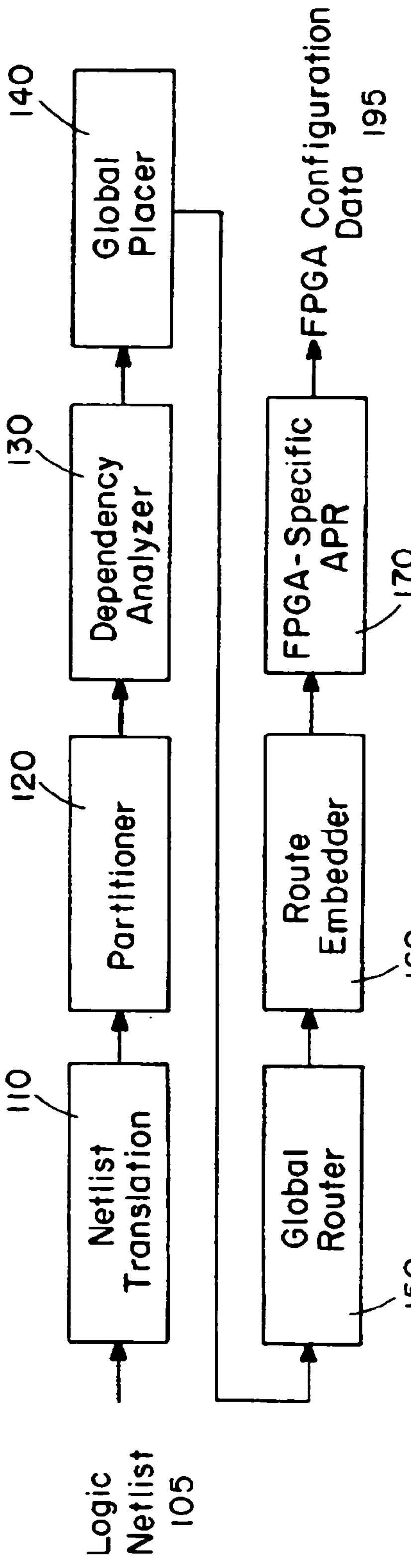

Logic Netlist 105
110
Netlist Translation
120
Partitioner
130
Dependency Analyzer
140
Global Placer
150
Global Router
160
Route Embedder
170
FPGA-Specific APR
FPGA Configuration Data 195

FIG. 5

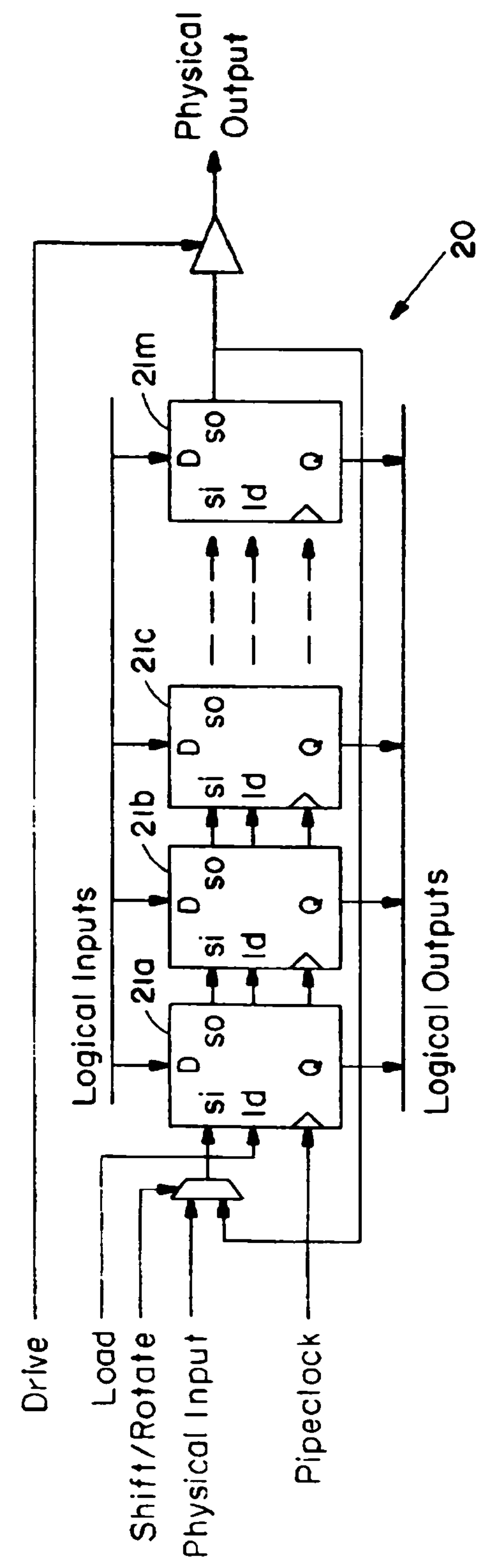

Physical Output
20
21m
21c
21b
21a
si
ld
D
so
Q
Logical Inputs
Logical Outputs
Drive
Load
Shift/Rotate
Physical Input
Pipeclock

FIG. 6

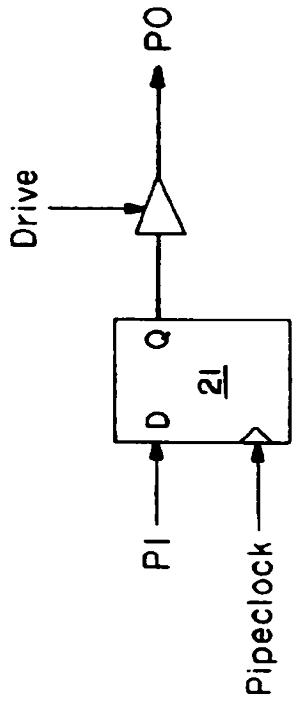

PO
Drive
Q
21
D
PI
Pipeclock

FIG. 7

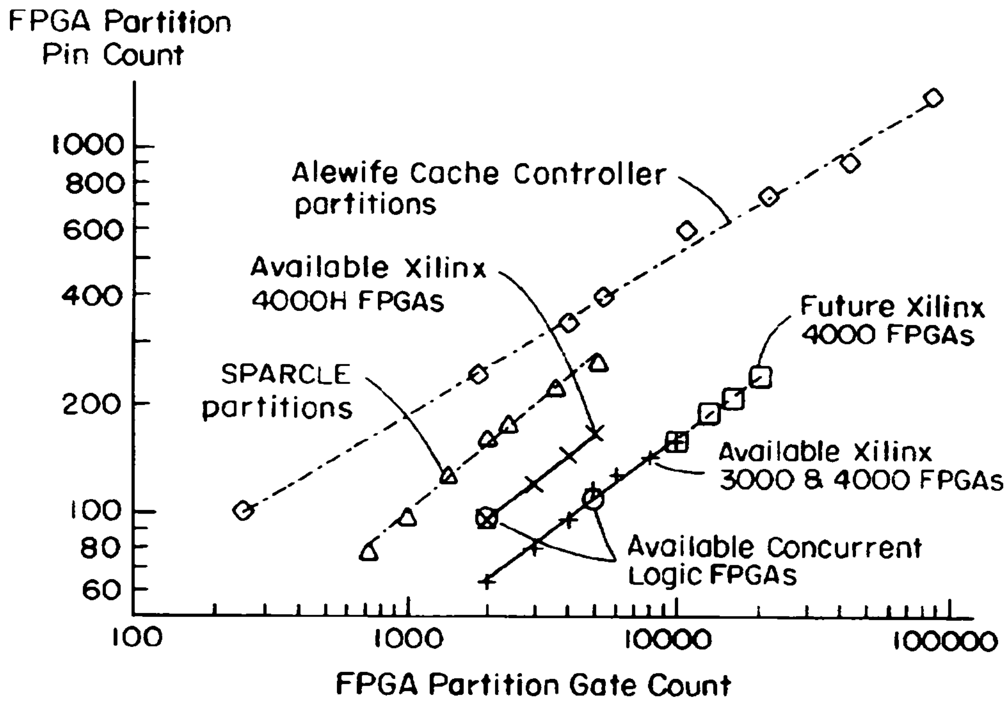

FPGA Partition Pin Count
1000
800
600
400
200
100
80
60
100
1000
10000
100000
FPGA Partition Gate Count
Alewife Cache Controller partitions
Available Xilinx 4000H FPGAs
SPARCLE partitions
Future Xilinx 4000 FPGAs
Available Xilinx 3000 & 4000 FPGAs
Available Concurrent Logic FPGAs

FIG. 8

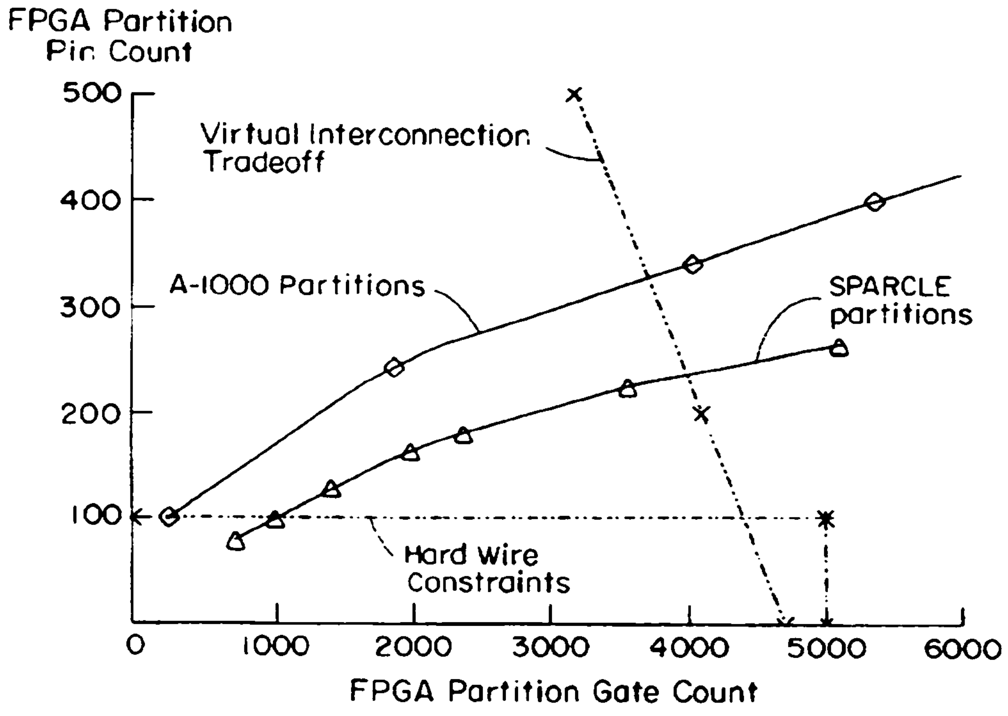

FPGA Partition Pin Count
500
400
300
200
100
0
1000
2000
3000
4000
5000
6000
FPGA Partition Gate Count
Virtual Interconnection Tradeoff
A-1000 Partitions
SPARCLE partitions
Hard Wire Constraints

FIG. 9

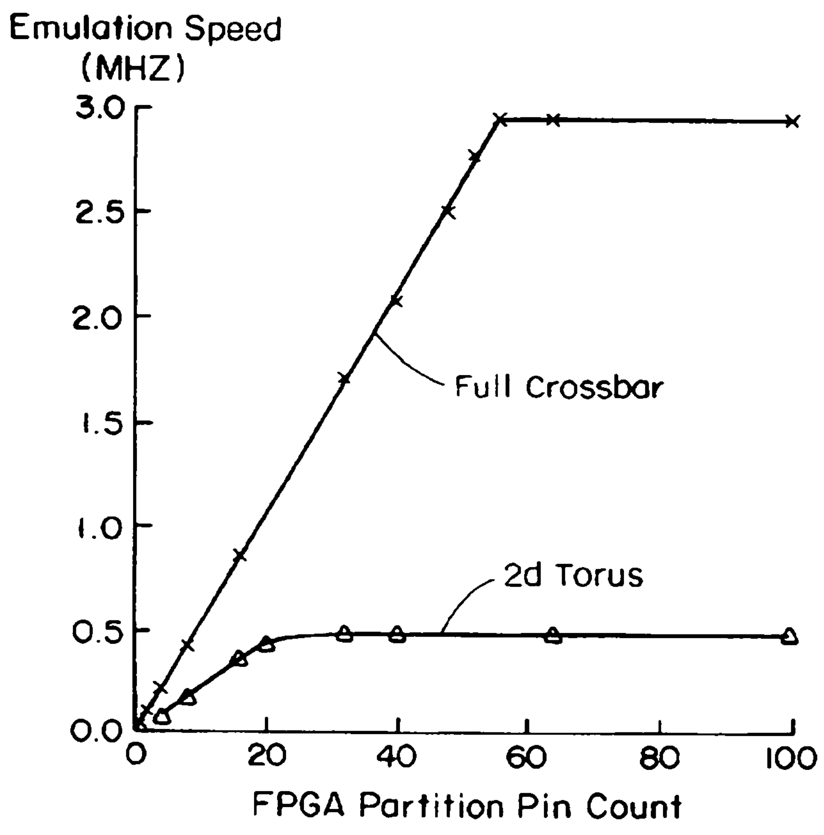

Emulation Speed
(MHZ)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
FPGA Partition Pin Count
Full Crossbar
2d Torus

FIG. 10

VIRTUAL INTERCONNECTIONS FOR RECONFIGURABLE LOGIC SYSTEMS

RELATED APPLICATIONS

This application is the U.S. National phase of International Application No. PCT/US94/03620, filed Apr. 1, 1994 which claimed priority to U.S. Ser. No. 08/042,151, filed Apr. 2, 1993, now U.S. Pat. No. 5,596,742; the teachings of which are incorporated herein by reference in their entirety.

BACKGROUND OF THE INVENTION

Field Programmable Gate Array (FPGA) based logic emulators are capable of emulating complex logic designs at clock speeds four to six orders of magnitude faster than even an accelerated software simulator. Once configured, an FPGA-based emulator is a heterogeneous network of special purpose processors, each FPGA processor being specifically designed to cooperatively execute a partition of the overall simulated circuit. As parallel processors, these emulators are characterized by their interconnection topology (network), target FPGA (processor), and supporting software (compiler). The interconnection topology describes the arrangement of FPGA devices and routing resources (i.e. full crossbar, two dimension mesh, etc). Important target FPGA properties include gate count (computational resources), pin count (communication resources), and mapping efficiency. Supporting software is extensive, combining netlist translators, logic optimizers, technology mappers, global and FPGA-specific partitioners, placers, and routers.

FPGA-based logic emulation systems have been developed for design complexity ranging from several thousand to several million gates. Typically, the software for these system is considered the most complex component. Emulation systems have been developed that interconnect FPGAs in a two-dimensional mesh and in a partial crossbar topology. In addition, a hierarchical approach to interconnection has been developed. Another approach uses a combination of nearest neighbor and crossbar interconnections. Logic partitions are typically hardwired to FPGAs following partition placement.

Statically routed networks can be used whenever communication can be predetermined. Static refers to the fact that all data movement can be determined and optimized at compile-time. This mechanism has been used in scheduling real-time communication in a multiprocessor environment. Other related uses of static routing include FPGA-based systolic arrays and in the very large simulation subsystem (VLSS), a massively parallel simulation engine which uses time-division multiplexing to stagger logic evaluation.

In prior systems, circuit switching techniques are used to provide output signals from one chip to another chip. A given output pin of one chip can be directly connected to a given input pin of another chip or provided during a dedicated time slot over a bus. The entire path of the signal through the bus is dedicated, using assigned bus pins and time slots to provide a direct connection during any time slot. A full resource is thus used to transmit the signal from the output chip to the input chip. An example of such a prior art system is discussed in Van Den Bout, *AnyBoard: An FPGA-Based Reconfigurable System*, IEEE Design and Test of Computers (Sept. 1992), pps. 21–30.

SUMMARY OF THE INVENTION

Existing FPGA-based logic emulators suffer from limited inter-chip communication bandwidth, resulting in low gate

utilization (10 to 20 percent). This resource imbalance increases the number of chips needed to emulate a particular logic design and thereby decreases emulation speed, because signals must cross more chip boundaries, and increases system cost. Prior art emulators only use a fraction of potential communication bandwidth because the prior art emulators dedicate each FPGA pin (physical wire) to a single emulated signal (logical wire). These logical wires are not active simultaneously and are only switched at emulation clock speeds.

A preferred embodiment of the invention presents a compilation technique to overcome device pin limitations using virtual interconnections. This method can be applied to any topology and FPGA device, although some benefit substantially more than others. Although a preferred embodiment of the invention focuses on logic emulation, the technique of virtual interconnections is also applicable to other areas of reconfigurable logic. Such reconfigurable logic systems (RLS) include, but are not limited to, simulation acceleration systems, rapid prototyping systems, multiple FPGA systems and virtual computing systems.

Virtual interconnections overcome pin limitations by intelligently multiplexing each physical wire among multiple logical interconnections and pipelining these connections at the maximum clocking frequency of the FPGA. A virtual interconnections represents a connection from a logical output on one FPGA to a logical input on another FPGA. Virtual interconnections not only increase usable bandwidth, but also relax the absolute limits imposed on gate utilization. The resulting improvement in bandwidth reduces the need for global interconnect, allowing effective use of low dimension inter-chip connections (such as nearest-neighbor). In a preferred embodiment, a "softwire" compiler utilizes static routing and relies on minimal hardware support. Virtual interconnections can increase FPGA gate utilization beyond 80% without a significant slowdown in emulation speed.

In a preferred embodiment of the invention, a FPGA logic emulation system comprises a plurality of FPGA modules. Each module is preferably a chip having a number of pins for communicating signals between chips. There are also interchip connections between the FPGA pins. In addition, a software or hardware compiler programs each FPGA chip to emulate a partition of an emulated circuit with interconnections between partitions of the emulated circuit being provided through FPGA pins and interchip connections. A partition of the emulated circuit has a number of interconnections to other partitions that exceed the number of pins on the FPGA chip. The chip is programmed to communicate through virtual interconnections in a time-multiplexed fashion through the pins. The inter-chip communications include interconnections which extend through the intermediate FPGA chips.

The FPGA chips may comprise gates that are programmed to serve as a multiplexer for communicating through the virtual interconnections. Alternatively, the FPGA chips may comprise hardwire multiplexers that are separate from the programmable gates. The interconnections may be point-to-point between pins, over a bus, or other interconnection networks. The pins of the FPGA chips may be directly connected to pins of other FPGA chips, where signals between the chips are routed through intermediate FPGAs. The FPGA chips may also be programmed to operate in phases within an emulation clock cycle with interchip communications being performed within each phase.

The compiler may optimize partition selection and phase division of an emulated circuit based on interpartition dependencies.

Data may also be accessed from memory elements external to the FPGAs during each phase by multiplexing the data on the virtual interconnections.

In a preferred embodiment of the invention, the FPGA chips comprise logic cells as an array of gates, shift registers, and several multiplexers. The gates are programmable to emulate a logic circuit. Each shift register receives plural outputs from the program gate array and communicates the outputs through a single pin in a multiplexed fashion. Some fraction of the gates in an FPGA chip may be programmed to serve as shift registers and multiplexer for communicating through virtual connections.

In a preferred embodiment of the invention, a compiler configures a FPGA logic emulation system using a partitioner for partitioning an emulated logic circuit and a programming mechanism for programming each FPGA to emulate a partition of an emulated circuit. The partitions are to be programmed into individual FPGA chips. The compiler produces virtual interconnections between partitions of the emulated circuit that correspond to one or more common pins with signals along the virtual interconnections being time-multiplexed through the common pin.

The compiler may comprise a dependency analyzer and a divider for dividing an emulation clock into phases, the phase division being a function of partition dependencies and memory assignments. During the phases, program logic functions are performed and signals are transmitted between the FPGA chips. The compiler may also comprise a router for programming the FPGA chips to route signals between chips through intermediate chips. In particular, the routed signals are virtual interconnections.

Results from compiling two complex designs, the 18K gate SPARCLE microprocessor and the 86K gate Alewife Cache Compiler (A-1000), show that the use of virtual interconnections decreases FPGA chip count by a factor of 3 for SPARCLE and 10 for the A-1000, assuming a crossbar interconnect. With virtual interconnections, a two dimensional torus interconnect can be used for only a small increase in chip count (17 percent for the A-1000 and 0 percent for SPARCLE). Without virtual interconnections, the cost of replacing the full crossbar with a torus interconnect is over 300 percent for SPARCLE, and virtually impossible for the A-1000. Emulation speeds are comparable with the no virtual interconnections case, ranging from 2 to 8 MHZ for SPARCLE and 1 to 3 MHZ for the A-1000. Neither design was bandwidth limited, but rather constrained by its critical path. With virtual interconnections, use of a lower dimension network reduces emulation speed proportional to the network diameter; a factor of 2 for SPARCLE and 6 for the A-1000 on a two dimensional torus.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other features of the invention, including various novel details of construction and combinations of parts, will now be more particularly described with reference to the accompanying drawings and pointed out in the claims. It will be understood that the particular virtual interconnection technique embodying the invention is shown by way of illustration only and not as a limitation of the invention. The principles and features of this invention may be employed in varied and numerous embodiments without departing from the scope of the invention.

FIG. 1 is a block diagram of a typical prior art logic emulation system.

FIG. 2 is a block diagram of a prior art hardwire interconnect system between Field Programmable Gate Arrays (FPGA) 10 of FIG. 1.

FIG. 3 is a block diagram of a virtual interconnection interconnect system between FPGAs 10 of FIG. 1.

FIG. 4 is a graphical representation of an emulation phase clocking scheme.

FIG. 5 is a flowchart of a preferred software compiler.

FIG. 6 is a block diagram of a preferred shift register or shift loop architecture.

FIG. 7 is a block diagram of the intermediate hop, single bit, pipeline stage of FIG. 6.

FIG. 8 is a graph illustrating pin count as a function of FPGA partition size.

FIG. 9 is a graph illustrating a determination of optimal partition size.

FIG. 10 is a graph illustrating emulation speed vs. pin count for a torus and a crossbar configuration.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

Although aspects of the invention are applicable to simulator systems, the invention is particularly advantageous in emulator systems where the emulator may be directly connected to peripheral circuitry. Pins for interchip communications can be limited by multiplexing interchip signals, yet input/output signals may be assigned dedicated pins for connection to the peripheral circuitry.

FIG. 1 is a block diagram of a typical prior art logic emulation system 5. The performance of the system 5 is achieved by partitioning a logic design, described by a netlist, across an interconnected array of FPGAs 10. This array is connected to a host workstation 2 which is capable of downloading design configurations, and is directly wired into the target system 8 for the logic design. Memory elements 6 may also be connected to the array of FPGAs 10. The netlist partition on each FPGA (hereinafter FPGA partition), configured directly into logic circuitry, can then be executed at hardware speeds.

In existing architectures, shown in FIG. 2, both the logic configuration and the network connectivity remain fixed for the duration of the emulation. FIGS. 2 shows an example of six logical wires 11*a*–*f*, 19*a*–*f* allocated to six physical interconnections 15*a*–*f*. Each emulated gate is mapped to one FPGA equivalent gate and each emulated signal is allocated to one FPGA pin. Thus, for a partition to be feasible, the partition gate and pin requirements must be no greater that the available FPGA resources. This constraint yields the following possible scenarios for each FPGA partition:

1. Gate limited: no unused gates, but some unused pins.
2. Pin limited: no unused pins, but some unused gates.
3. Not limited: unused FPGA pins and gates.
4. Balanced: no unused pins or gates.

For mapping typical circuits onto available FPGA devices, partitions are predominately pin limited; all available gates cannot be utilized due to a lack of pin resources to support them. Low utilization of gate resources increases both the number of FPGAs 10 needed for emulation and the time required to emulate a particular design. Pin limits set a hard upper bound on the maximum usable gate count any FPGA gate size can provide. This discrepancy will only get worse as technology scales; trends (and geometry) indicate that available gate counts are increasing faster than available pin counts.

In a preferred embodiment of the invention, shown in FIG. 3, virtual interconnections are used to overcome pin limitations in FPGA-based logic emulators. FIG. 3 shows an example of six logical wires 11a–f sharing a single physical wire 15x. The physical wire 15x is multiplexed 13 between two pipelined shift loops 20a, 20b, which are discussed in detail below. Pipelining refers to signal streams in a particular phase and multiplexing refers to signals across phases. A virtual interconnection represents a connection between a logical output 11a on one FPGA 10 and a logical input 19'a on another FPGA 10'. Established via a pipelined, statically routed communication network, these virtual interconnections increase available off-chip communication bandwidth by multiplexing 13 the use of FPGA pin resources (physical wires) 15 among multiple emulation signals (logical interconnections).

Virtual interconnections effectively relax pin limitations. Although low pin counts may decrease emulation speed, there is not a hard pin constraint that must be enforced. Emulation speed can be increased if there is a large enough reduction in system size. The gate overhead of using virtual interconnections is low, comprising gates that are not utilized in the purely hardwired implementation. Furthermore, the flexibility of virtual interconnections allows the emulation architecture to be balanced for each logic design application.

The logic emulator or the reconfigurable logic system may emulate a logic design that has a clock. The corresponding clock in the emulation or reconfigurable logic system is an emulation clock. One-to-one allocation of emulation signals (logical wires) 11, 19 to FPGA pins (physical wires) 15 does not exploit available off-chip bandwidth because emulation clock frequencies are one or two orders of magnitude lower than the potential clocking frequency of the FPGA technology, and all logical interconnections 11, 19 are not active simultaneously.

By pipelining and multiplexing physical wires 15, virtual interconnections are created to increase usable bandwidth. By clocking physical wires 15 at the maximum frequency of the FPGA technology, several logical connections can share the same physical resource.

In a logic design, evaluation flows from system inputs to system outputs. In a synchronous design with no combinatorial loops, this flow can be represented as a directed acyclic graph. Thus, through intelligent dependency analysis of the underlying logic circuit, logical values between FPGA partitions need to only be transmitted once. Furthermore, because circuit communication is inherently static, communication patterns repeat in a predictable fashion.

In a preferred embodiment of the invention, virtual interconnections are supported with a "softwire" compiler. This compiler analyzes logic signal dependencies and statically schedules and routes FPGA communication. These results are then used to construct (in the FPGA technology) a statically routed network. This hardware consists of a sequencer and shift loops. The sequencer is a distributed finite state machine. The sequencer establishes virtual connections between FPGAs by strobing logical interconnections in a predetermined order into special shift registers 21, the shift loops 20. The shift loops 20 serve as multiplexers 13 and are described in detail below. Shift loops 20 are then alternately connected to physical wires 15 according to the predetermined schedule established by the sequences.

The use of virtual interconnections is limited to synchronous logic. Any asynchronous signals must still be "hardwired" to dedicated FPGA pins. This limitation is imposed by the inability to statically determine dependencies in asynchronous loops. Furthermore, each combinational loop (such as a flip-flop) in a synchronous design is completely

contained in a single FPGA partition. For simplicity and clarity of description, it is assumed that the emulated logic has a single global clock.

In a preferred embodiment of the invention, virtual interconnections are implemented in the context of a complete emulation software system, independent of target FPGA device and interconnect topology. While this embodiment focuses primarily on software, the ultimate goal of the invention is a low-cost, reconfigurable emulation system.

In a preferred embodiment, the signals are routed through each FPGA by assigning a plurality of pins and time slots through intermediate FPGAs. This embodiment avoids the use of a crossbar. By routing the signals through each FPGA, speed is increased because there are no long wires connecting the FPGAs to a crossbar.

In contrast to prior systems, a preferred embodiment of the invention does not dedicate a signal path from source to destination. In particular, a preferred embodiment of the invention employs static routed packet switching where the wires over which a first signal propagates can be reused by a second signal before the first signal reaches its destination. Thus only a single link in the signal path is dedicated during any system clock period. Indeed, the FPGAs can buffer signals such that higher priority signals can propagate over a wire before a competing lower priority signal.

FIG. 4 graphically represents an emulation phase clocking scheme. The emulation clock period 52x is the clock period of the logic design being emulated. This is broken into evaluation phases (54a, 54b, 54c) to accommodate multiplexing. Multiple phases are required because the combinational logic between flip-flops in the emulated design may be split across multiple FPGA partitions and multiplexing of virtual interconnections prevents direct pass of all signals through the partitions. The phases permit a single pin to send different logical signals on every phase. Within a phase 54, evaluation is accomplished within each partition, and the results are then communicated to other FPGA partitions. Although three phases are illustrated per emulation period, it will be understood that more or less phases can be employed.

At the beginning of the phase 54, logical outputs of each FPGA partition are determined by the logical inputs in input shift loops. At the end of the phase 54, outputs are then sent to other FPGA partitions with pipelined shift loops and intermediate hop stages. As illustrated in FIG. 4, these pipelines are clocked with a pipeline clock 56 at the maximum frequency of the FPGA. After all phases 54 within an emulation clock period 52x are complete, the emulation clock 52 is ticked to clock all flip-flops of the target circuit.

The input to the softwire compiler consists of a netlist 105 of the logic design to be emulated, target FPGA device characteristics, and interconnect topology. The compiler then produces a configuration bitstream that can be downloaded into the emulator. FIG. 5 is a flowchart of the compilation steps. Briefly, these steps include translation and mapping of the netlist to the target FPGA technology (step 110), partitioning the netlist (step 120), placing the partitions into interconnect topology (steps 130, 140), routing the inter-node communication paths (steps 150, 160), and finally FPGA-specific automated placement and routing (APR) (step 170).

The input netlist 105 to be emulated is usually generated with a hardware description language or schematic capture program. This netlist 105 must be translated and mapped (step 110) to a library of FPGA macros. It is important to perform this operation before partitioning so that partition gate counts accurately reflect the characteristics of the target

FPGAs. Logic optimization tools can also be used at this point to optimize the netlist for the target architecture (considering the system as one large FPGA).

After mapping (step 110) the netlist to the target architecture, the netlist must be partitioned (step 120) into logic blocks that can fit into the target FPGA. With only hardwires, each partition must have both fewer gates and fewer pins than the target device. With virtual interconnections, the total gate count (logic gates and virtual interconnections overhead) must be no greater than the target FPGA gate count. A preferred embodiment uses the Concept Silicon partitioner manufactured by InCA, Inc. This partitioner performs K-way partitioning with min-cut and clustering techniques to minimize partition pin counts.

Because a combinatorial signal may pass through several FPGA partitions during an emulated clock cycle, all signals will not be ready to schedule at the same time. A preferred embodiment solves this problem by only scheduling a partition output once all the inputs it depends upon are scheduled (step 130). An output depends on an input if a change in that input can change the output. To determine input to output dependencies, the logic netlist is analyzed, backtracing from partition outputs to determine which partition inputs they depend upon. In backtracing, it is assumed that all outputs depend on all inputs for gate library parts, and no outputs depend on any inputs for latch (or register) library parts. If there are no combinatorial loops that cross partition boundaries, this analysis produces a directed acyclic graph, the signal flow graph (SFG), to be used by the global router.

Following logic partitioning, individual FPGA partitions must be placed into specific FPGAs (step 140). An ideal placement minimizes system communication, thus requiring fewer virtual interconnection cycles to transfer information. A preferred embodiment first makes a random placement followed by cost-reduction swaps, and then further optimize with simulated annealing.

During global routing (150), each logical wire is scheduled to a phase, and assigned a pipeline time slot (corresponding to one cycle of the pipeline clock in that phase on a physical wire). Before scheduling, the criticality of each logical wire is determined (based on the signal flow graph produced by dependency analysis). In each phase, the router first determines the schedulable wires A wire is schedulable if all wires it depends upon have been scheduled in previous phases. The router than uses shortest path analysis with a cost function based on pin utilization to route as many schedulable signals as possible, routing the most critical signals first. Any schedulable signals which cannot be routed are delayed to the next phase.

Once routing is completed, appropriately-sized shift loops and associated logic are added to each partition to complete the internal FPGA hardware description (step 160). At this point, there is one netlist for each FPGA. These netlists are then be processed with the vendor-specific FPGA place and route software (step 170) to produce configuration bitstreams (step 195).

Technically, there is no required hardware support for implementation of virtual interconnections (unless one considers re-designing an FPGA optimized for virtual interconnecting). The necessary "hardware" is compiled directly into configuration for the FPGA device. Thus, any existing FPGA-based logic emulation system can take advantage of virtual interconnecting. Virtual interconnections can be used to store and retrieve data from memory elements external to the FPGAs by multiplexing the data on the virtual interconnections during a phase. There are many

possible ways to implement the hardware support for virtual interconnections. A preferred embodiment employs a simple and efficient implementation. The additional logic to support virtual interconnections can be composed entirely of shift loops and a small amount of phase control logic.

FIG. 6 is a block diagram of a preferred shift loop architecture. A shift loop 20 is a circular, loadable shift register with enabled shift in and shift out ports. Each shift register 21 is capable of performing one or more of the operations of load, store, shift, drive, or rotate. The Load operation strobes logical outputs into the shift loop. The Store operation drives logical inputs from the shift loop. The Shift operation shifts data from a physical input into the shift loop. The Drive operation drives a physical output with the last bit of the shift loop. The Rotate operation rotates bits in the shift loop. In a preferred embodiment, all outputs loaded into a shift loop 20 must have the same final destination FPGA. As described above, a logical output can be strobed once all corresponding depend inputs have been stored. The purpose of rotation is to preserve inputs which have reached their final destination and to eliminate the need for empty gaps in the pipeline when shift loop lengths do not exactly match phase cycle counts. In this way, a signal may be rotated from the shift loop output back to the shift loop input to wait for an appropriate phase. Note that in this implementation the store operation cannot be disabled.

Shift loops 20 can be re-scheduled to perform multiple output operations. However, because the internal latches being emulated depend on the logical inputs, inputs need to be stored until the tick of the emulation clock.

For networks where multiple hops are required (i.e. a mesh), one-bit shift registers 21 that always shift and sometimes drive are used for intermediate stages. FIG. 7 is a block diagram of the intermediate hop pipeline stage. These stages are chained together, one per FPGA hop, to build a pipeline connecting the output shift loop on the source FPGA 10 with the input shift loop on the destination FPGA 10'.

The phase control logic is the basic run-time kernel in a preferred embodiment. This kernel is a sequencer that controls the phase enable and strobe (or load) lines, the pipeline clock, and the emulation clock. The phase enable lines are used to enable shift loop to FPGA pin connections. The phase strobe lines strobe the shift loops on the correct phases. This logic is generated with a state machine specifically optimized for a given phase specification.

EXPERIMENTAL RESULTS

The system compiler described above was implemented by developing a dependency analyzer, global placer, global router, and using the InCA partitioner. Except for the partitioner, which can take hours to optimize a complex design, running times on a SPARC 2 workstation were usually 1 to 15 minutes for each stage.

To evaluate the costs and benefits of virtual interconnections, two complex designs were compiled, SPARCLE and the A-1000. SPARCLE is an 18K gate SPARC microprocessor enhanced with multiprocessing features. The Alewife compiler and memory management unit (A-1000) is an 86K gate cache compiler for the Alewife Multiprocessor, a distributed shared memory machine being designed at the Massachusetts Institute of Technology. For target FPGAs, the Xilinx 3000 and 4000 series (including the new 4000H series) and the Concurrent Logic Cli6000 series were considered. This analysis does not include the final FPGA-specific APR stage; a 50 percent APR mapping efficiency for both architectures is assumed.

In the following analysis, the FPGA gate costs of virtual interconnections based on the Concurrent Logic CLI6000 series FPGA were estimated. The phase control logic was assumed to be 300 gates (after mapping). Virtual interconnections overhead can be measured in terms of shift loops. In the Cli6000, a bit stage shift register takes 1 of 3136 cells in the 5K gate part (C_s=3 mapped gates). Thus, total required shift register bits for a partition is then equal to the number of inputs. When routing in a mesh or torus, intermediate hops cost 1 bit per hop. The gate overhead is then $C_s \times S$, where C_s is the cost of a shift register bit, and S is the number of bits. S is determined by the number of logical inputs, V_i, and M_p, the number of times a physical wire p is multiplexed (this takes into account the shift loop tristate driver and the intermediate hop bits). Gate overhead is then approximately:

$$Gate_{vw} = C_s \times (V_i + \Sigma_p M_p),$$

Storage of logical outputs is not counted because logical outputs can be overlapped with logical inputs.

Before compiling the two test designs, their communication requirements were compared to the available FPGA technologies. For this comparison, each design was partitioned for various gate counts and the pin requirements were measured. FIG. 8 shows the resulting curves, plotted on a log-log scale. Note that the partition gate count is scaled to represent mapping inefficiency.

Both design curves and the technology curves fit Rent's Rule, a rule of thumb used for estimating communication requirement in random logic. Rent's Rule can be stated as:

$$pins_2/pins_1 = (gates_2/gate_1)^b,$$

where $pins_2$, $gates_2$ refer to a partition, and $pins_1$, $gates_1$ refer to a sub-partition, and b is constant between 0.4 and 0.7. Table 1 shows the resulting constants. For the technology curve, a constant of 0.5 roughly corresponds to the area versus perimeter for the FPGA die. The lower the constant, the more locality there is within the circuit. Thus, the A-1000 has more locality than SPARCLE, although it has more total communication requirements. As FIG. 8 illustrates, both SPARCLE and the A-1000 will be pin-limited for any choice of FPGA size. In hardwired designs with pin-limited partition sizes, usable gate count is determined solely by available pin resources. For example, a 5000 gate FPGA with 100 pins can only utilize 1000 SPARCLE gates or 250 A-1000 gates.

TABLE 1

Rent's Rule Parameter (slope of log-log curve)		
FPGA Technology	SPARCLE	A-1000
0.50	0.06	0.44

Table 1: Rent's Rule Parameter (slope of log-log curve)

Next, both designs were compiled for a two dimensional torus and a full crossbar interconnect of 5000 gate, 100 pin FPGAs, 50 percent mapping efficiency. Table 2 shows the results for both hard wires and virtual interconnections. Compiling the A-1000 to a torus, hardwires only, was not practical with the partitioning software. The gate utilizations obtained for the hardwired cases agree with

TABLE 2

Number of 5K Gates, 100 Pin FPG As Required for Logic Emulation

	Hardwires Only		Virtual Interconnections Only	
Design	2-D Torus	Full Crossbar	2-D Torus	Full Crossbar
Sparcle (18K gates)	>100 (<7%)	31 (23%)	9 (80%)	9 (80%)
A-1000 (86K gates)	Not Practical	>400 (<10%)	49 (71%)	42 (83%)

Number of FPGAs (Average Usable Gate Utilization)

Table 2: Number of 5K Gates, 100 Pin FPG

As Required for Logic Emulation

reports in the literature on designs of similar complexity. To understand the tradeoffs involved, the hardwires pin/gate constraint and the virtual interconnections pin/gate tradeoff curve were plotted against the partition curves for the two designs (FIG. 9). The intersection of the partition curves and the wire curves gives the optimal partition and sizes. This graph shows how virtual interconnections add the flexibility of trading gate resources for pin resources.

Emulation clock cycle time T_E is determined by:

1. Communication delay per hop, t_c;
2. Length of longest path in dependency graph L;
3. Total FPGA gate delay along longest path T_L;
4. Sum of pipeline cycles across all phases, n;
5. Network diameter, D (D=1 for crossbar); and
6. Average network distance, k_d (k_d=1 for crossbar).

The total number of phases and pipeline cycles in each phase are directly related to physical wire contention and the combinatorial path that passes through the largest number of partitions. If the emulation is latency dominated, then the optimal number of phases is L, and the pipeline cycles per phase should be no greater than D, giving:

$$n = L \times D.$$

If the emulation is bandwidth dominated, then the total pipeline cycles (summed over all phases) is at least:

$$n = MAX_p[(Vi_p/Pi_p)]$$

where Vi_p and Pi_p are the number of virtual and physical wires for FPGA partition p. If there are hot spots in the network (not possible with a crossbar), the bandwidth dominated delay will be higher. Emulation speeds for SPARCLE and the A-1000 were both latency dominated.

Based on CLi6000 specifications, it was assumed that T_L=250 ns and t_c=20 ns (based on a 50 MHZ clock). A computation-only delay component, and a communication-only delay component were considered. This dichotomy is used to give a lower and upper bound on emulation speed.

The computation-only delay component is given by:

$$T_p = T_L + t_c \times n,$$

where n=0 for the hardwired case.

The communication-only delay component is given by:

$$T_c = t_c \times n.$$

Table 3 shows the resulting emulation speeds for virtual and hardwires for the crossbar topology. The emulation clock range given is based on the sum and minimum of the two components (lower and upper bounds). When the use of virtual interconnections allows a design to be partitioned across fewer FPGAs, L is decreased, decreasing T_c. However, the pipeline stages will increase T_p by t_c per pipeline cycle.

TABLE 3

Emulation Clock Speed Comparison

		Hardwire Only	Virtual Interconnection Only
SPARCLE	Longest Path	9 hops	6 hops
	Computation Only Delay	250 ns	370 ns
	Communication Only Delay	180 ns	120 ns
	Emulation Clock Range	2.3–5.6 MHz	2.0–8.3 MHz
A-1000	Longest Path	27 hops	17 hops
	Computation Only Delay	250 ns	590 ns
	Communication Only Delay	540 ns	340 ns
	Emulation Clock Range	1.3–4.0 MHz	1.1–2.9 MHz

Table 3: Emulation Clock Speed Comparison

In Table 3, the virtual interconnections emulation clock was determined solely by the length of the longest path; the communication was limited by latency, not bandwidth. To determine what happens when the design becomes bandwidth limited, the pin count was varied and the resulting emulation clock (based on T_c) was record ed f or both a crossbar and torus topology. FIG. 10 shows the results for the A-1000. The knee of the curve is where the latency switches from bandwidth dominated to latency dominated. The torus is slower because it has a larger diameter, D. However, the torus moves out of the latency dominated region sooner because it exploits locality; several short wires can be routed during the time of a single long wire. Note that this analysis assumes the crossbar can be clocked as fast as the torus; the increase in emulation speed obtained with the crossbar is lower if t_c is adjusted accordingly.

With virtual interconnections, neither designs was bandwidth limited, but rather limited by its respective critical paths. As shown in FIG. 10, the A-1000 needs only about 20 pins per FPGA to run at the maximum emulation frequency. While this allows the use of lower pin count (and thus cheaper) FPGAs, another option is to trade this surplus bandwidth for speed. This tradeoff is accomplished by hardwiring logical interconnections at both ends of the critical paths. Critical wires can be hardwired until there is no more surplus bandwidth, thus fully utilizing both gate and pin resources. For designs on the 100 pin FPGAs, hardwiring reduces the longest critical path from 6 to 3 for SPARCLE and from 17 to 15 for the A-1000.

Virtual interconnections allow maximum utilization of FPGA gate resources at emulation speeds competitive with existing hardwired techniques. This technique is independent of topology. Virtual interconnections allow the use of less complex topologies, such as a torus instead of a crossbar, in cases where such a topology was not practical otherwise.

Using timing and/or locality sensitive partitioning with virtual interconnections has potential for reducing the required number of routing sub-cycles. Communication bandwidth can be further increased with pipeline compaction, a technique for overlapping the start and end of

long virtual paths with shorter paths traveling in the same direction. A more robust implementation of virtual interconnections replaces the global barrier imposed by routing phases with a finer granularity of communication scheduling, possible overlapping computation and communication as well.

Using the information gained from dependency analysis, one can now predict which portions of the design are active during which parts of the emulation clock cycle. If the FPGA device supports fast partial reconfiguration, this information can be used to implement virtual logic via invocation of hardware subroutines. An even more ambitious direction is event-driven emulation—only send signals which change, only activate (configure) logic when it is needed.

Equivalents

Those skilled in the art will know, or be able to ascertain using no more than routine experimentation, many equivalents to the specific embodiments of the invention described herein.

These and all other equivalents are intended to be encompassed by the following claims.

The invention claimed is:

1. A reconfigurable electronic system comprising:
   a plurality of reprogrammable logic modules, each logic module having a plurality of pins for communicating signals external to the logic module and a plurality of logic elements for implementing logic in hardware;
   inter-module connections between pins of different logic modules; and
   a configurer for automatically configuring each logic module to define a partition of a specified target circuit with communications between the partitions of the target circuit being provided through pins and inter-module connections,
   a partition of the configured system having a number of inter-module communications to other partitions that exceeds the number of pins on the logic module of the partition and the logic module of the partition being configured to communicate through virtual interconnections in a time-multiplexed fashion through at least one pin of the logic module of the partition, the configurer determining a static virtual interconnection which includes a communication path extending through an intermediate reprogrammable logic module.

2. A system as claimed in claim 1 wherein each logic module comprises an array of interconnected programmable logic cells.

3. A system as claimed in claim 1 wherein the configurer configures a logic module to form a multiplexer for communicating through virtual interconnections.

4. A system as claimed in claim 1 wherein the logic modules are configured to operate in phases within a target clock period with the inter-module communications being performed within each phase.

5. A system as claimed in claim 4 wherein the configurer optimizes logic module selection and phase division of the target circuit based on inter-module dependencies.

6. A system as claimed in claim 4 wherein the target clock period is a clock period of the target circuit which dictates the maximum rate at which signal lines of the target circuit change value and wherein each target clock period comprises a plurality of system clock periods which dictate the maximum rate at which signals in the electronic system change value.

7. A system as claimed in claim 1, including asynchronous logic hardwired to dedicated pins of the logic modules.

8. A system as claimed in claim 1 wherein data is accessed from memory elements external to the logic modules.

9. A system as claimed in claim 8 wherein there are time multiplexed interconnections between the logic modules and the memory elements.

10. A system as claimed in claim 1 wherein the logic modules are Field Programmable Gate Arrays (FPGAs).

11. A system as claimed in claim 1 wherein each logic module is a single chip.

12. A system as claimed in claim 1 wherein the digital system is an emulation system for emulating the target system.

13. A system as claimed in claim 1 wherein logic modules are configured to include pins dedicated to individual signals.

14. A logic system as claimed in claim 1 wherein the configurer comprises a partitioner for partitioning the target logic circuit, each partition being configured into a respective logic module.

15. A system as claimed in claim 14 further comprising a dependency analyzer and a divider for dividing a target clock period into phases during which program logic functions are performed and signals are transmitted between the logic modules, the phase division being a function of partition dependencies and memory assignments.

16. A system as claimed in claim 14 further comprising a router for configuring the logic modules to route signals between logic modules through intermediate reprogrammable logic modules.

17. A reconfigurable electronic system comprising:

a plurality of reprogrammable logic modules, each logic module having an array of gates reconfigurable to define a hardware logic circuit and a plurality of pins for communicating signals external to the logic module;

inter-module connections between pins of different logic modules; and

a configurer for automatically configuring the array of gates, each logic module to define a partition of a specified target circuit with communications between the partitions of the target circuit being provided through pins and inter-module connections

a partition of the configured system having a number of inter-module communications to other partitions that exceeds the number of pins on the logic module of the partition and gates of the logic module of the partition being configured as a multiplexer for receiving a plurality of outputs from the configured array of gates and for statically communicating the received outputs through a single pin in a multiplexed, pipelined fashion.

18. A system as claimed in claim 17 further comprising at least one shift register coupled between the multiplexer and the configured gate array.

19. A system as claimed in claim 18 wherein the shift registers are configured from gates in the logic module.

20. A reconfigurable electronic system comprising:

a plurality of reprogrammable logic modules, each logic module having a plurality of pins for communicating signals external to the logic module and a plurality of logic elements for implementing logic in hardware;

inter-module connections between pins of different logic modules; and

a configurer for automatically configuring each logic module to define a partition of a specified target circuit with communications between the partitions of the target circuit being provided through pins and inter-module connections,

a partition of the configured system having a number of inter-module communications to other partitions that exceeds the number of pins on the logic module of the partition and the logic module of the partition being configured to communicate through static virtual interconnections in a time-multiplexed fashion through at least one pin of the logic module of the partition, the electronic system including dedicated pins for providing a predetermined signal.

21. A method of automatically compiling a reconfigurable digital system, comprising the steps of:

partitioning a target circuit into a plurality of partitions, each partition to be configured into a reprogrammable logic module having a plurality of pins and a plurality of logic elements;

configuring the logic modules to create partitions of the target circuit;

determining static virtual interconnections between partitions corresponding to at least one common pin with signals along the virtual interconnections being time-multiplexed through the at least one common pin; and

configuring the logic modules to route signals between logic modules through intermediate logic modules.

22. A method as claimed in claim 21 further comprising the step of dividing a first clock period which dictates the maximum rate at which signal lines within the target circuit change value into phases during which program logic functions are performed and signals are transmitted between logic modules.

23. A reconfigurable logic module comprising:

an array of gates configurable to define a logic circuit; and

a virtual interconnection comprising a plurality of gates reconfigured as a multiplexer, the multiplexer receiving a plurality of outputs from the configured gate array and communicating each of the received outputs through a single pin in a multiplexed, pipelined fashion.

24. A logic module as claimed in claim 23 further comprising at least one register coupled between the multiplexer and the configured gate array.

25. A logic module as claimed in claim 24 wherein the at least one register is configured from gates in the logic module.

26. A logic module as claimed in claim 24 wherein the at least one register includes a shift register.

27. A logic module as claimed in claim 23 wherein the logic module is a Field Programmable Gate Array (FPGA).

28. A reconfigurable electronic system comprising

a plurality of reprogrammable logic modules, each logic module having a plurality of pins for communicating signals between logic modules; and

a configurer to configure each logic module to define a partition of a specified target circuit, a partition of the configured target circuit having a number of interconnections to other partitions that exceeds the number of pins on the logic module and the logic module being configured to communicate through virtual interconnections through at least one pin.

29. A system as claimed in claim 28 wherein the configurer configures a logic module to form a multiplexer for communicating through virtual interconnections.

30. A system as claimed in claim 28 wherein pins of logic modules are directly connected to pins of other logic modules and routing of signals between the logic modules is through intermediate logic modules.

31. A system as claimed in claim 28 wherein the logic modules comprise hardwired multiplexers.

32. A system as claimed in claim 28 wherein the configurer optimizes logic module selection and phase division of the target circuit based on interpartition dependencies.

33. A system as claimed in claim 28 wherein data is accessed from memory elements external to the logic modules.

34. A system as claimed in claim 28 wherein the logic modules are Field Programmable Gate Arrays (FPGAs).

35. A system as claimed in claim 28 wherein the system is an emulation system for emulating the target circuit.

36. A system as claimed in claim 28 further comprising inter-module connections between pins of different logic modules with interconnections between the partitions of the

target circuit being provided through pins and inter-module connections.

37. A system as claimed in claim 36 wherein the inter-module communications include interconnections which extend through intermediate reconfigurable logic modules.

38. A system as claimed in claim 28 wherein the logic modules are configured to operate in phases within a target clock period with inter-module communications being performed within each phase.

39. A system as claimed in claim 28 wherein the logic module is configured to communicate through virtual interconnections in a time-multiplexed fashion.

* * * * *