EXHIBIT A

US006009531A

United States Patent [19]

Selvidge et al.

[11] Patent Number: 6,009,531

[45] Date of Patent: *Dec. 28, 1999

[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: **Ikos Systems, Inc.**, Cupertino, Calif.

[*] Notice: This patent is subject to a terminal disclaimer.

[21] Appl. No.: **08/863,963**

[22] Filed: **May 27, 1997**

Related U.S. Application Data

[63] Continuation of application No. 08/513,605, Aug. 10, 1995, Pat. No. 5,649,176.

[51] **Int. Cl.**[6] **G06F 1/12**

[52] **U.S. Cl.** **713/400**

[58] **Field of Search** 713/400; 710/8, 710/10, 12; 712/10, 15, 36, 37

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,450,560	5/1984	Conner	371/25
4,697,241	9/1987	Lavi	364/488
5,513,338	4/1996	Alexander et al.	395/500
5,572,710	11/1996	Asano et al.	395/500

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).

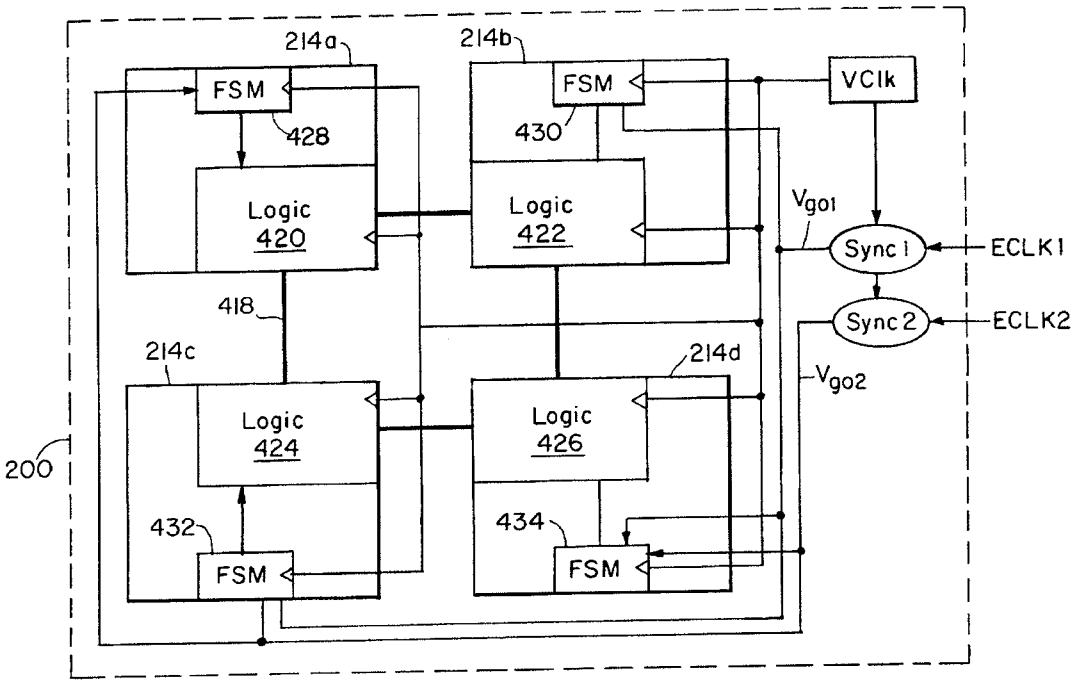
Primary Examiner—Thomas M. Heckler

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.

19 Claims, 14 Drawing Sheets



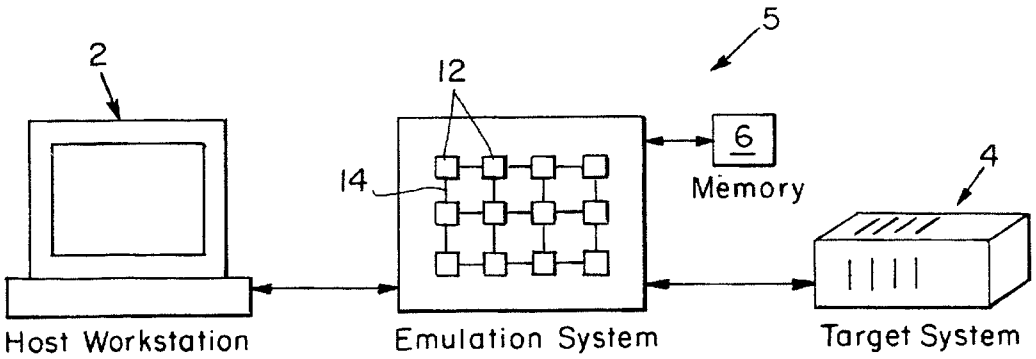


FIG. 1
(Prior Art)

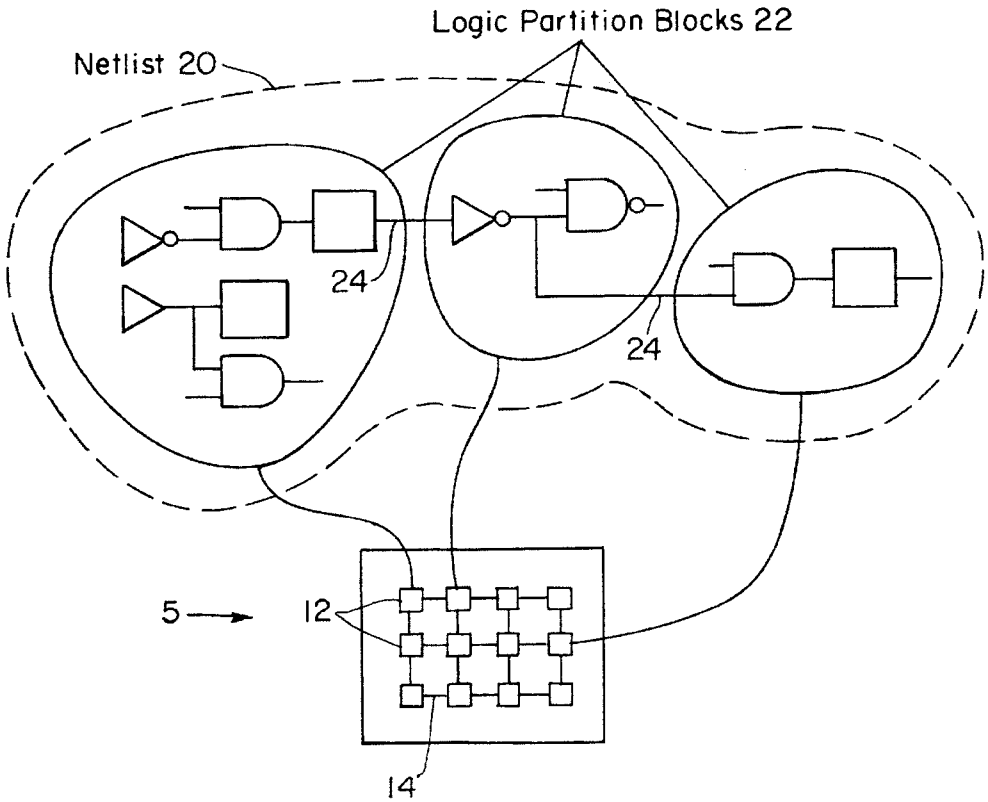


FIG. 2

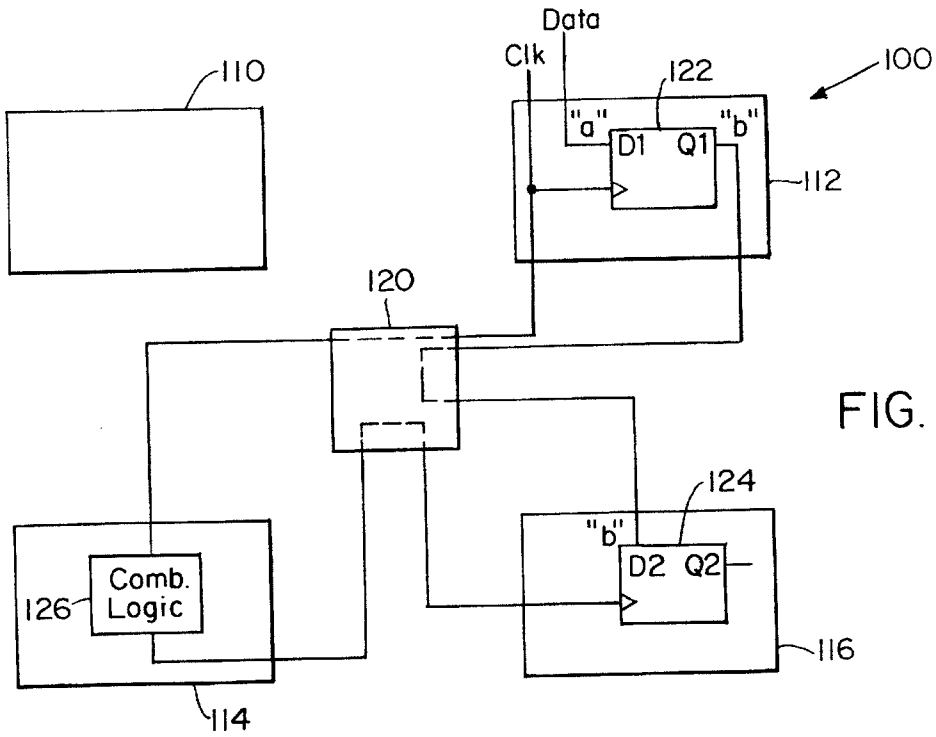

Data
Clk
110
122
100
"a"
D1
Q1
"b"
112
120
124
"b"
D2
Q2
Comb.
Logic
126
116
114

FIG. 3

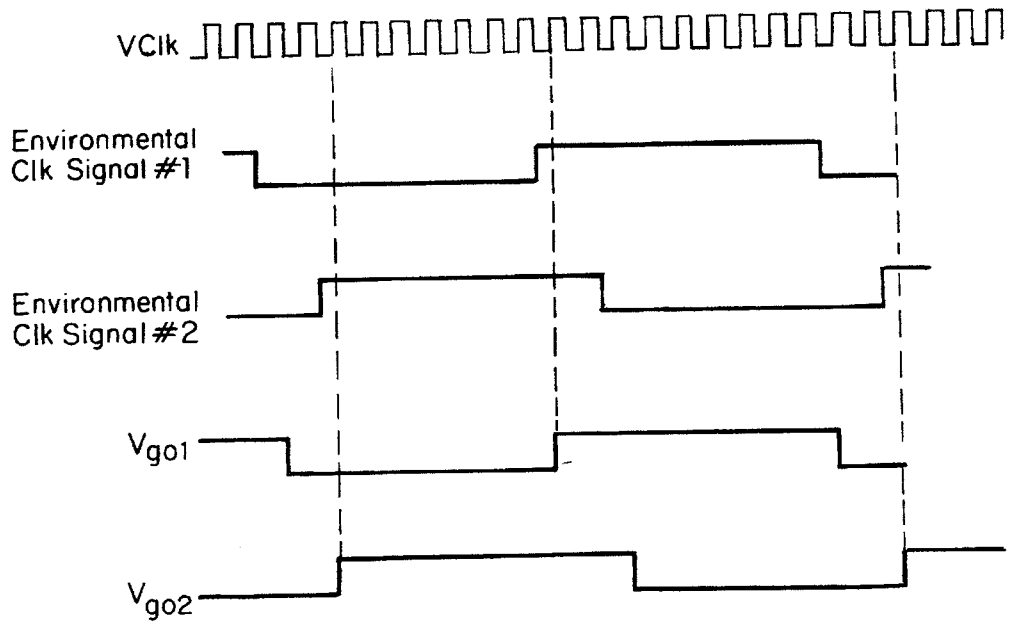

VClk
Environmental
Clk Signal #1
Environmental
Clk Signal #2
Vgo1
Vgo2

FIG. 4B

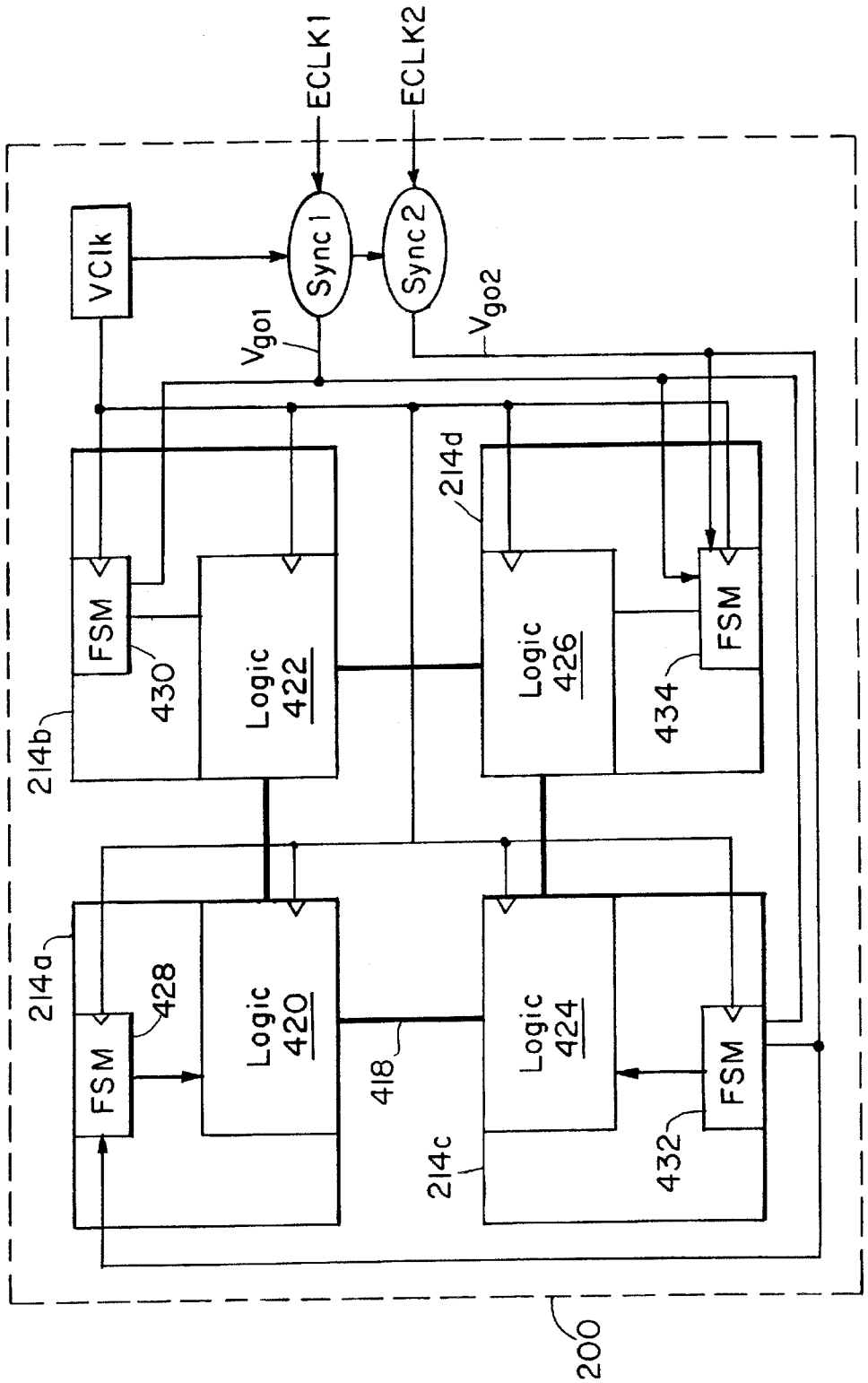

ECLK1
ECLK2
VClk
Sync 1
Sync 2
Vgo1
Vgo2
214d
FSM
430
Logic 422
214b
Logic 426
FSM
434
214a
428
FSM
Logic 420
418
Logic 424
FSM
214c
432
200

FIG. 4A

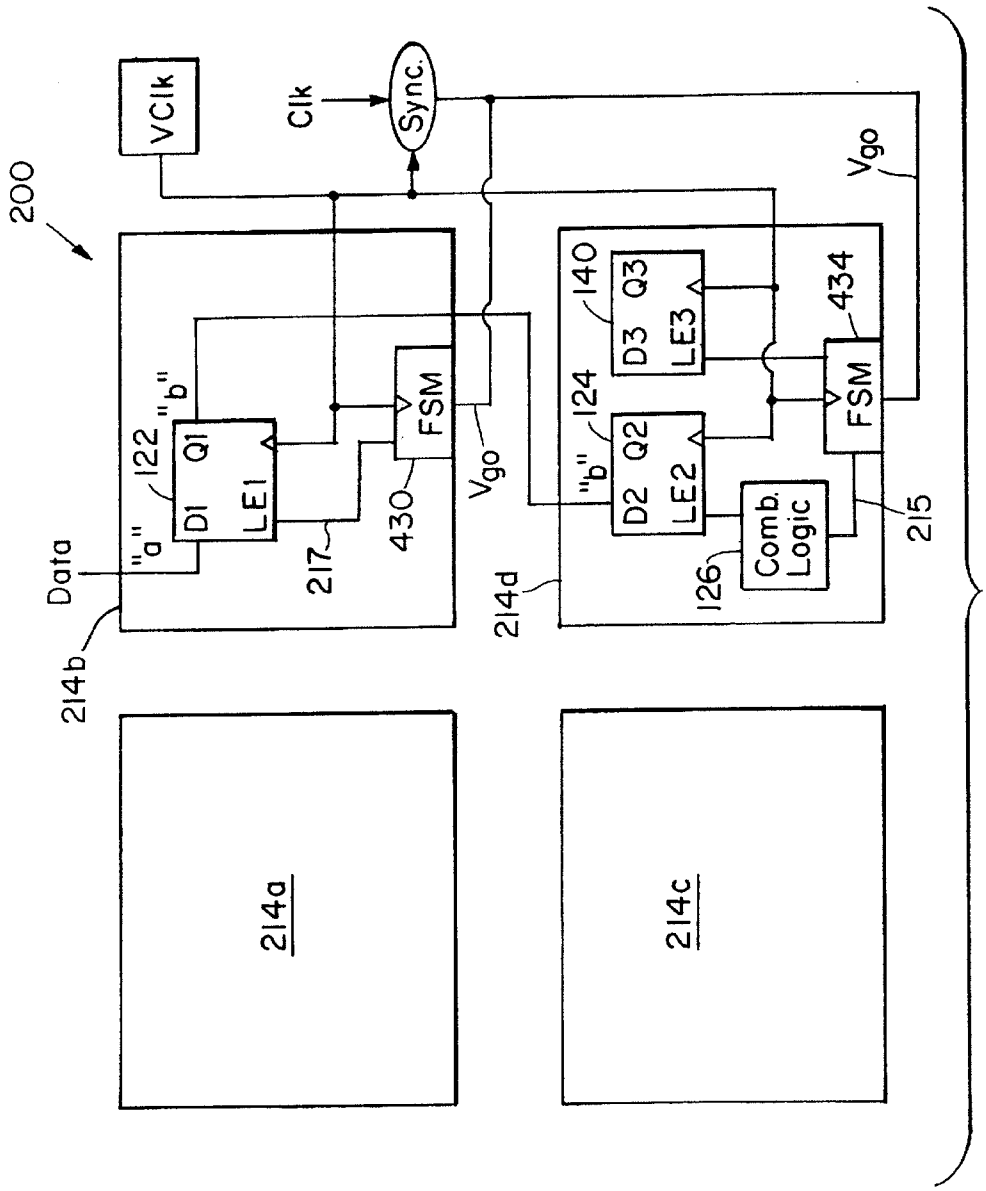

200
VClk
Clk
Sync.
Vgo
140
D3 Q3
LE3
434
FSM
124
"b"
D2 Q2
LE2
Comb. Logic
215
126
214d
"b"
122
"a"
D1 Q1
LE1
FSM
430
217
Vgo
Data
214b
214a
214c

FIG. 5A

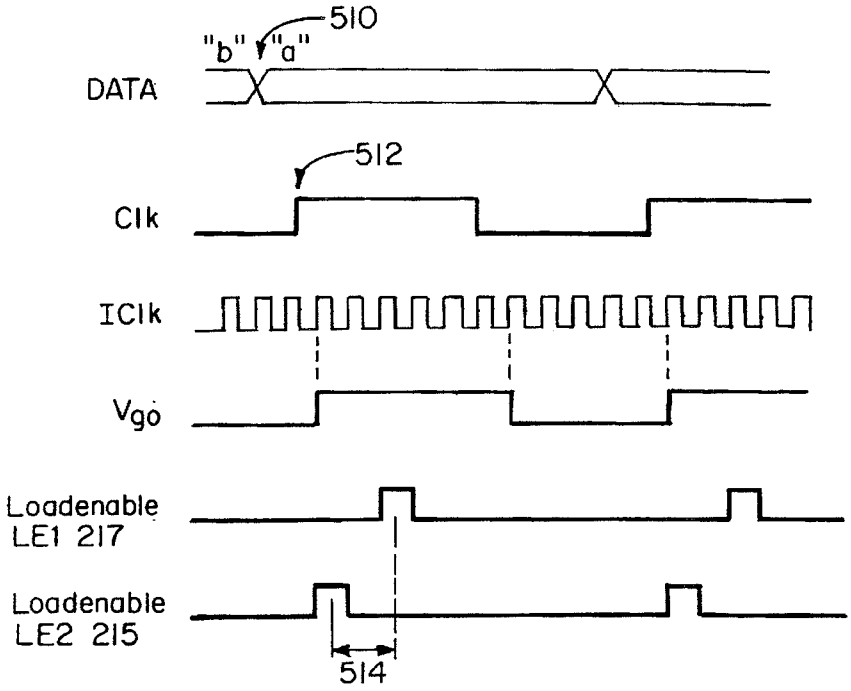

510
"b"
"a"
DATA
512
Clk
ICIk
Vgo
Loadenable
LE1 217
Loadenable
LE2 215
514

FIG. 5B

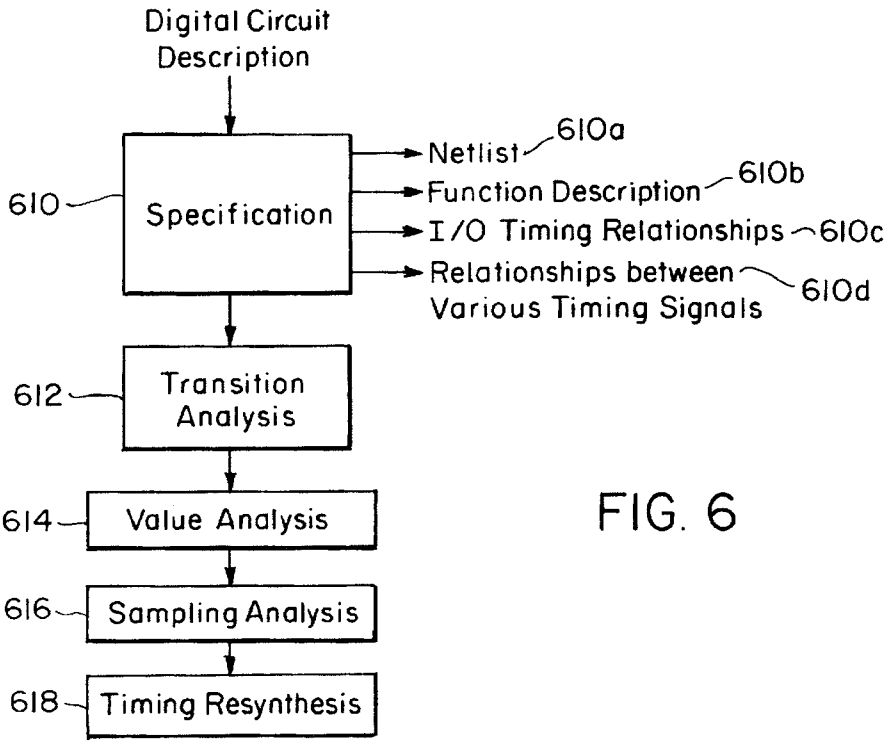

Digital Circuit
Description
610
Specification
Netlist
610a
Function Description
610b
I/O Timing Relationships
610c
Relationships between
Various Timing Signals
610d
612
Transition
Analysis
614
Value Analysis
616
Sampling Analysis
618
Timing Resynthesis

FIG. 6

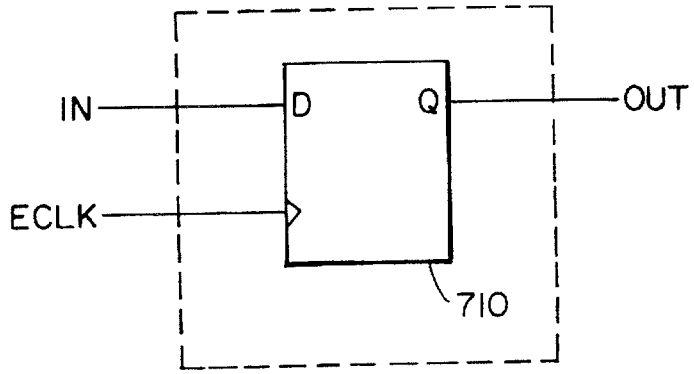

IN
ECLK
D
Q
OUT
710

FIG. 7A

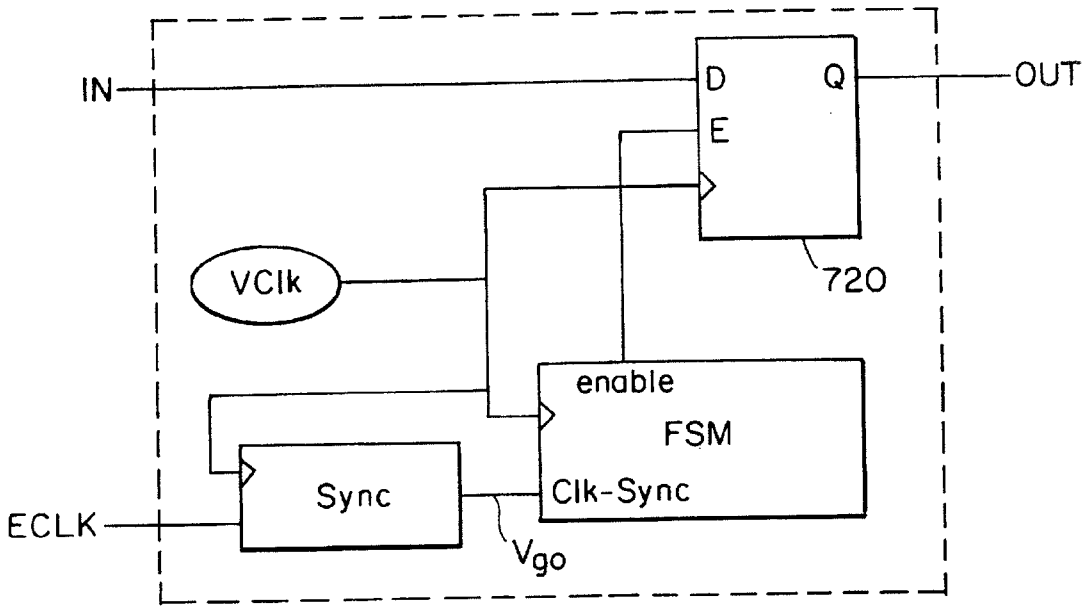

IN
D
E
Q
OUT
720
VClk
enable
FSM
Sync
Clk-Sync
ECLK
Vgo

FIG. 7B

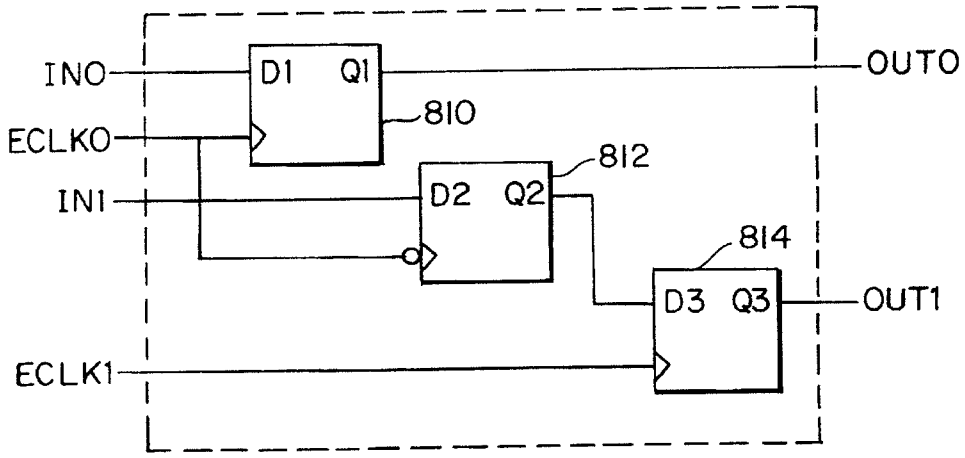

INO
ECLKO
IN1
ECLK1
D1 Q1
810
D2 Q2
812
D3 Q3
814
OUT0
OUT1

FIG. 8A

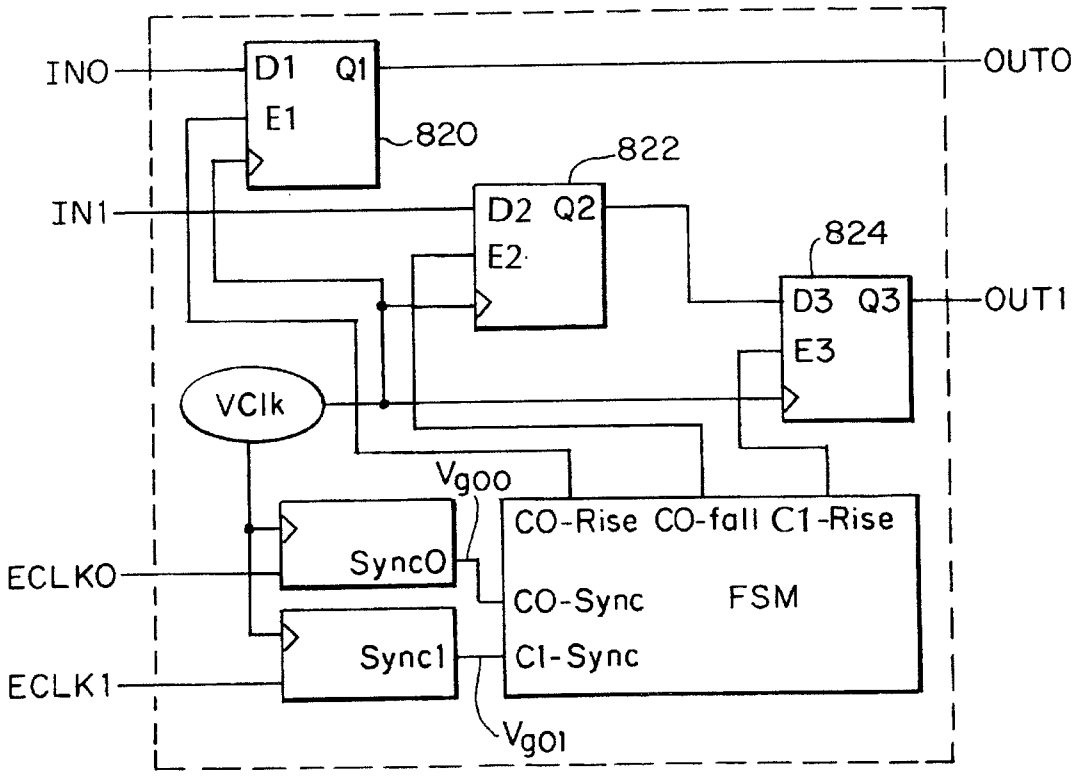

INO
IN1
ECLKO
ECLK1
D1 Q1
E1
820
D2 Q2
E2
822
D3 Q3
E3
824
OUT0
OUT1
VClk
Vgoo
Vgo1
Sync0
Sync1
CO-Rise CO-fall C1-Rise
CO-Sync
C1-Sync
FSM

FIG. 8B

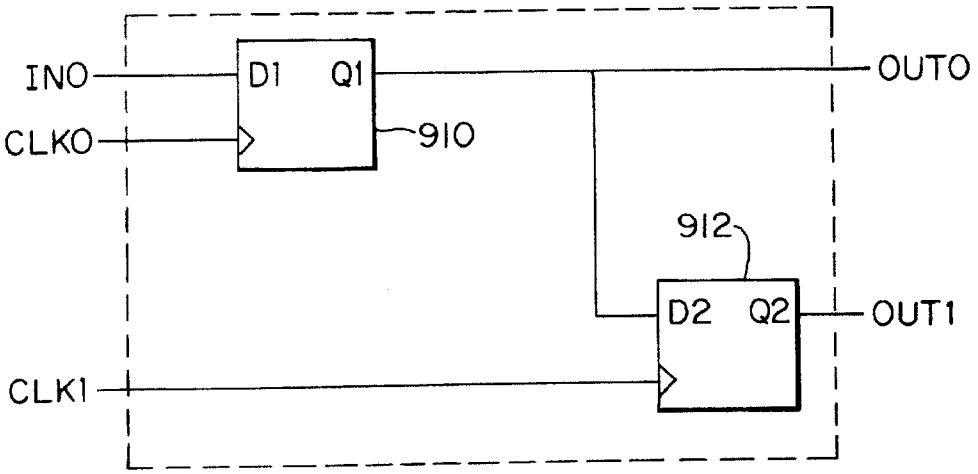

INO
CLKO
D1
Q1
910
OUTO
912
D2
Q2
OUT1
CLK1

FIG. 9A

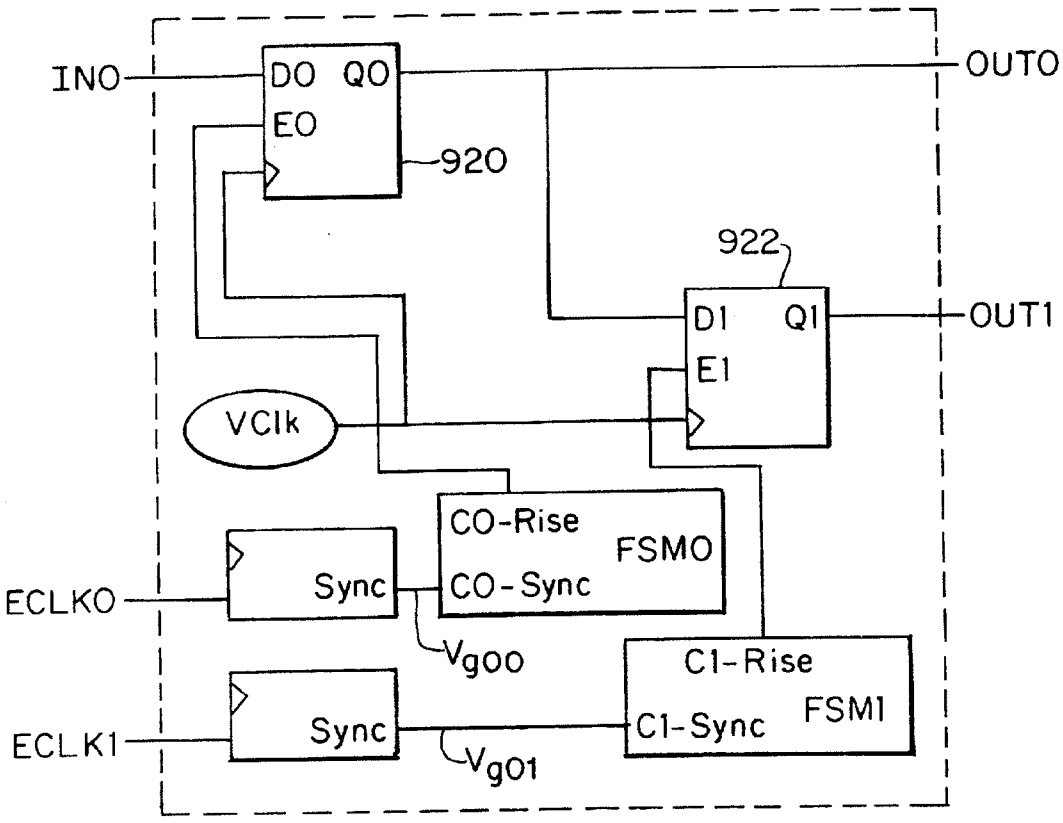

INO
DO
QO
EO
920
OUTO
922
D1
Q1
E1
OUT1
VClk
CO-Rise
FSMO
CO-Sync
Sync
ECLKO
Vgoo
C1-Rise
FSM1
C1-Sync
Sync
ECLK1
Vg01

FIG. 9B

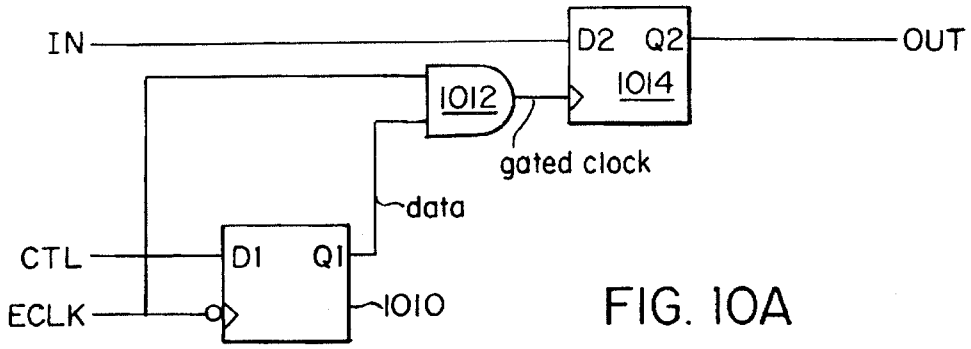


FIG. 10A

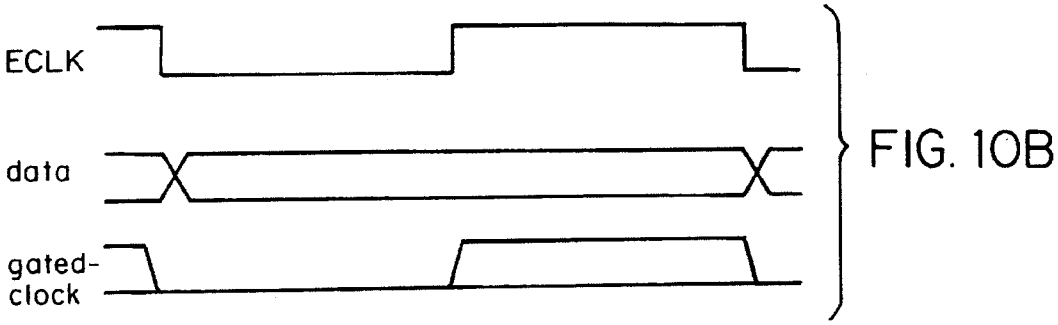


FIG. 10B

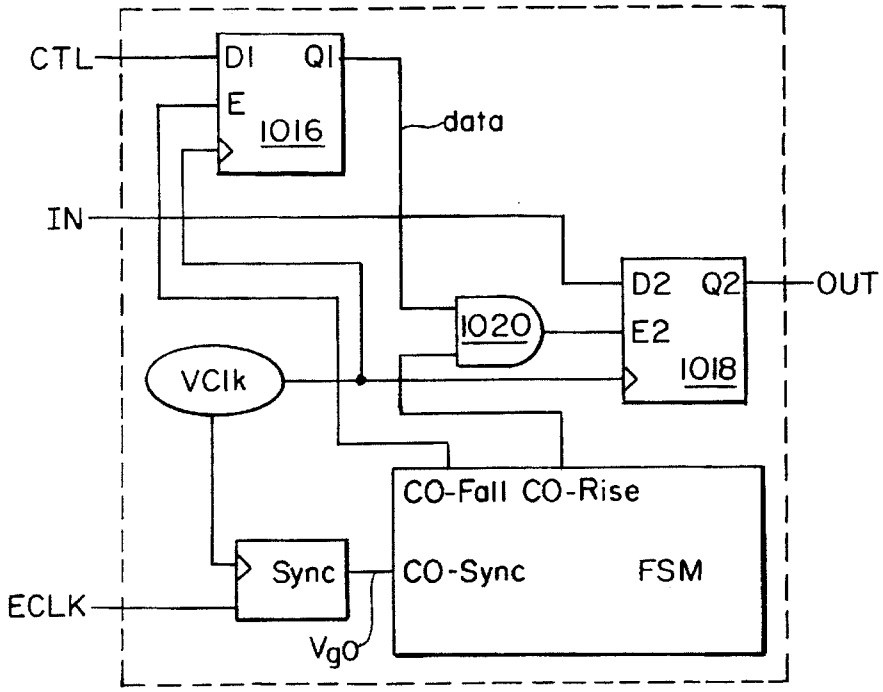


FIG. 10C

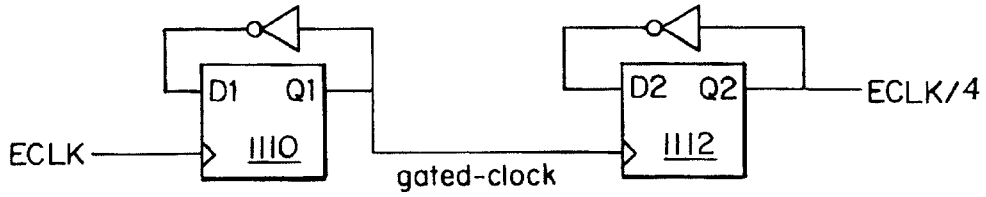


FIG. 11A

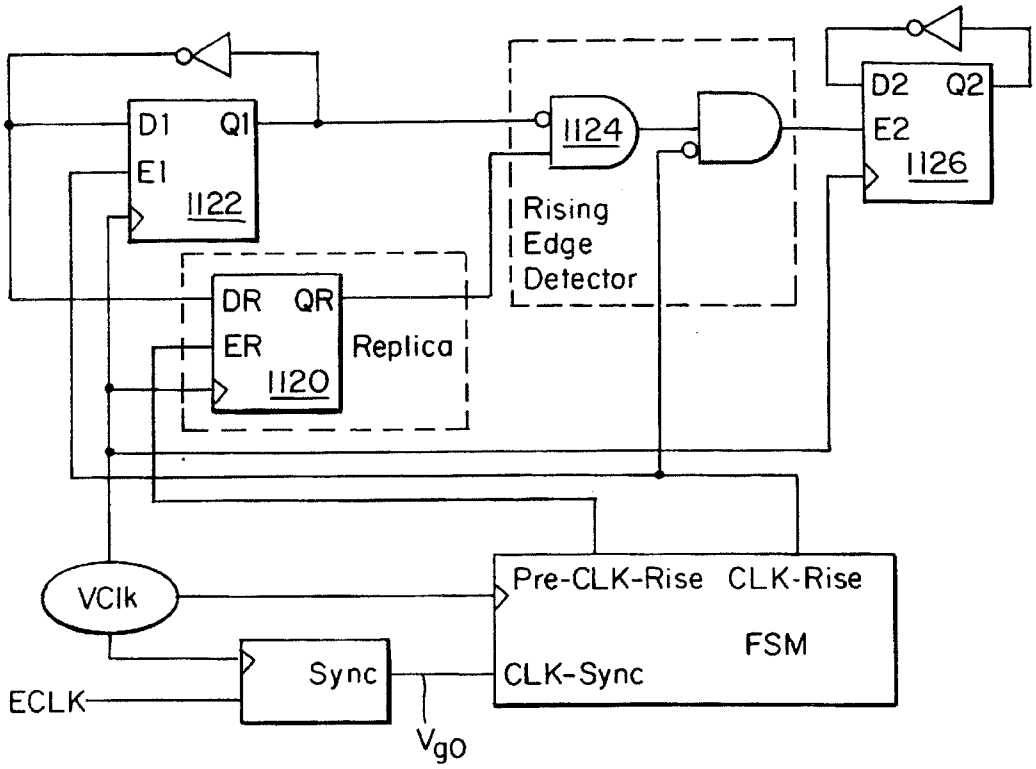


FIG. 11B

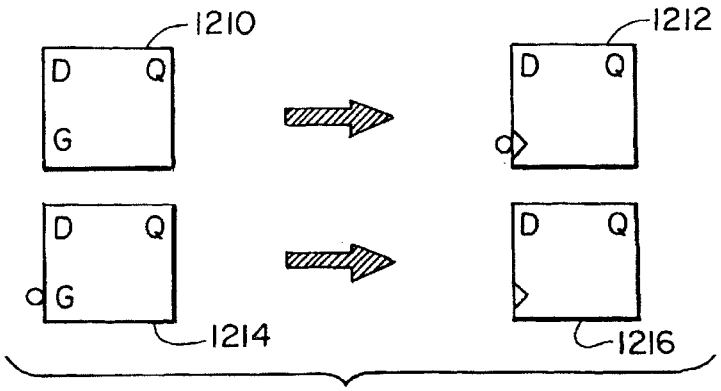


FIG. 12

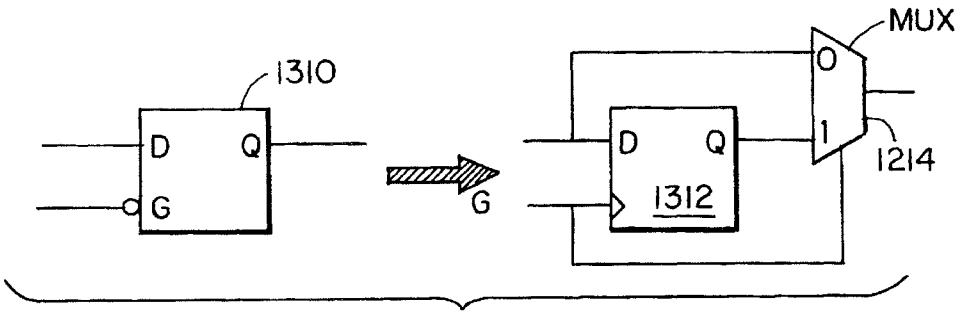


FIG. 13

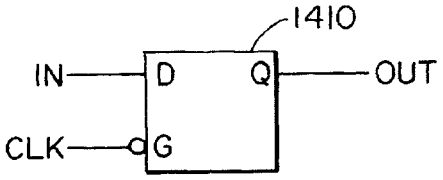


FIG. 14A

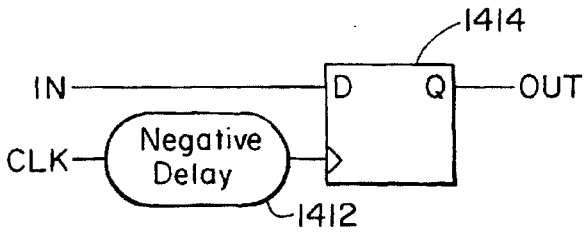


FIG. 14B

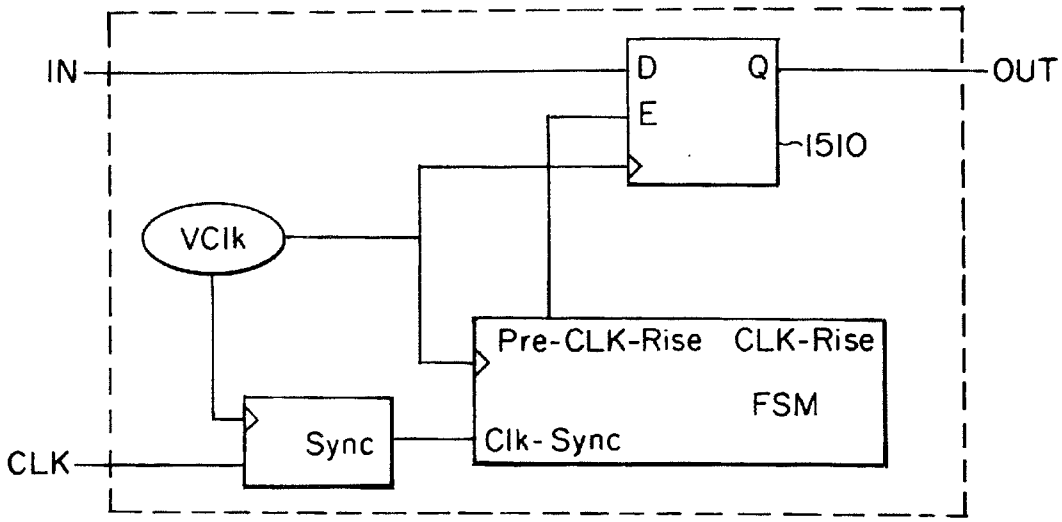


FIG. 15

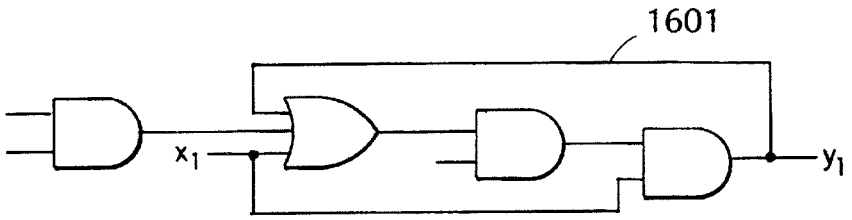


FIG. 16A

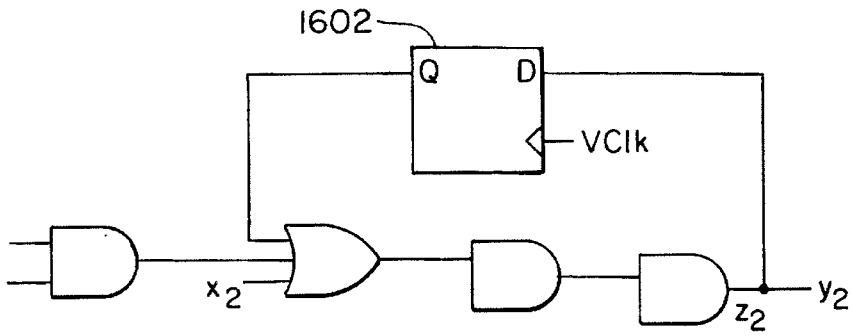


FIG. 16B

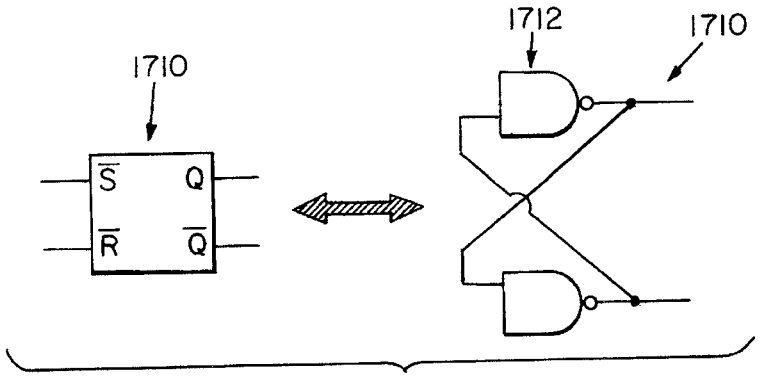


FIG. 17A

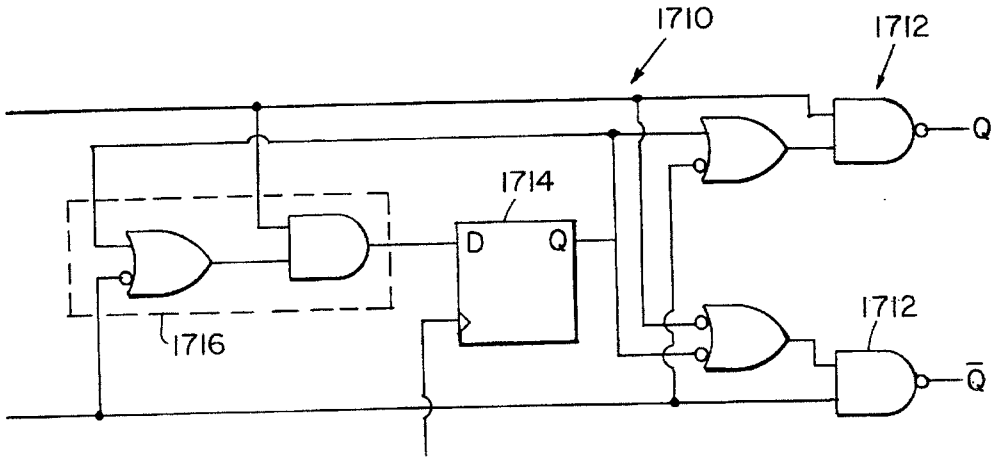


FIG. 17B

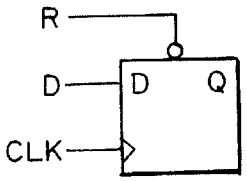


FIG. 18A

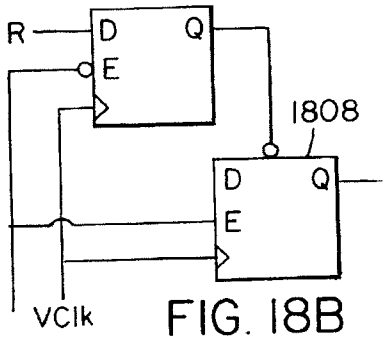


FIG. 18B

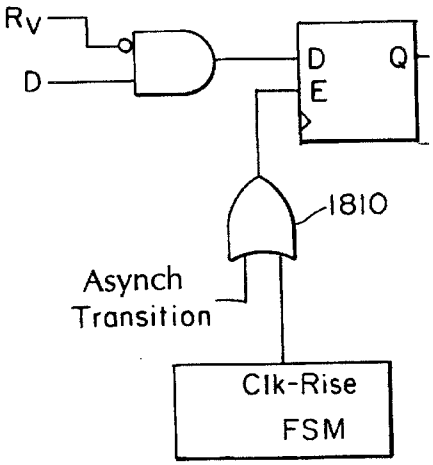


FIG. 18C

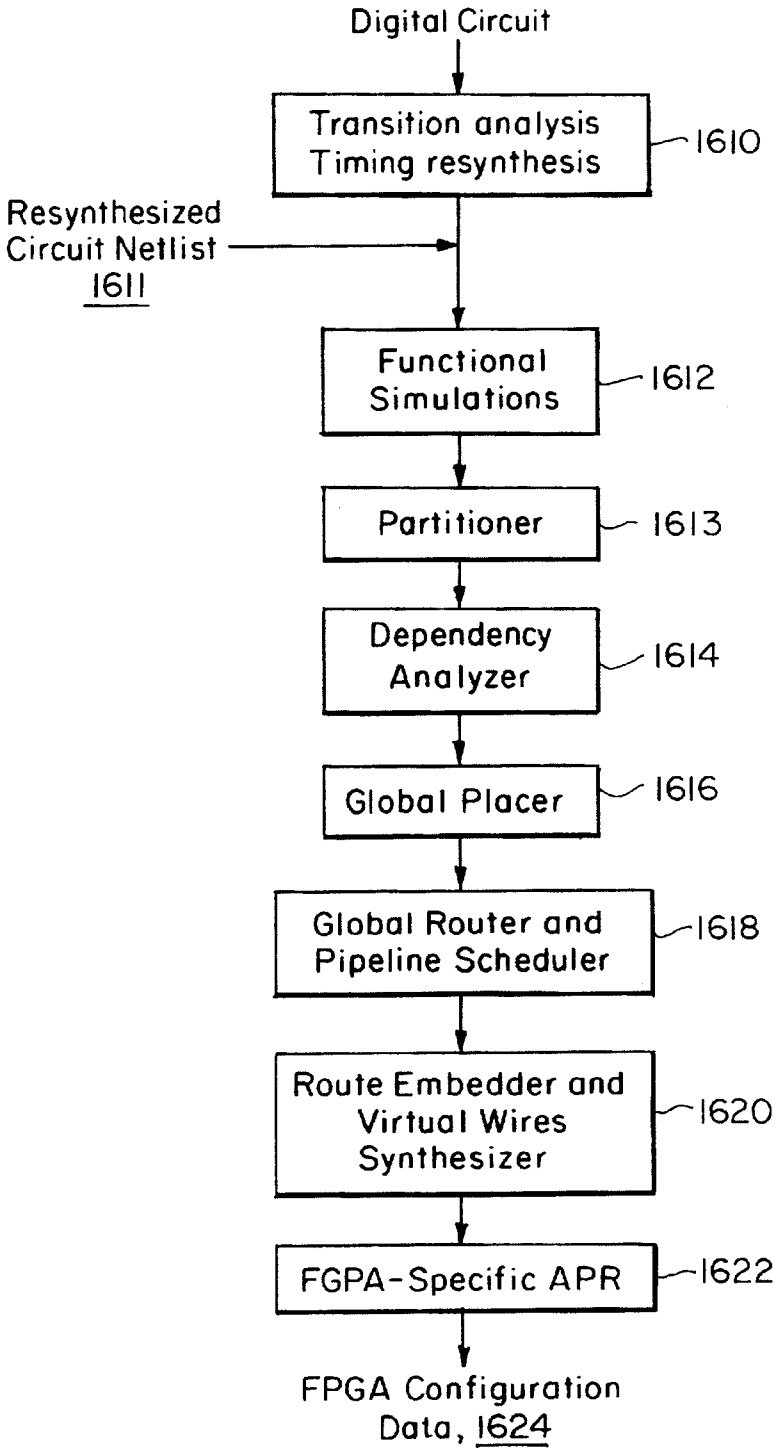

Digital Circuit
Transition analysis
Timing resynthesis
1610
Resynthesized
Circuit Netlist
1611
Functional
Simulations
1612
Partitioner
1613
Dependency
Analyzer
1614
Global Placer
1616
Global Router and
Pipeline Scheduler
1618
Route Embedder and
Virtual Wires
Synthesizer
1620
FGPA-Specific APR
1622
FPGA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

FIG. 19

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

RELATED APPLICATION

This application is a continuation of application Ser. No. 08/513,605 filed Aug. 10, 1995, now U.S. Pat. No. 5,649,176, which is incorporated herein by reference in its entirety.

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, he typical approach is somewhat ad hoc. Delay is added on he 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. **9**A and **9**B 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. **10**A, **10**B, and **10**C 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. **10**B is a timing diagram showing the signal values over time in the circuit;

FIGS. **11**A and **11**B 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. **14**A and **14**B 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. **14**B into a flip-flop that operates synchronously with the internal clock signal;

FIGS. **16**A and **16**B 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. **17**A and **17**B illustrate a timing resynthesis circuit transformation in which an RS flip-flop is transformed into a device that is synchronous with the virtual clock;

FIGS. **18**A, **18**B, and **18**C 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 **214***a*–**214***d*. 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 **214***a*–**214***d* 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 **214***a*–**214***d*, 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 EClk1 and EClk2, 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_{G01}, V_{G02} in response to an edge of the respective environmental timing signal EClk1 and EClk2, 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 EClk1 and a second environmental clock signal EClk2. 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 EClk1 and EClk2 are rising edge-active. The signals V_{G01} and V_{G02} 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 EClk1 and EClk2, upon the next or a later rising edge of the internal clock. For example, the second synchronizing signal V_{G02} is active in response to the receipt of the second environmental clock signal EClk2 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_{G01} V_{G02} 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_{G01} V_{G02}. 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_{G01} V_{G02} to each chip, and generate the multiple control signals on-chip to preserve the interconnect for other signal transmission.

FIG. **5**A 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. **5**A 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. **5**A 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_{G0} 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. **5**B is a timing diagram showing the timing of the signals in the circuit of FIG. **5**A. 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_{G0} is essentially the version of the environmental clock signal that is synchronized to the internal clock signal. Specifically, the new signal V_{G0} 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. **5**A, 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 **610***c*. 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 0, 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 ECLK 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_{G0} 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_{G0} 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_{G0} is generated by a synchronizer SYNC in response to receiving the environmental timing signal EClk 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 EClk 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 EClk**0** and EClk**1**. For the purposes of this description, both environmental clock signals EClk**0** and EClk**1** 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 SYNC**1** and SYNC**2** 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 EClk**0** and the second environmental clock signal EClk**1** are synchronized to the internal clock by the first synchronizer SYNC**0** and the second synchronizer SYNC**1**. The synchronizing signals V_{G00} and V_{G01} are generated by the synchronizers SYNC**0** and SYNC**1** to the finite state machine FSM. The finite state machine FSM watches for the synchronizing signals V_{G00} and V_{G01} and then produces a distinct load enable pulse C**0**-Rise, C**0**-Fall, C**1**-Rise for each timing edge on which the clocks EClk**0** and EClk**1** 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 C**0**-Rise, C**0**-Fall, and C**1**-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 CLK**0** and the second clock signal CLK**1** 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 FSM**0** and FSM**1** is assigned to each domain. specifically, a first finite state machine FSM**0** is synchronized to the first environmental clock EClk**0** to generate the load enable signal to the load enable terminal E**0** of the first flip-flop **920**. The second finite state machine FSM**1** generates a load enable signal to E**1** of the second flip-flop **922** in response to the second environmental clock signal EClk**1**. It should be noted, however, that although FSM**0** and FSM**1** 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 EClk**0** and EClk**1**, 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. **10**A shows a circuit that exhibits a simple gated clock behavior. FIG. **10**B 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 EClk, 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 EClk. As a result, only when the environmental clock EClk 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. **10**C shows the transformed circuit. Here, a finite state machine FSM receives a signal V_{G0} from the synchronizer SYNC upon receipt of the environmental clock EClk. 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 EClk.

The transformed circuit functions as follows. On the first period of the internal clock VClk after the falling edge of the environmental clock signal EClk, 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 EClk. 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. **11**A shows two cascaded edge-triggered flip-flops **1100** and **1112**. This configuration is generally known as a frequency divider. The environmental clock signal EClk 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 EClk. The divider of FIG. **11**A 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 EClk 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 EClk.

FIG. **11**B shows the transformed circuit of FIG. **11**A. 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 EClk. 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 EClk, 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. **14**A and **14**B, the latch **1410** of FIG. **14**A is replaced by a flip-flop which receives the phase-advanced clock indicated by the negative delay **1412** as shown in FIG. **14**B. 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. **14**B, 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 EClk. 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 EClk 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. **16**A, 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. **16**B.

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. **16**B 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. **17**A.

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

The circuit is FIG. **17**B forces the outputs Q and $\overline{Q}$ 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. **18**A 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. **18**B, 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. **18**C 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:

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

configuring the logic system to comprise a finite state machine for generating control signals to control the logic operations in response to the environmental timing signal and the internal clock signal.

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

4. A method of configuring as described in claim **1**, further comprising configuring the finite stare machine to dictate set-up and hold times of signals to the environment.

5. A method of configuring as described in claim **1**, further comprising configuring the finite state machine to dictate sampling times of the environmental data signals.

6. 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.

7. A method of configuring as described in claim **6**, further comprising configuring the finite state machine to generate control signals to the sequential logic in response to the environmental timing signal and the internal clock signal.

8. A method of configuring as described in claim **7**, 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.

9. 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.

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

11. A method as configuring as described in claim **9**, wherein the interconnect utilizes a direct-connect topology.

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

13. 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 finite state machine which coordinates operation of the logic in response to the internal clock signal and the environmental timing signal.

14. A logic system as described in claim **13**, wherein the at least one configurable logic device comprises at least one field programmable gate array.

15. A logic system as described in claim **13**, further comprising an interconnect for transmitting signals between plural configurable logic devices.

16. A logic system as described in claim **13**, further comprising a synchronizer for sampling the environmental timing signal in response to the internal clock signal.

17. A logic system as described in claim **16**, wherein the synchronizer is constructed from non-programmable logic.

18. A logic system as described in claim **13**, wherein the logic comprises combinational logic and sequential logic.

19. A logic system as described in claim **18**, 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.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,009,531
DATED : December 28, 1999
INVENTOR(S) : Charles W. Selvidge and Matthew L. Dahl

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 2, line 34, replace "he" with --the--.
In column 2, line 35, replace "he" with --the--.
In column 22, line 24, replace "stare" with --state--.

Signed and Sealed this

Twenty-fifth Day of July, 2000

Attest:

Attesting Officer

Q. TODD DICKINSON

Director of Patents and Trademarks

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).

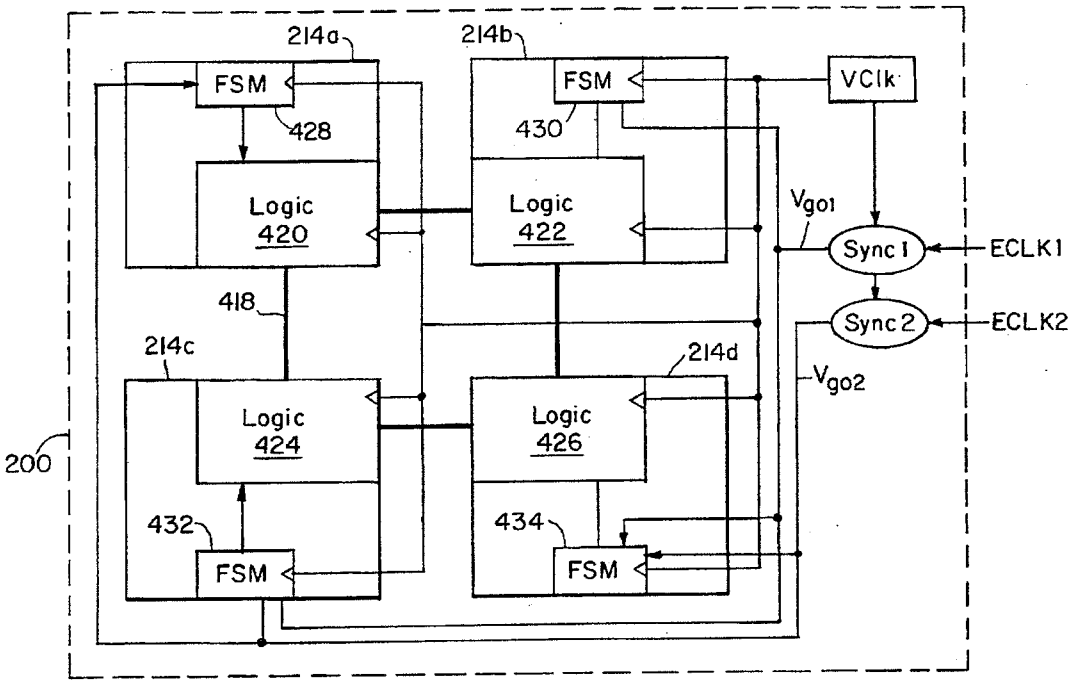
Primary Examiner—Thomas M. Heckler

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



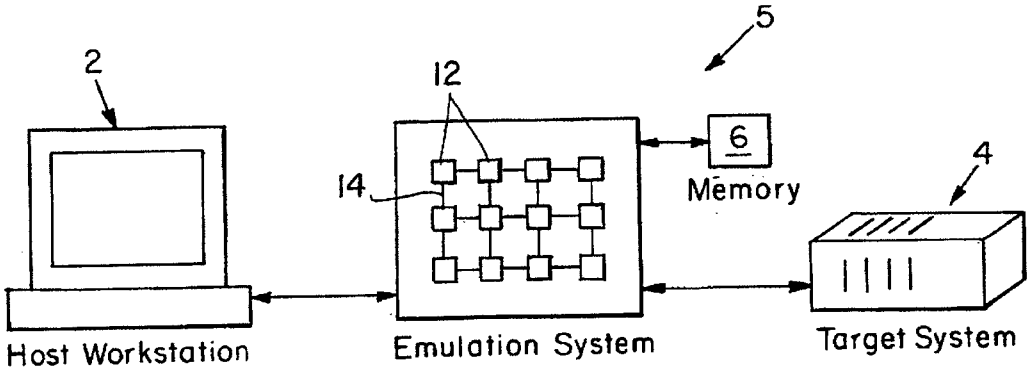


FIG. 1
(Prior Art)

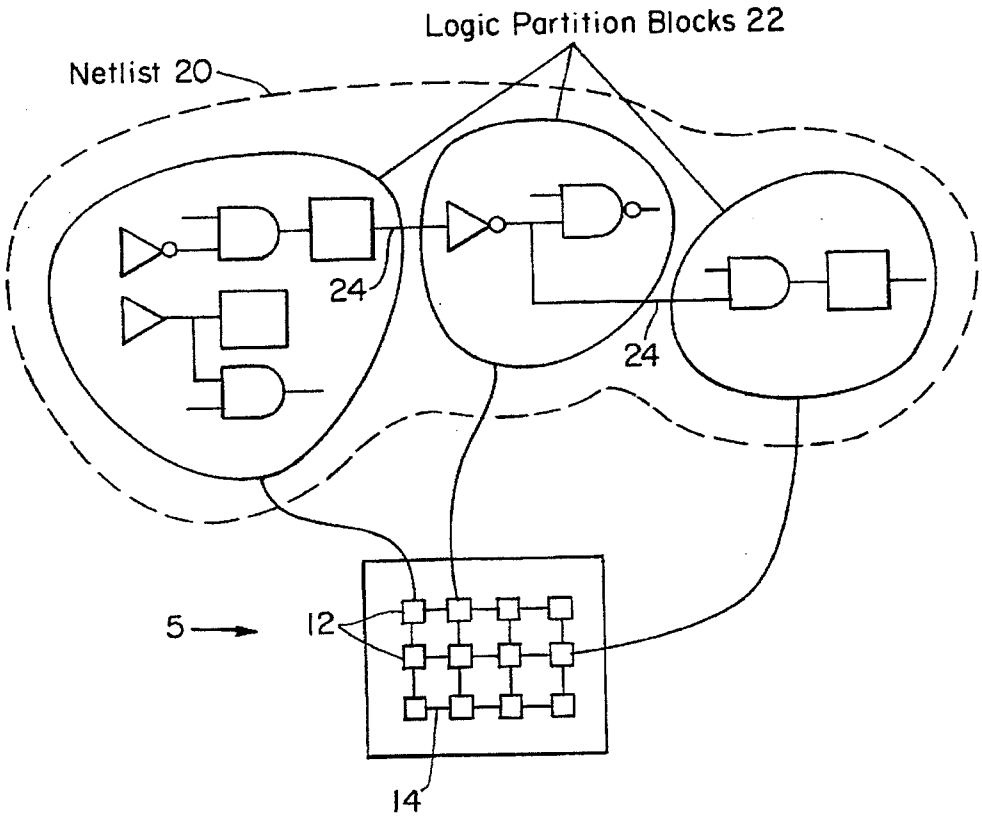


FIG. 2

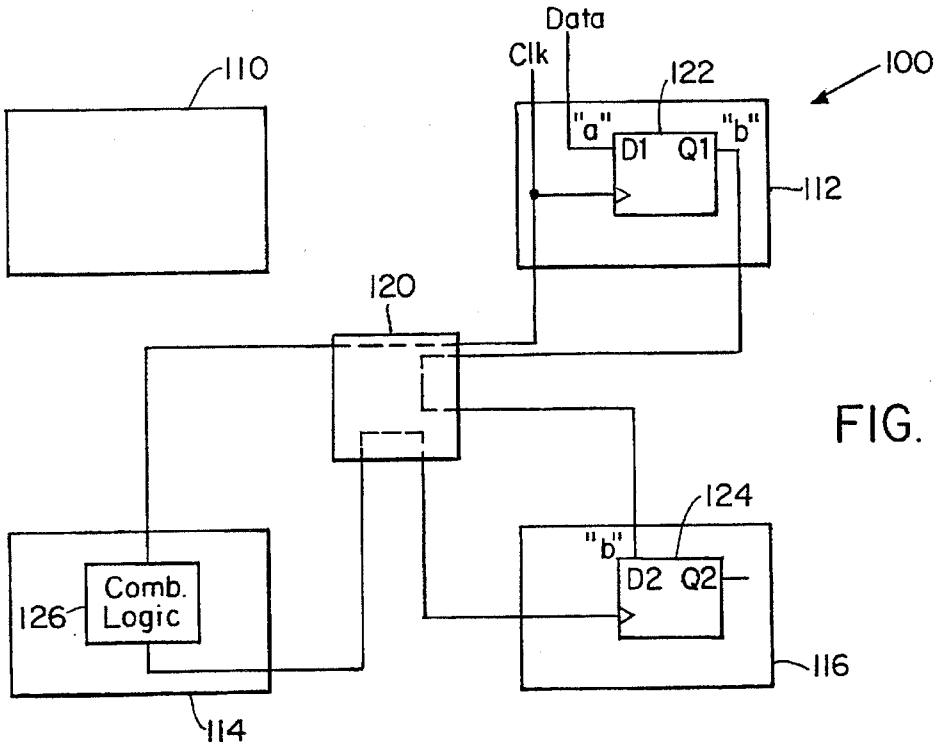

Data
Clk
110
122
100
"a"
"b"
D1
Q1
112
120
124
"b"
D2
Q2
Comb.
Logic
126
116
114

FIG. 3

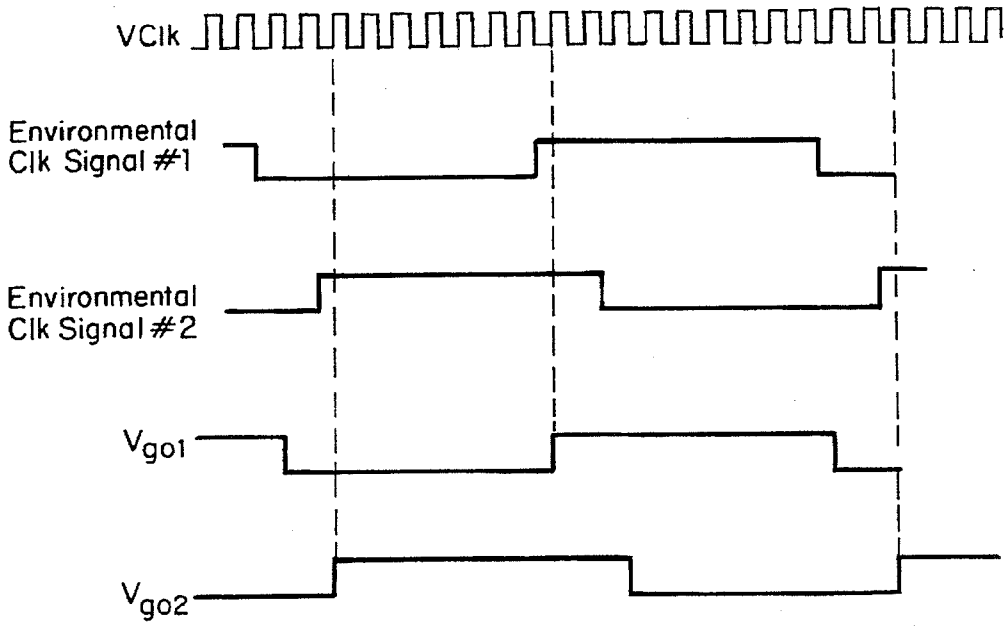

VClk
Environmental
Clk Signal #1
Environmental
Clk Signal #2
Vgo1
Vgo2

FIG. 4B

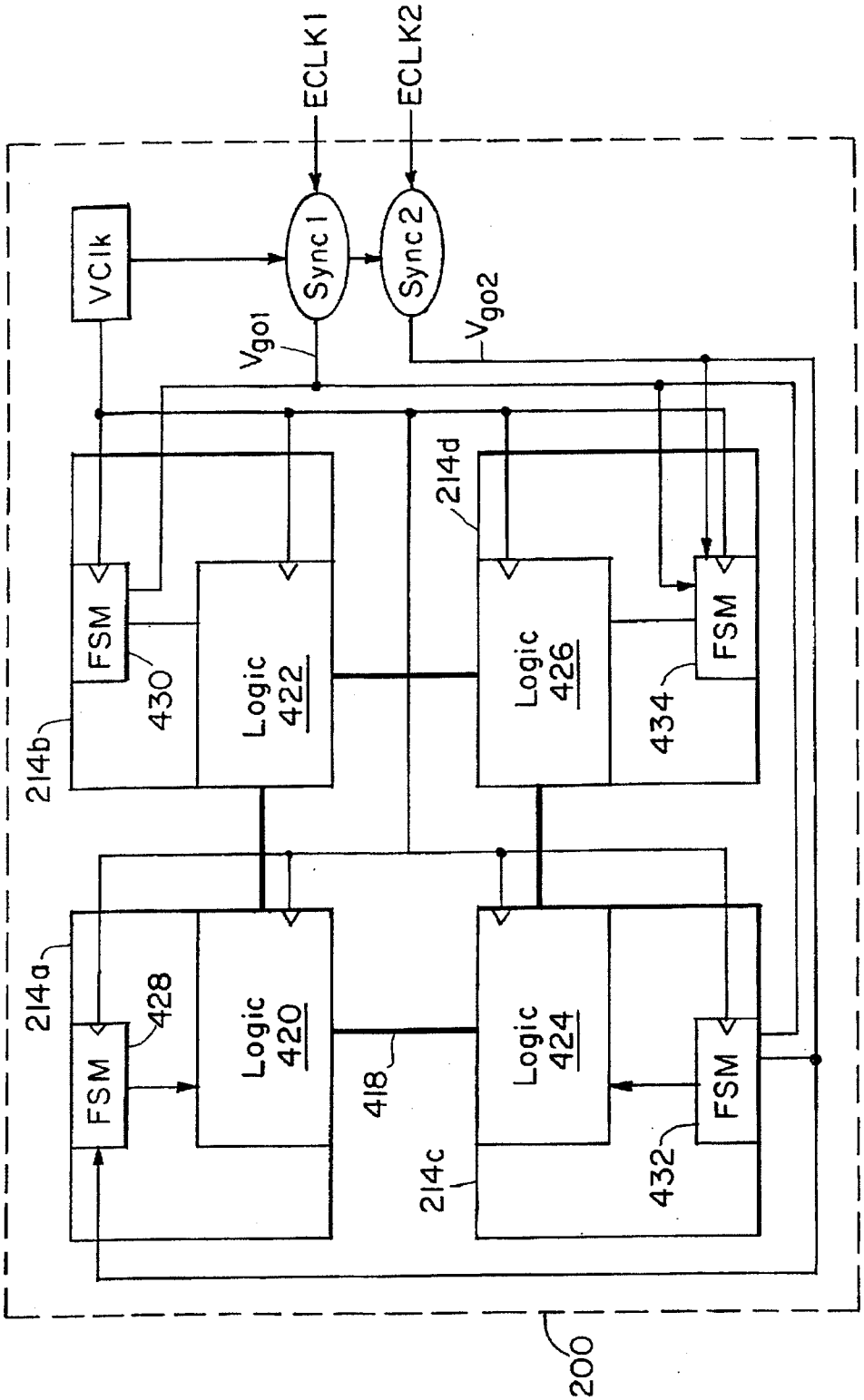

ECLK1
ECLK2
VClk
Sync 1
Sync 2
V_{go1}
V_{go2}
214d
FSM
430
Logic 422
214b
Logic 426
FSM
434
214a
428
FSM
Logic 420
418
Logic 424
FSM
214c
432
200

FIG. 4A

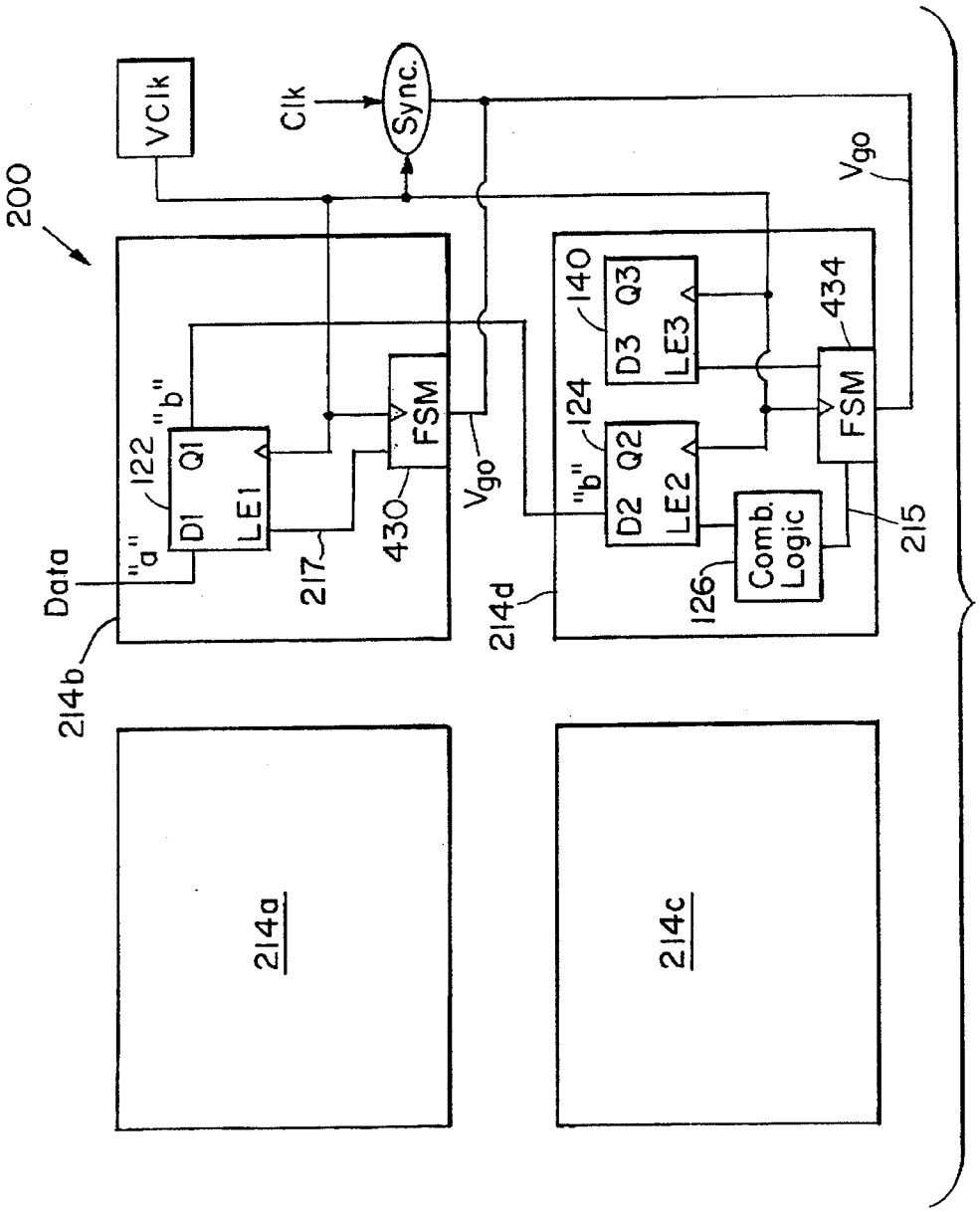

200
VClk
Clk
Sync.
Vgo
Data
"a"
"b"
122
D1 Q1
LE1
FSM
217
430
Vgo
140
D3 Q3
LE3
434
124
"b"
D2 Q2
LE2
FSM
Comb. Logic
126
215
214b
214d
214a
214c

FIG. 5A

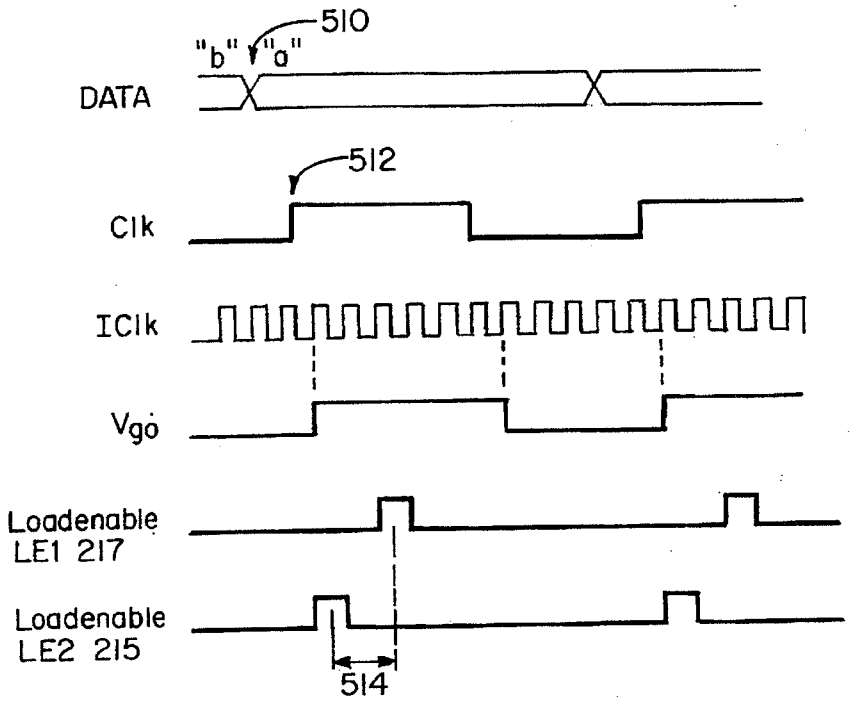

510
"b" "a"
DATA
512
Clk
ICIk
Vgo
Loadenable
LE1 217
Loadenable
LE2 215
514

FIG. 5B

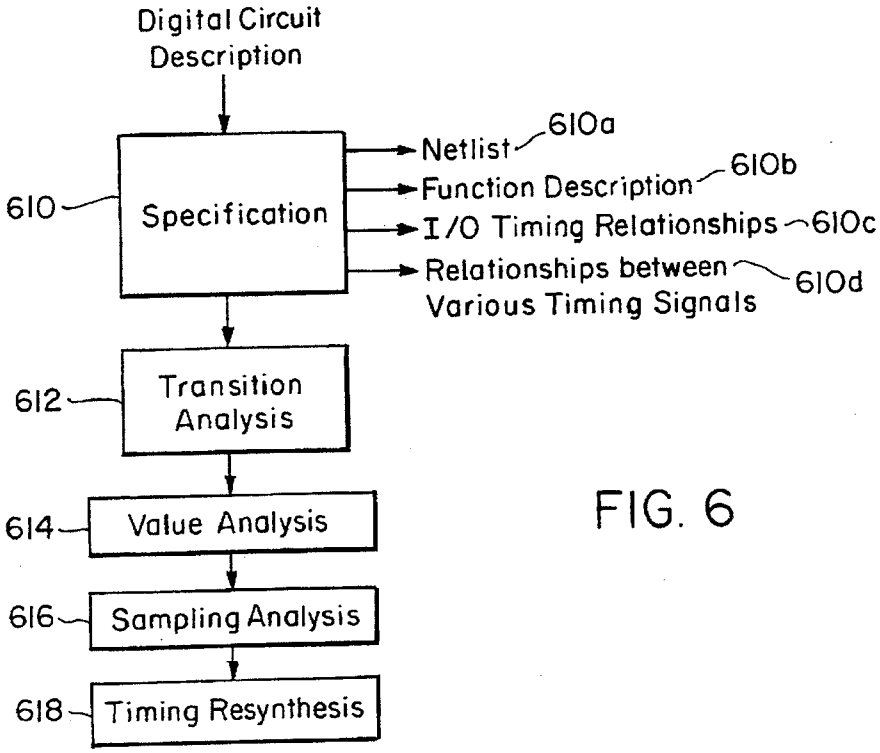

Digital Circuit
Description
610
Specification
Netlist
610a
Function Description
610b
I/O Timing Relationships
610c
Relationships between
Various Timing Signals
610d
612
Transition
Analysis
614
Value Analysis
616
Sampling Analysis
618
Timing Resynthesis

FIG. 6

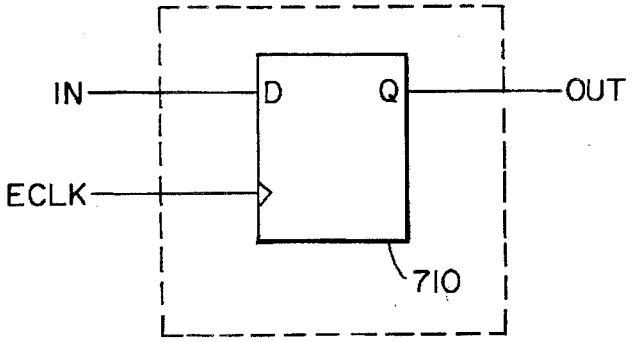

IN
D
Q
OUT
ECLK
710

FIG. 7A

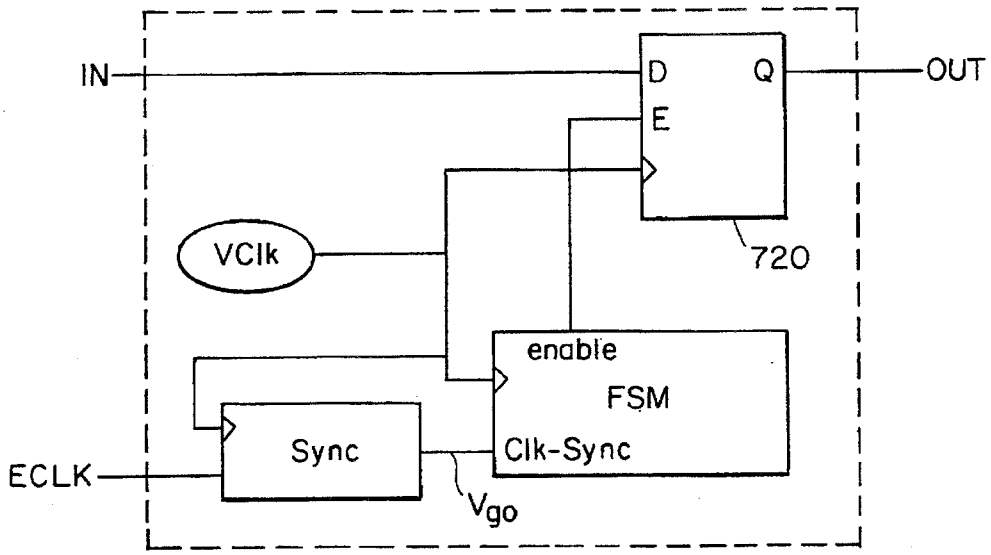

IN
D
Q
OUT
E
VClk
720
enable
FSM
Sync
Clk-Sync
ECLK
Vgo

FIG. 7B

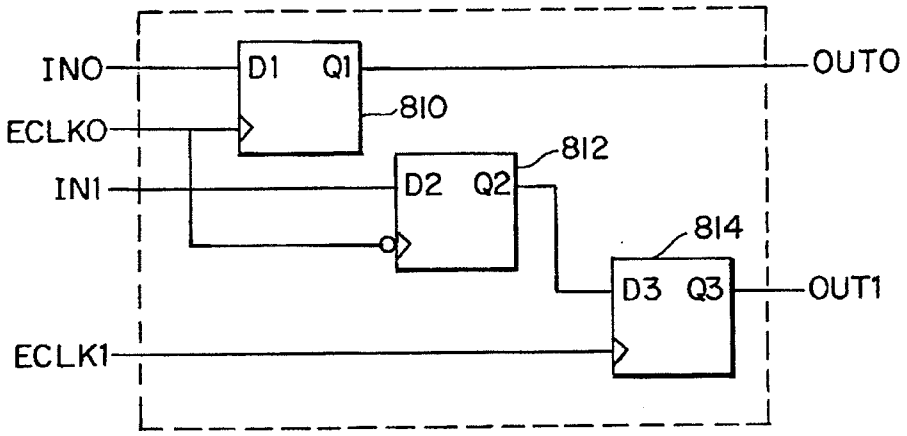

INO
ECLKO
IN1
ECLK1
D1 Q1
810
D2 Q2
812
D3 Q3
814
OUTO
OUT1

FIG. 8A

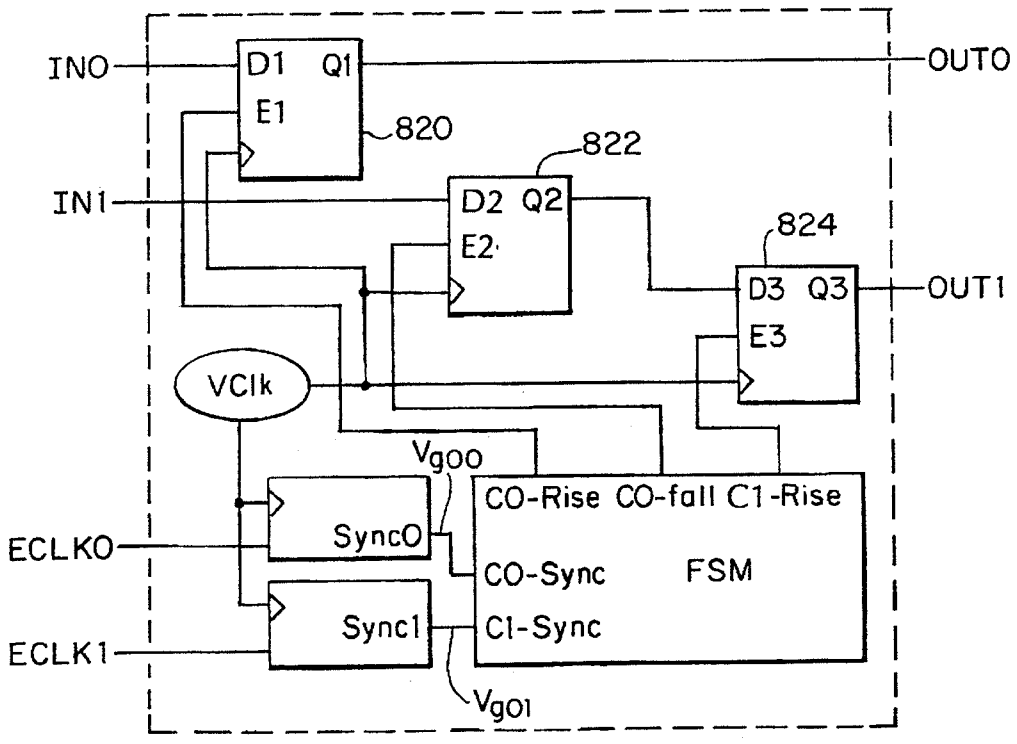

INO
IN1
ECLKO
ECLK1
D1 Q1
E1
820
D2 Q2
E2
822
D3 Q3
E3
824
OUTO
OUT1
VClk
Vgoo
Vgo1
Sync0
Sync1
CO-Rise CO-fall C1-Rise
CO-Sync
C1-Sync
FSM

FIG. 8B

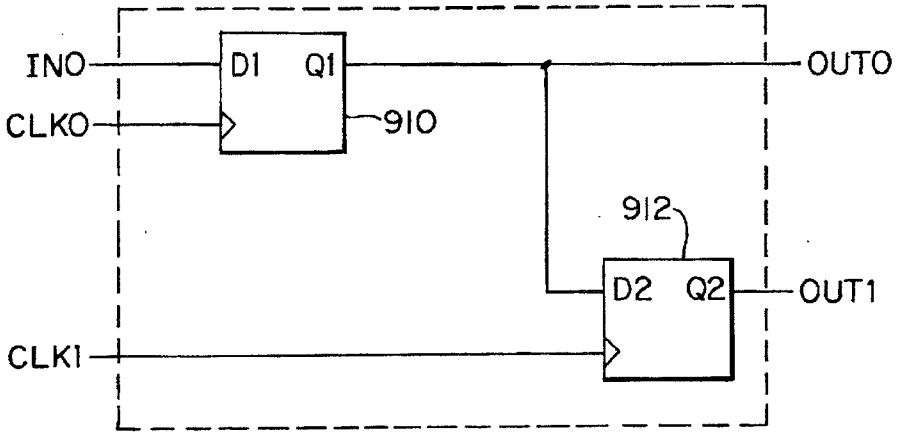

INO
CLKO
D1
Q1
910
OUTO
912
D2
Q2
OUT1
CLK1

FIG. 9A

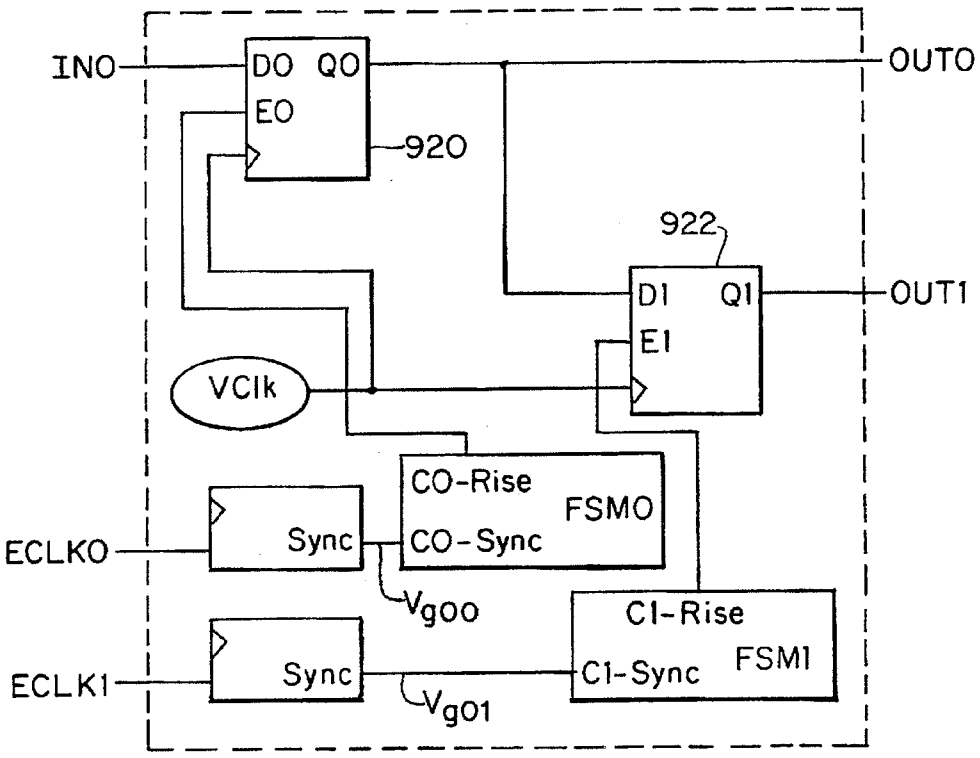

INO
DO
QO
EO
920
OUTO
922
D1
Q1
E1
OUT1
VClk
CO-Rise
FSMO
CO-Sync
Sync
ECLKO
Vgoo
C1-Rise
FSM1
C1-Sync
Sync
ECLK1
Vgo1

FIG. 9B

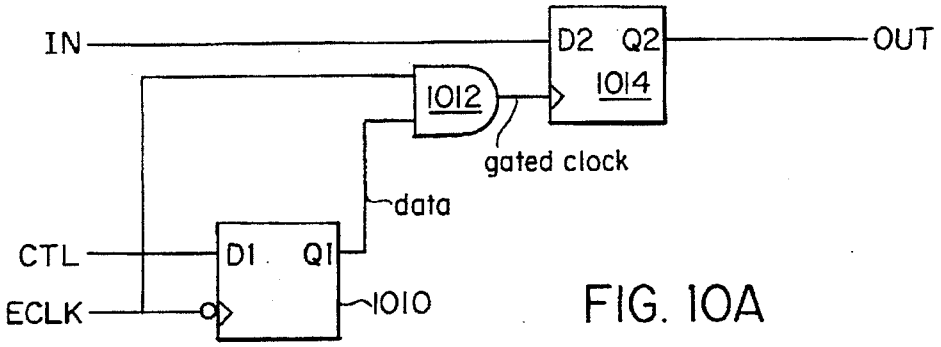


FIG. 10A

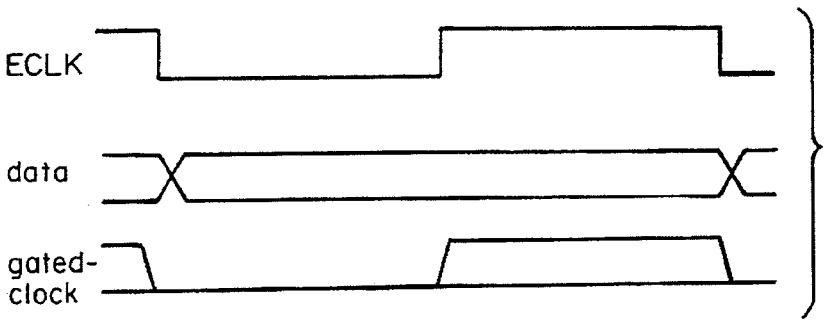


FIG. 10B

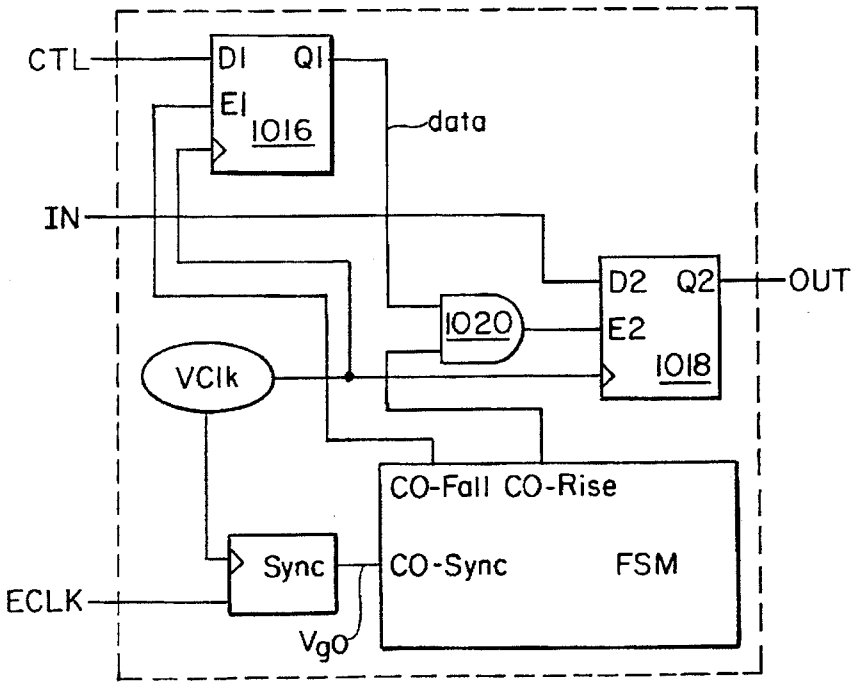


FIG. 10C

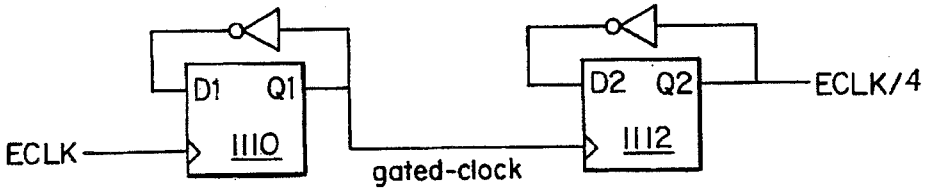


FIG. 11A

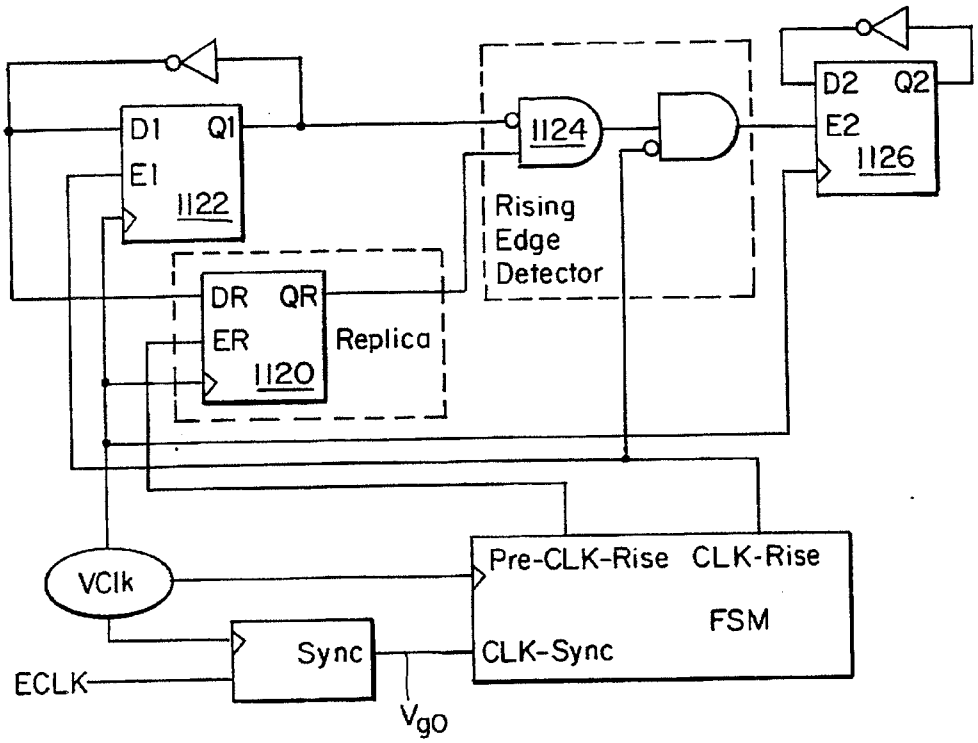


FIG. 11B

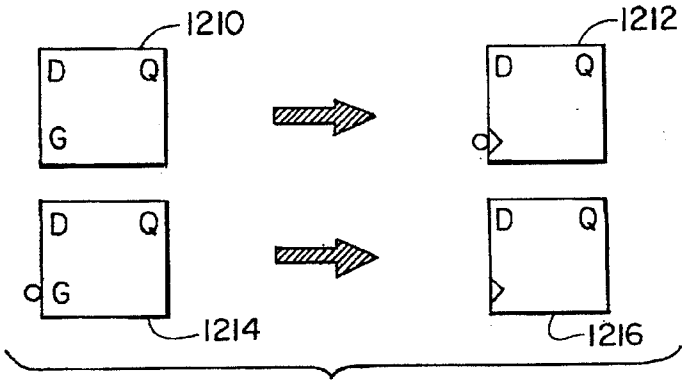


FIG. 12

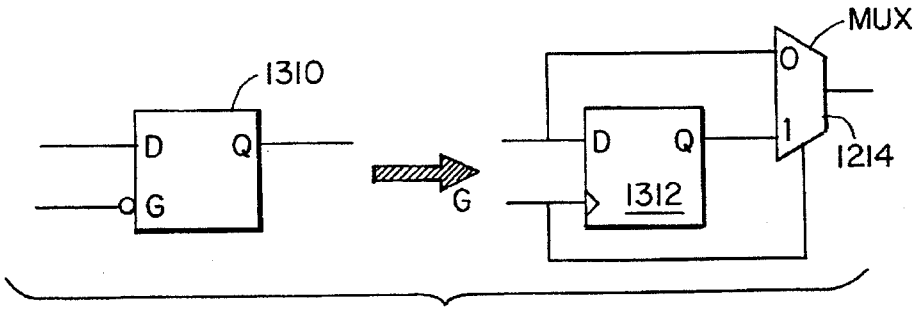


FIG. 13

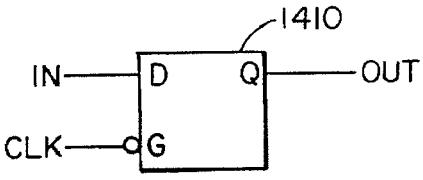


FIG. 14A

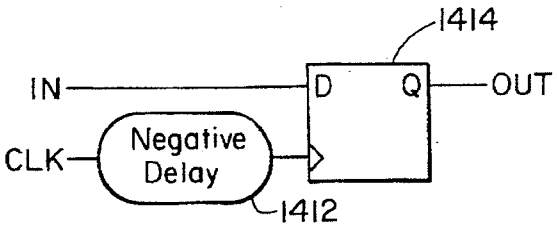


FIG. 14B

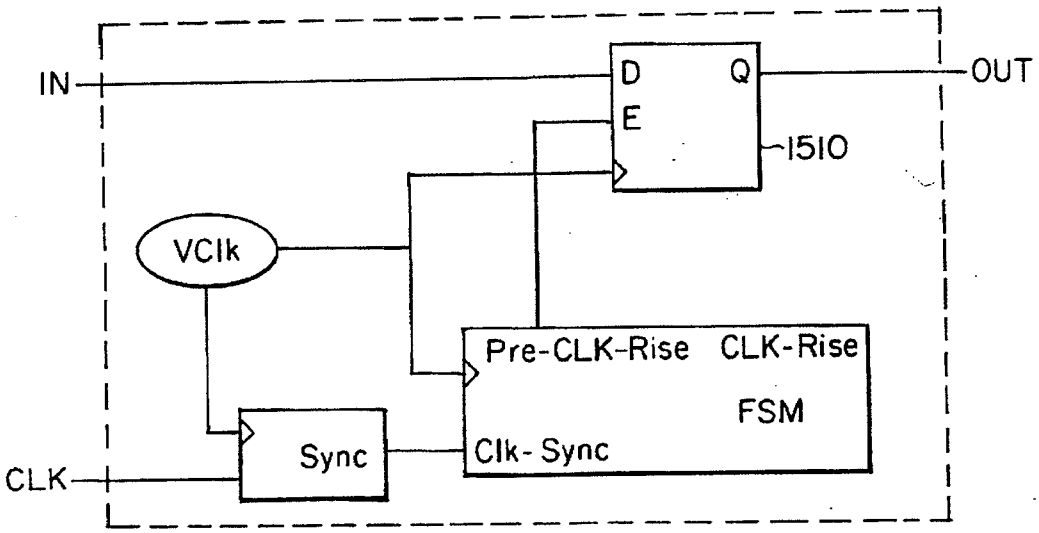


FIG. 15

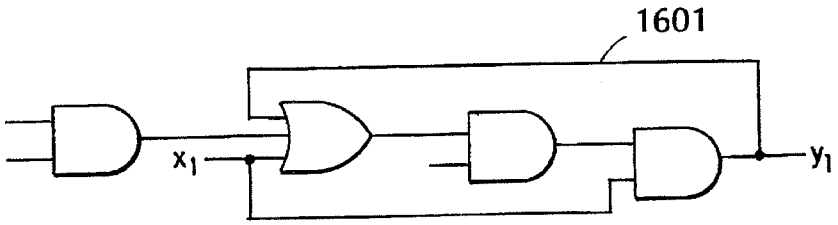


FIG. 16A

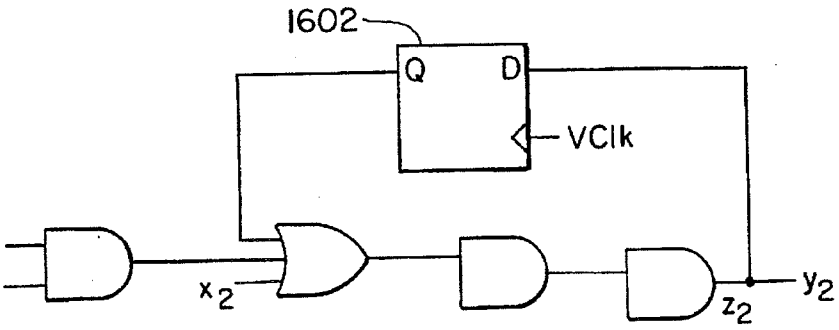


FIG. 16B

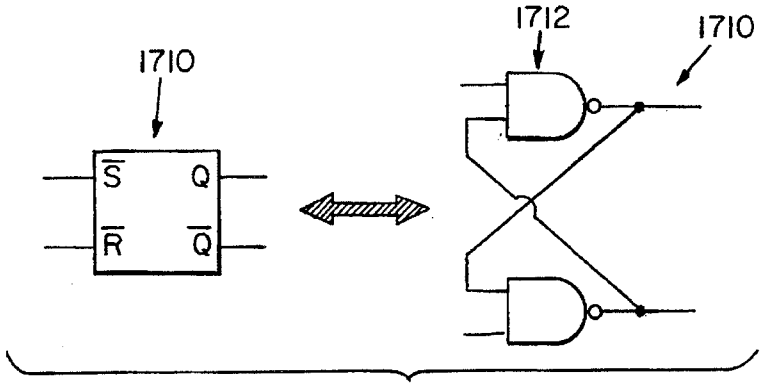


FIG. 17A

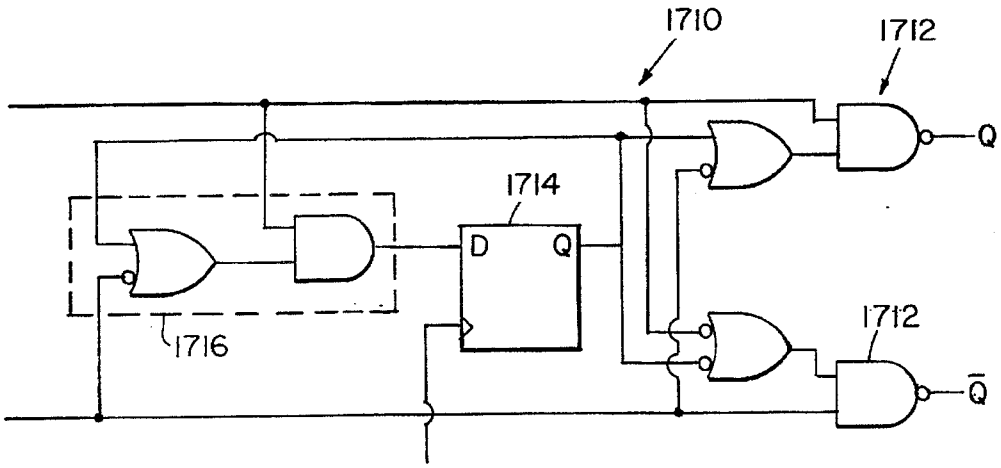


FIG. 17B

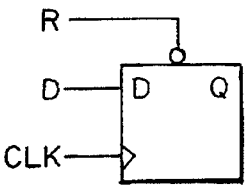


FIG. 18A

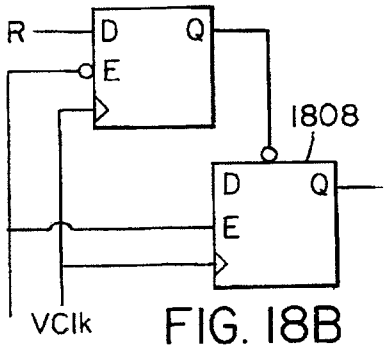


FIG. 18B

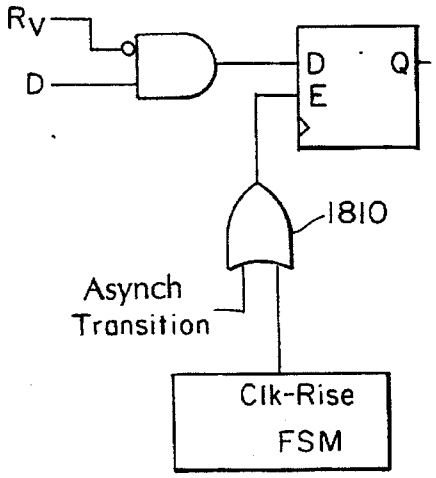


FIG. 18C

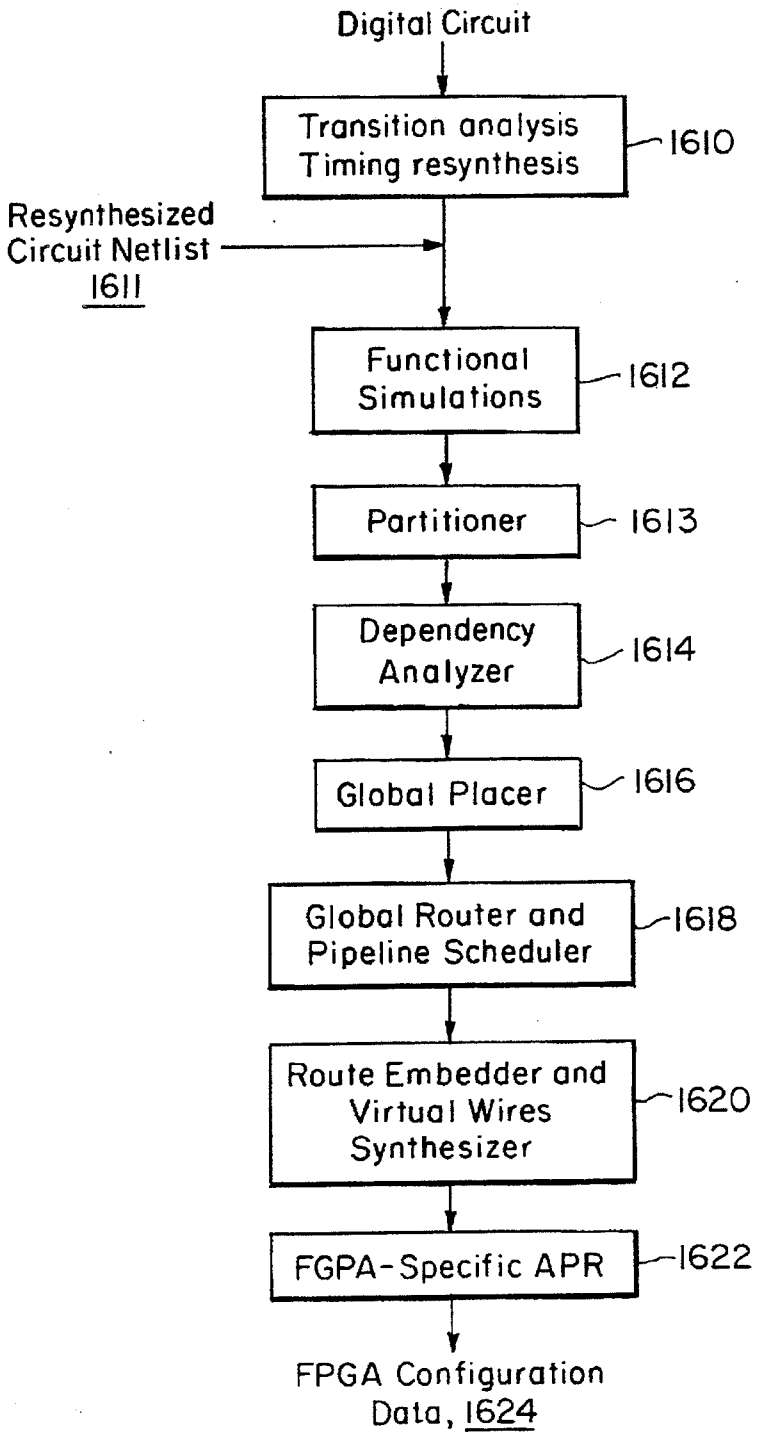

Digital Circuit
Transition analysis
Timing resynthesis
1610
Resynthesized
Circuit Netlist
1611
Functional
Simulations
1612
Partitioner
1613
Dependency
Analyzer
1614
Global Placer
1616
Global Router and
Pipeline Scheduler
1618
Route Embedder and
Virtual Wires
Synthesizer
1620
FGPA-Specific APR
1622
FPGA 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. **10**A, **10**B, and **10**C 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. **10**B is a timing diagram showing the signal values over time in the circuit;

FIG. **11**A and **11**B 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. **14**A and **14**B 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. **14**B into a flip-flop that operates synchronously with the internal clock signal;

FIGS. **16**A and **16**B 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. **17**A and **17**B 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. **18**A, **18**B, and **18**C 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 **214***a*–**214***d*. 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 **214***a*–**214***d* 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 **214***a*–**214***d*, 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 EClk1 and EClk2, 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_{G01}, V_{G02} in response to an edge of the respective environmental timing signal EClk1 and EClk2, 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 EClk1 and a second environmental clock signal EClk2. 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 EClk1 and EClk2 are rising edge-active. The signals V_{G01} and V_{G02} 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 EClk1 and EClk2, upon the next or a later rising edge of the internal clock. For example, the second synchronizing signal V_{G02} is active in response to the receipt of the second environmental clock signal EClk2 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_{G01} V_{G02} 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_{G01} V_{G02}. 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_{GO1} V_{GO2} to each chip, and generate the multiple control signals on-chip to preserve the interconnect for other signal transmission.

FIG. **5**A 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. **5**A 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. **5**A 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_{GO} 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. **5**B is a timing diagram showing the timing of the signals in the circuit of FIG. **5**A. 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_{GO} is essentially the version of the environmental clock signal that is synchronized to the internal clock signal. Specifically, the new signal V_{GO} 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. **5**A, 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 **610***c*. 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 ECLK 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_{GO} 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_{GO} 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_{GO} is generated by a synchronizer SYNC in response to receiving the environmental timing signal EClk 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 EClk 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 EClk**0** and EClk**1**. For the purposes of this description, both environmental clock signals EClk**0** and EClk**1** 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 EClk**0** and the second environmental clock signal EClk**1** are synchronized to the internal clock by the first synchronizer SYNC**0** and the second synchronizer SYNC**1**. The synchronizing signals V_{GO0} and V_{GO1} are generated by the synchronizers SYNC**0** and SYNC**1** to the finite state machine FSM. The finite state machine FSM watches for the synchronizing signals V_{GO0} and V_{GO1} and then produces a distinct load enable pulse C**0**-Rise, C**0**-Fall, C**1**-Rise for each timing edge on which the clocks EClk**0** and EClk**1** 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 C**0**-Rise, C**0**-Fall, and C**1**-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 CLK**0** and the second clock signal CLK**1** 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 FSM**0** and FSM**1** is assigned to each domain. Specifically, a first finite state machine FSM**0** is synchronized to the first environmental clock EClk**0** to generate the load enable signal to the load enable terminal E**0** of the first flip-flop **920**. The second finite state machine FSM**1** generates a load enable signal to E**1** of the second flip-flop **922** in response to the second environmental clock signal EClk**1**. It should be noted, however, that although FSM**0** and FSM**1** 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 EClk**0** and EClk**1**, 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 FSM**0** and the second finite state machine FSM**1** 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 FSM**0** and the second finite state machine FSM**1** 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. **10**A 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 EClk, 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 EClk. As a result, only when the environmental clock EClk 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. **10**C 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 EClk. 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 EClk.

The transformed circuit functions as follows. On the first period of the internal clock VClk after the falling edge of the environmental clock signal EClk, 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 EClk. 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. **11**A shows two cascaded edge-triggered flip-flops **1100** and **1112**. This configuration is generally known as a frequency divider. The environmental clock signal EClk 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 EClk. The divider of FIG. **11**A 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 EClk 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 EClk.

FIG. **11**B shows the transformed circuit of FIG. **11**A. 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 EClk. 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 EClk, 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 EClk. 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 EClk 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 $\overline{Q}$ 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 set-up 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

US006240376B1

(12) **United States Patent**
Raynaud et al.

(10) **Patent No.: US 6,240,376 B1**
(45) **Date of Patent: May 29, 2001**

(54) **METHOD AND APPARATUS FOR GATE-LEVEL SIMULATION OF SYNTHESIZED REGISTER TRANSFER LEVEL DESIGNS WITH SOURCE-LEVEL DEBUGGING**

(75) Inventors: **Alain Raynaud**, Paris; **Luc M. Burgun**, Creteil, both of (FR)

(73) Assignee: **Mentor Graphics Corporation**, Wilsonville, OR (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/127,584**

(22) Filed: **Jul. 31, 1998**

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/122,493, filed on Jul. 24, 1998.

(51) **Int. Cl.**[7] **G06F 17/50**
(52) **U.S. Cl.** **703/15**; 703/14; 714/741
(58) **Field of Search** 395/500.35, 500.36, 395/500.37; 714/724, 734; 703/14, 15, 16; 716/4, 724, 734, 741

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,220,512 6/1993 Watkins et al. 364/489
5,253,255 * 10/1993 Carbine 714/734

(List continued on next page.)

OTHER PUBLICATIONS

Chen et al., "A Source–Level Dynamic Analysis Methodology and Tool for High–LevelSynthesis", Proceedings of the Tenth International Symposium on System Synthesis, 1997, pp. 134–140, Sep. 1997.*

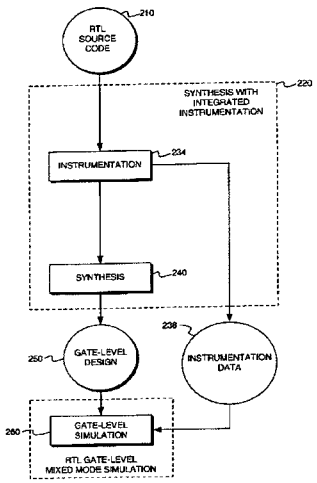
(List continued on next page.)

Primary Examiner—Kevin J. Teska
Assistant Examiner—Douglas W. Sergent
(74) *Attorney, Agent, or Firm*—Columbia IP Law Group, LLC

(57) **ABSTRACT**

Methods of instrumenting synthesizable source code to enable debugging support akin to high-level language programming environments for gate-level simulation are provided. One method of facilitating gate level simulation includes generating cross-reference instrumentation data including instrumentation logic indicative of an execution status of at least one synthesizable register transfer level (RTL) source code statement. A gate-level netlist is synthesized from the source code. Evaluation of the instrumentation logic during simulation of the gate-level netlist facilitates simulation by indicating the execution status of a corresponding source code statement. One method results in a modified gatelevel netlist to generate instrumentation signals corresponding to synthesizable statements within the source code. This may be accomplished by modifying the source code or by generating the modified gate-level netlist as if the source code was modified during synthesis. Alternatively, cross-reference instrumentation data including instrumentation logic can be generated without modifying the gate-level design. The instrumentation logic indicates the execution status of a corresponding cross-referenced synthesizable statement. An execution count of a cross-referenced synthesizable statement can be incremented when the corresponding instrumentation signals indicates the statement is active to determine source code coverage. Source code statements can be highlighted when active for visually tracing execution paths. For breakpoint simulation, a breakpoint can be set at a selected source code statement. The corresponding instrumentation logic from the cross-reference instrumentation data is implemented as a simulation breakpoint. The simulation is halted at a simulation cycle where the values of the instrumentation signals indicate that the source code statement is active.

33 Claims, 22 Drawing Sheets



U.S. PATENT DOCUMENTS

5,325,309		6/1994	Halaviati et al.	364/488
5,423,023		6/1995	Batch et al.	395/500
5,461,576		10/1995	Tsay et al.	364/490
5,513,123	*	4/1996	Dey et al.	716/4
5,519,627		5/1996	Mahamood et al.	364/488
5,541,849		7/1996	Rostoker et al.	364/489
5,544,067		8/1996	Rostoker et al.	364/489
5,553,002		9/1996	Dangelo et al.	364/489
5,555,201		9/1996	Dangelo et al.	364/489
5,568,396		10/1996	Bamji et al.	364/491
5,598,344		1/1997	Dangelo et al.	364/489
5,615,356		3/1997	King et al.	395/500
5,623,418		4/1997	Rostoker et al.	364/489
5,632,032	*	5/1997	Ault et al.	709/100
5,727,187		3/1998	Lemche et al.	395/500
5,758,123		5/1998	Sano et al.	395/500
5,768,145		6/1998	Roethig	364/488
5,801,958		9/1998	Dangelo et al.	364/489
5,835,380		11/1998	Roethig	364/488
5,841,663		11/1998	Sharma et al.	364/490
5,870,308		2/1999	Dangelo et al.	364/489
5,870,585	*	2/1999	Stapleton	703/15
5,880,971	*	3/1999	Dangelo et al.	703/16
5,920,711		7/1999	Seawright et al.	395/500
5,937,190		8/1999	Gregory et al.	395/704
5,960,191		9/1999	Sample et al.	395/500.49
5,991,533		11/1999	Sano et al.	395/500.49
6,006,022		12/1999	Rhim et al.	395/500.02
6,009,256		12/1999	Tseng et al.	395/500.34

OTHER PUBLICATIONS

Kucukcakar et al., "Matisse: An Architectural Design Tool for Commodity IC's", IEEE Design & Test of Computers, vol. 15, Issue 2, pp. 22–33, Jun. 1998.*

Koch et al. "Debugging of Behavioral VHDL Specifications by Source Level Emulation", Proceedings of the European Design Automation Conference, pp. 256–261, Sep. 1995.*

Fang et al., "A Real–Time RTL Engineering–Change Method Supporting On–Line Debugging for Logic–Emulation Applications", Proceedings of the 34th Design Automation Conference, pp. 101–106, Jun. 1997.*

Howe, H., "Pre– and Postsynthesis Mismatches", IEEE International Conf. on Verilog HDL 1997, pp. 24–31, Apr. 1997.*

Postula et al., "A Comparison of High Level Synthesis and Register Transfer Design Techniques for Custom Computing Machines", Proc. of the 31st Hawaii Inter. Conf. on System Sciences, vol. 7, pp. 207–214, Jan. 1998.*

Orailoglu, A., "Microarchitectural Sythesis for Rapid BIST Testing", IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems, vol. 16, Issue 6, pp. 573–586, Jun. 1997.*

* cited by examiner

FIG. 1
(Prior Art)

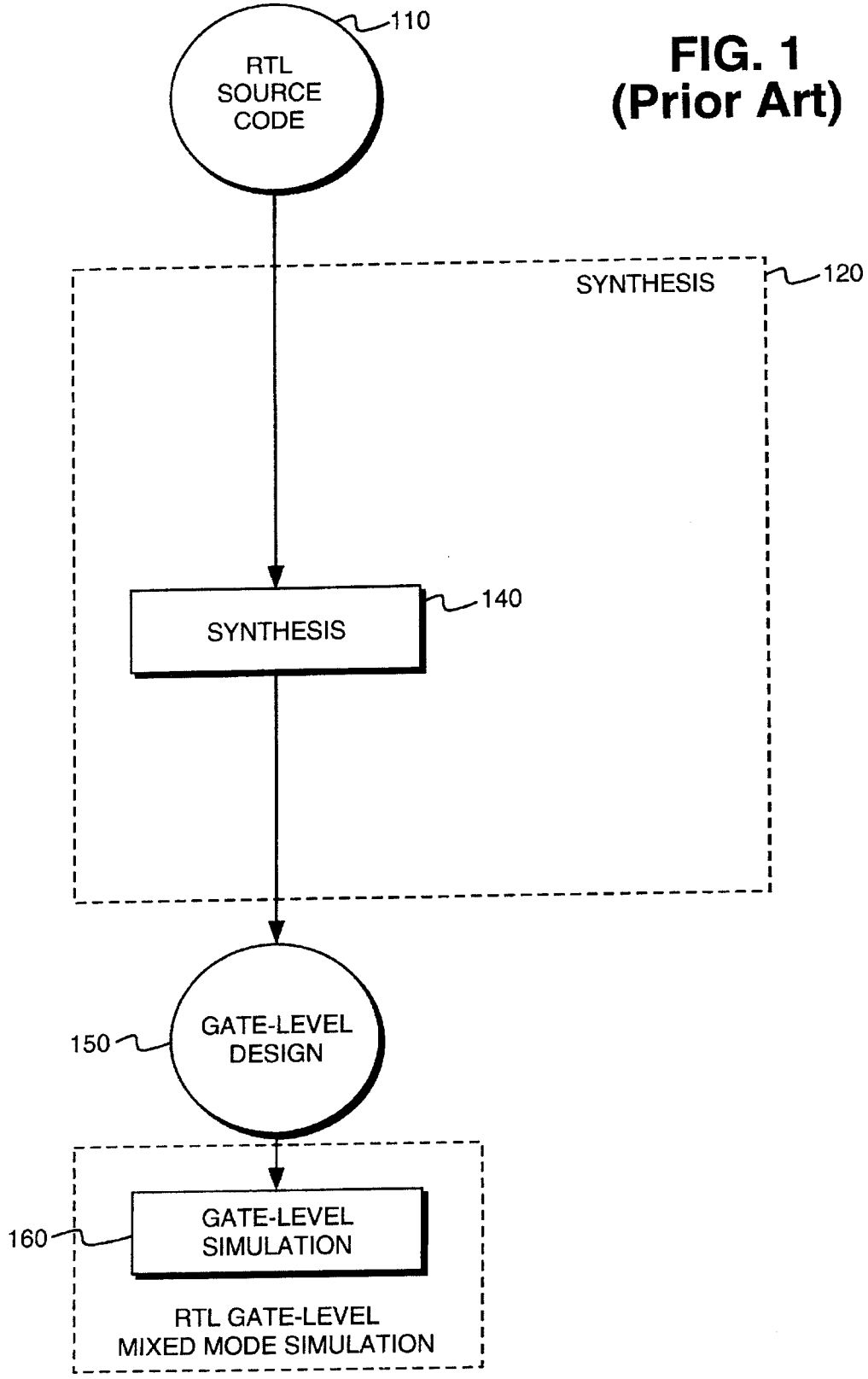


FIG. 2

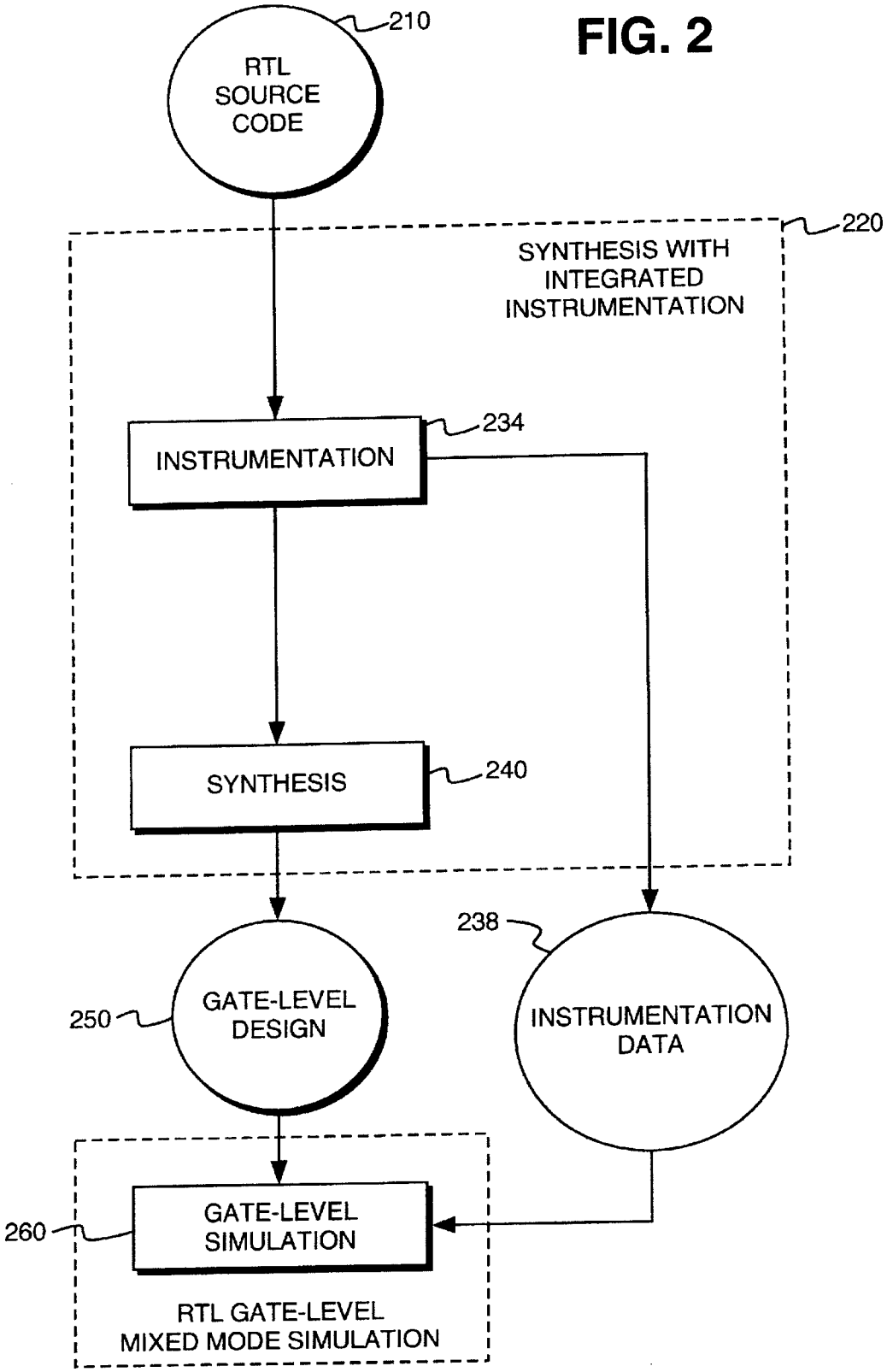

210
RTL SOURCE CODE
220
SYNTHESIS WITH INTEGRATED INSTRUMENTATION
234
INSTRUMENTATION
240
SYNTHESIS
238
250
GATE-LEVEL DESIGN
INSTRUMENTATION DATA
260
GATE-LEVEL SIMULATION
RTL GATE-LEVEL MIXED MODE SIMULATION

FIG. 3

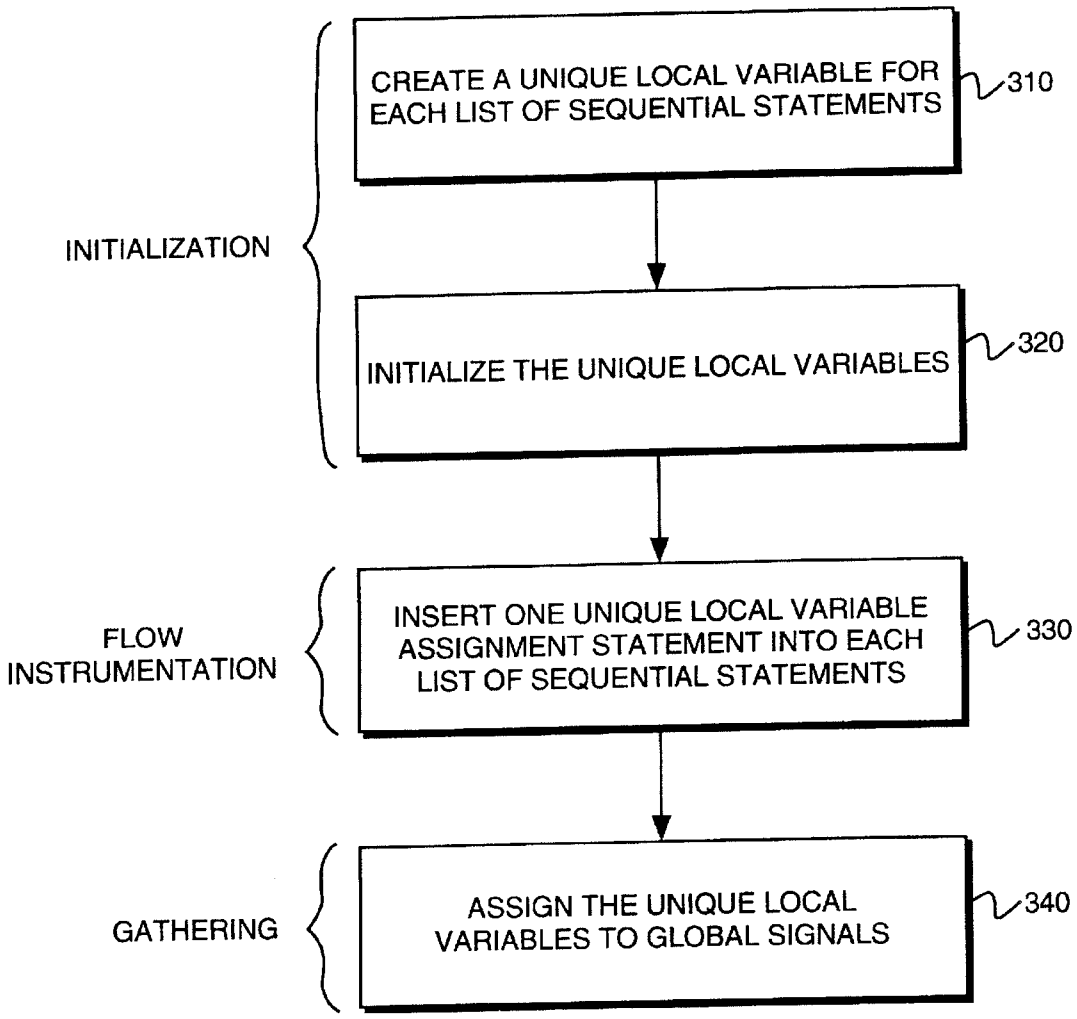

CREATE A UNIQUE LOCAL VARIABLE FOR EACH LIST OF SEQUENTIAL STATEMENTS
310
INITIALIZATION
INITIALIZE THE UNIQUE LOCAL VARIABLES
320
INSERT ONE UNIQUE LOCAL VARIABLE ASSIGNMENT STATEMENT INTO EACH LIST OF SEQUENTIAL STATEMENTS
330
FLOW INSTRUMENTATION
ASSIGN THE UNIQUE LOCAL VARIABLES TO GLOBAL SIGNALS
340
GATHERING

FIG. 4

400

```
ENTITY ALOOP IS

PORT(
        A : IN BIT_ VECTOR ( 0 TO 1 ) ;
        RESET : IN BOOLEAN;
        STATUS : OUT BOOLEAN ) ;

END ENTITY ALOOP ;


ARCHITECTURE RTL OF ALOOP IS

BEGIN

PROCESS ( A, RESET )

VARIABLE ZEROS, ONES : INTEGER ;

BEGIN
410 ---> IF ( RESET )                                   -- STATEMENT # 1
        THEN
420 --->    STATUS <= 0 ;                               -- STATEMENT # 2
        ELSE
430 --->    ZEROS := 0;                                 -- STATEMENT # 3
440 --->    ONES := 0;                                  -- STATEMENT # 4
450 --->    FOR I IN 0 TO 1 LOOP                        -- STATEMENT # 5
460 --->        IF A ( I ) = '0'                        -- STATEMENT # 6
                THEN
470 --->            ZEROS := ZEROS + 1 ;                -- STATEMENT # 7
                ELSE
480 --->            ONES := ONES + 1 ;                  -- STATEMENT # 8
                END IF ;

            END LOOP ;
490 --->    STATUS <= ( ZEROS > ONES ) ;                -- STATEMENT # 9
        END IF ;

END PROCESS ;
END ARCHITECTURE ;
```

FIG. 5

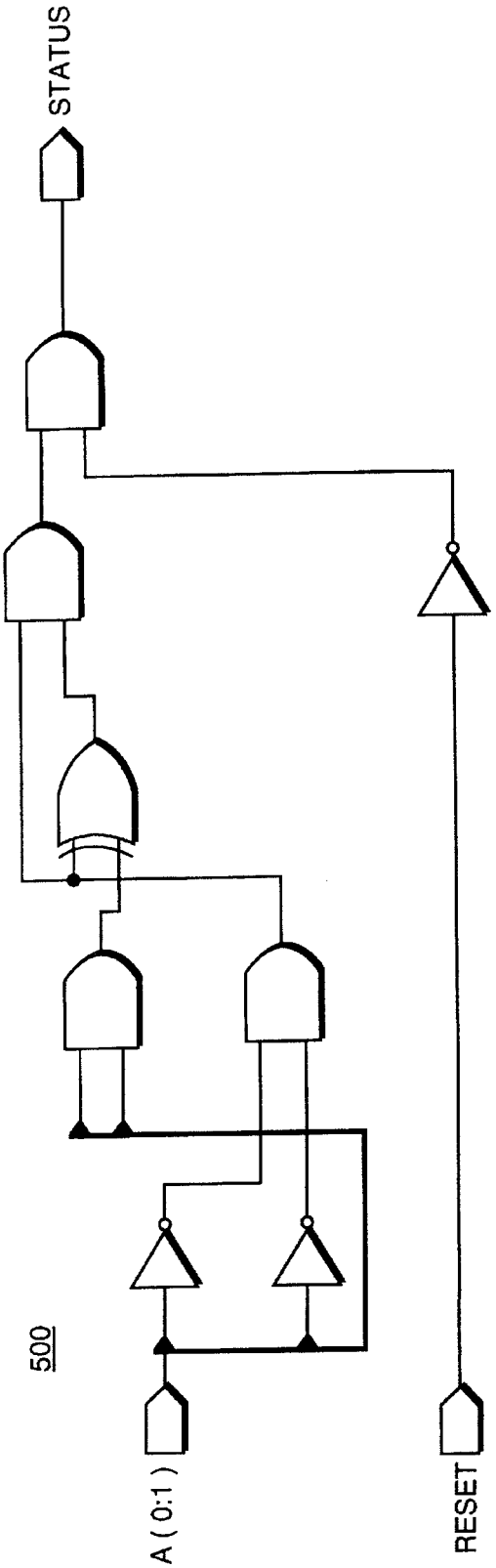
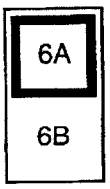


FIG. 6A



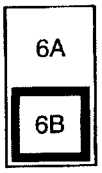
600

```
ENTITY ALOOP IS
PORT(
A : IN BIT_VECTOR (0 TO 1) ;
RESET : IN BOOLEAN ;
STATUS : OUT BOOLEAN ;
SIG_ TRACE1, SIG_ TRACE2, SIG_ TRACE3, SIG_ TRACE4, SIG_TRACE5,        } ~610
SIG_TRACE6 : OUT BIT
);
END ENTITY ALOOP ;

ARCHITECTURE RTL OF ALOOP IS
BEGIN
PROCESS (A, RESET)
VARIABLE TRACE1, TRACE2, TRACE3, TRACE4, TRACE5, TRACE6 : BIT ;  } ~612
VARIABLE ZEROS, ONES : INTEGER ;

BEGIN
TRACE1 :='0' ; TRACE2 :='0' ;
TRACE3 :='0' ; TRACE4 :='0' ;  } ~620
TRACE5 :='0' ; TRACE6 :='0' ;
```

FIG. 6B



600

```
630 ──► TRACE1 := '1' ;                    -- INSTRUMENT STATEMENT #1
        IF (RESET)                         -- STATEMENT #1
        THEN
632 ──►   TRACE2 := '1' ;                  -- INSTRUMENT STATEMENT #2
          STATUS <= FALSE ;                -- STATEMENT #2
        ELSE
634 ──►   TRACE3 := '1' ;                  -- INSTRUMENT STATEMENT #3, #4, #5, #9
          ZEROS := 0 ;                     -- STATEMENT #3
          ONES := 0 ;                      -- STATEMENT #4
          FOR I IN 0 TO 1 LOOP             -- STATEMENT #5
636 ──►     TRACE4 := '1' ;                -- INSTRUMENT STATEMENT #6
            IF A (I) ='0' ;                -- STATEMENT #6
            THEN
638 ──►       TRACE5 := '1' ;              -- INSTRUMENT STATEMENT #7
              ZEROS := ZEROS +1 ;          -- STATEMENT #7
            ELSE
640 ──►       TRACE6 := '1' ;              -- INSTRUMENT STATEMENT #8
              ONES := ONES +1;             -- STATEMENT  #8
            END IF;
          END LOOP;

642 ──► STATUS <= (ZEROS > ONES) ;         -- STATEMENT #9


        END IF ;
        SIG_ TRACE1 <= TRACE1 ; SIG_ TRACE2 <= TRACE2 ;
        SIG_ TRACE3 <= TRACE3 ; SIG_ TRACE4 <= TRACE4 ;   } ~ 650
        SIG_ TRACE5 <= TRACE5 ; SIG_ TRACE6 <= TRACE6 ;
        END PROCESS ;

        END ARCHITECTURE ;
```

FIG. 7

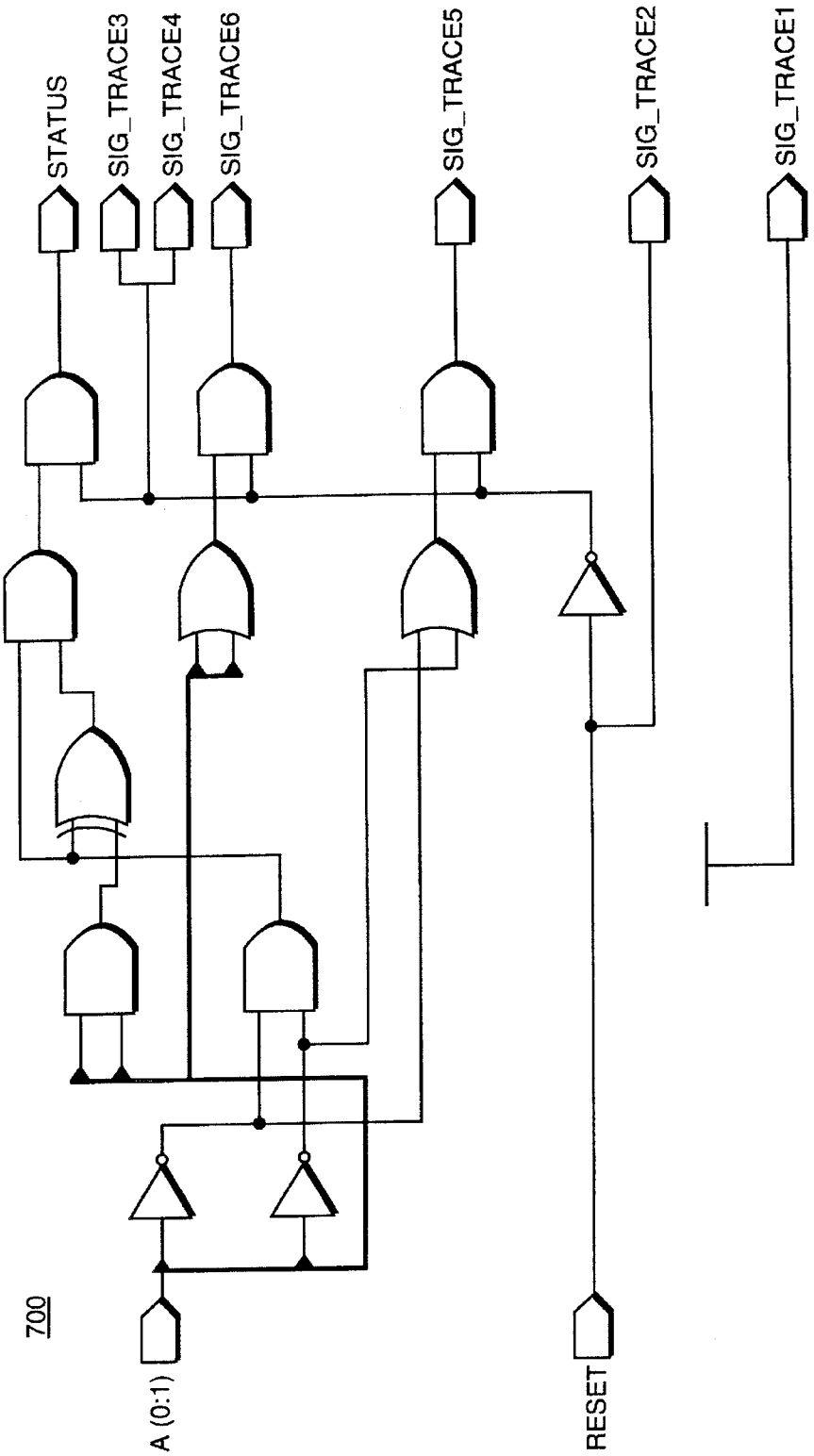

STATUS
SIG_TRACE3
SIG_TRACE4
SIG_TRACE6
SIG_TRACE5
SIG_TRACE2
SIG_TRACE1
700
A (0:1)
RESET

FIG. 8

800

```
MODULE SAMPLE (RESET, D, CLK, Q) ;

INPUT RESET ;
INPUT D ;
INPUT CLK ;
REG Q ;
OUTPUT Q ;


ALWAYS @ (CLK OR RESET OR D)
BEGIN
        IF (RESET==1)
            Q <= 0 ;
        ELSE
            IF (CLK==1)
                Q <= D ;

END


ENDMODULE
```

FIG. 9

900

```
MODULE SAMPLE
(RESET, D, CLK, Q, SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_ TRACE4 ) ;

INPUT RESET ;
INPUT D ;
INPUT CLK ;
REG Q ;
OUTPUT Q ;

REG SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_TRACE4 ;
OUTPUT SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_TRACE4 ;

INTEGER TRACE1, TRACE2, TRACE3, TRACE4 ;

ALWAYS @ (CLK OR RESET OR D)
BEGIN
        TRACE1 = 0 ; TRACE2 = 0 ; TRACE3 = 0 ; TRACE4 = 0 ;
        TRACE1 = 1 ;
        IF (RESET==1)
        BEGIN
                TRACE2 = 1 ;
                Q <= 0 ;
        END
        ELSE
        BEGIN
                TRACE3 = 1 ;
                IF (CLK== 1)
                BEGIN
                        TRACE4 = 1 ;
                          Q <= D ;
                END
        END

SIG_TRACE1 = TRACE1 ;
SIG_TRACE2 = TRACE2 ;
SIG_TRACE3 = TRACE3 ;
SIG_TRACE4 = TRACE4 ;

END

ENDMODULE
```

FIG. 10

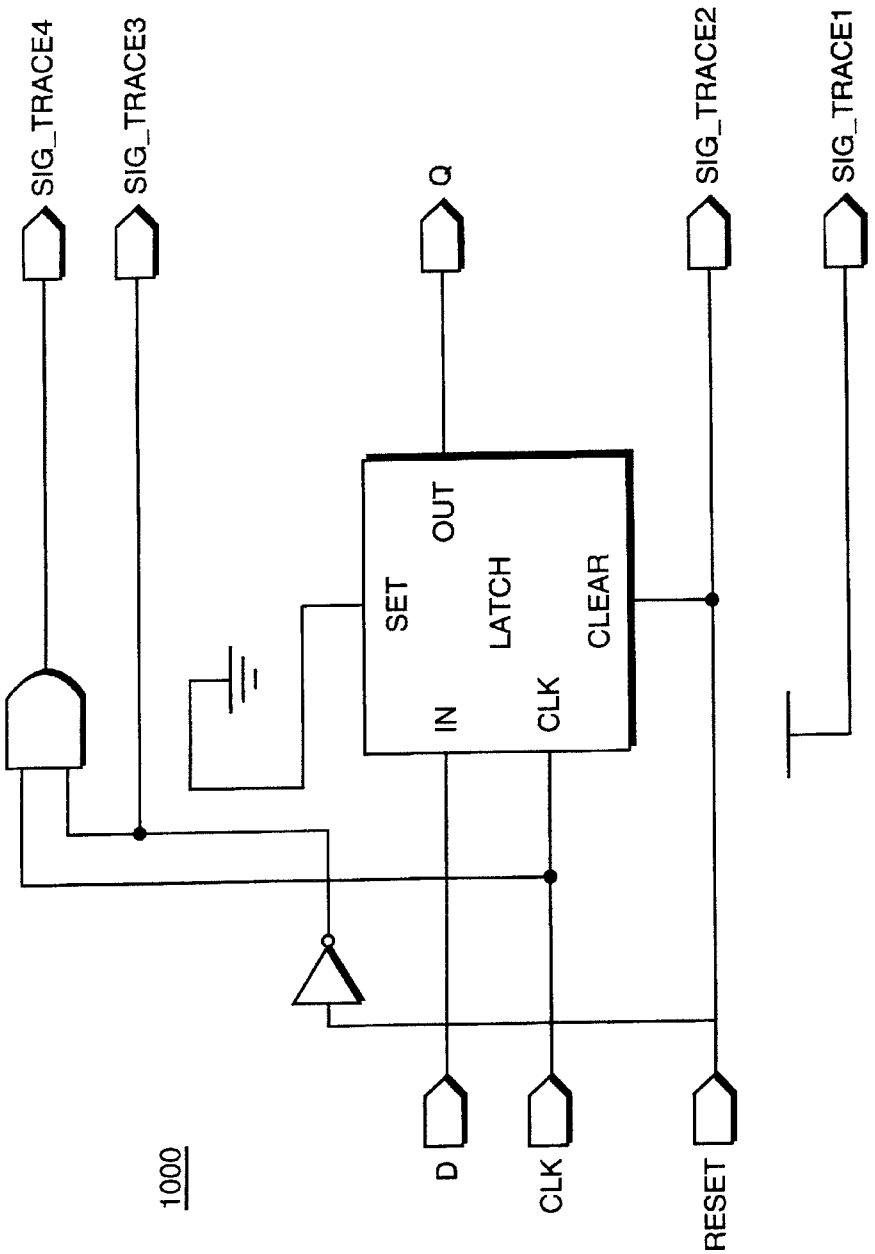

SIG_TRACE4
SIG_TRACE3
Q
SIG_TRACE2
SIG_TRACE1
SET
OUT
LATCH
CLEAR
IN
CLK
1000
D
CLK
RESET

FIG. 11

1100

```
PROCESS (CLK, D, RESET)
BEGIN
        IF (RESET = '1') THEN
                Q <= '0' ;

        ELSIF (CLK'EVENT AND CLK = '1') THEN
                Q <= D ;

        END IF;

END PROCESS
```

FIG. 12

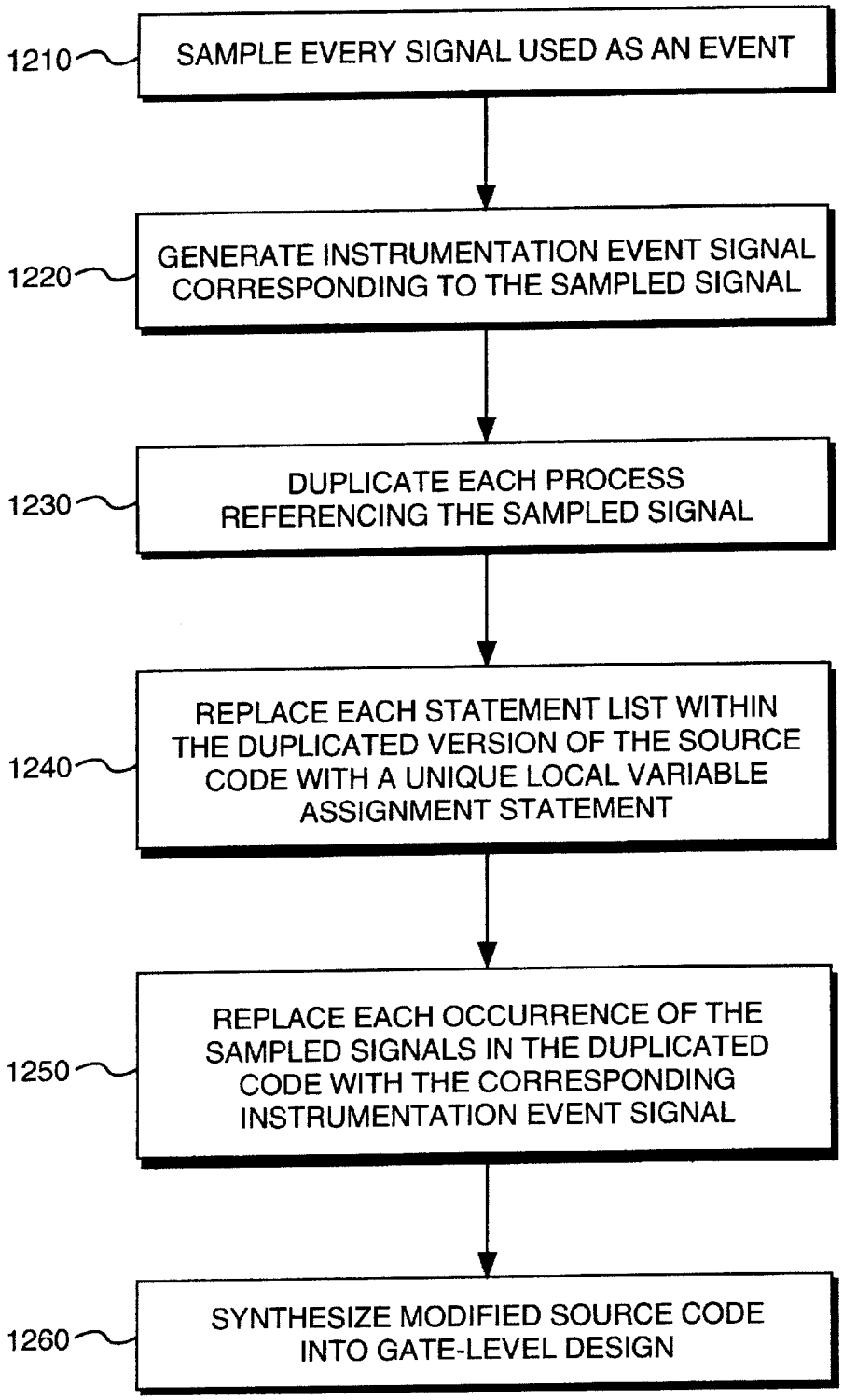

1210
SAMPLE EVERY SIGNAL USED AS AN EVENT
1220
GENERATE INSTRUMENTATION EVENT SIGNAL CORRESPONDING TO THE SAMPLED SIGNAL
1230
DUPLICATE EACH PROCESS REFERENCING THE SAMPLED SIGNAL
1240
REPLACE EACH STATEMENT LIST WITHIN THE DUPLICATED VERSION OF THE SOURCE CODE WITH A UNIQUE LOCAL VARIABLE ASSIGNMENT STATEMENT
1250
REPLACE EACH OCCURRENCE OF THE SAMPLED SIGNALS IN THE DUPLICATED CODE WITH THE CORRESPONDING INSTRUMENTATION EVENT SIGNAL
1260
SYNTHESIZE MODIFIED SOURCE CODE INTO GATE-LEVEL DESIGN

FIG. 13

1300

```
PROCESS (FAST_CLK)
BEGIN
        IF (FAST_CLK'EVENT AND FAST_CLK = '1')
        THEN
                SAMPLED_CLK <= CLK ;
        END IF
END PROCESS ;

CLK_EVENT <= SAMPLED_CLK / = CLK ;
CLK_STABLE <= SAMPLED_CLK = CLK ;
CLK_LASTVALUE <= SAMPLED_CLK ;
```

1310

```
PROCESS (CLK, D, RESET, CLK_ EVENT)
        VARIABLE TRACE1, TRACE2 : BIT ;
BEGIN
        TRACE1 := '0' ; TRACE2 := '0' ;
        IF (RESET = '1') THEN
                TRACE1 := '1' ;
        ELSIF (CLK_EVENT AND CLK = '1') THEN
                TRACE2 := '1' ;
        END IF;
SIG_TRACE1 <= TRACE1 ;  SIG_TRACE2 <= TRACE2 ;
END PROCESS ;
```

1320

```
PROCESS (CLK, D, RESET)
BEGIN
        IF (RESET = '1') THEN
                Q <= '0' ;
        ELSIF (CLK'EVENT AND CLK = '1') THEN
                Q <= D ;
        END IF ;
END PROCESS
```

1330

FIG. 14

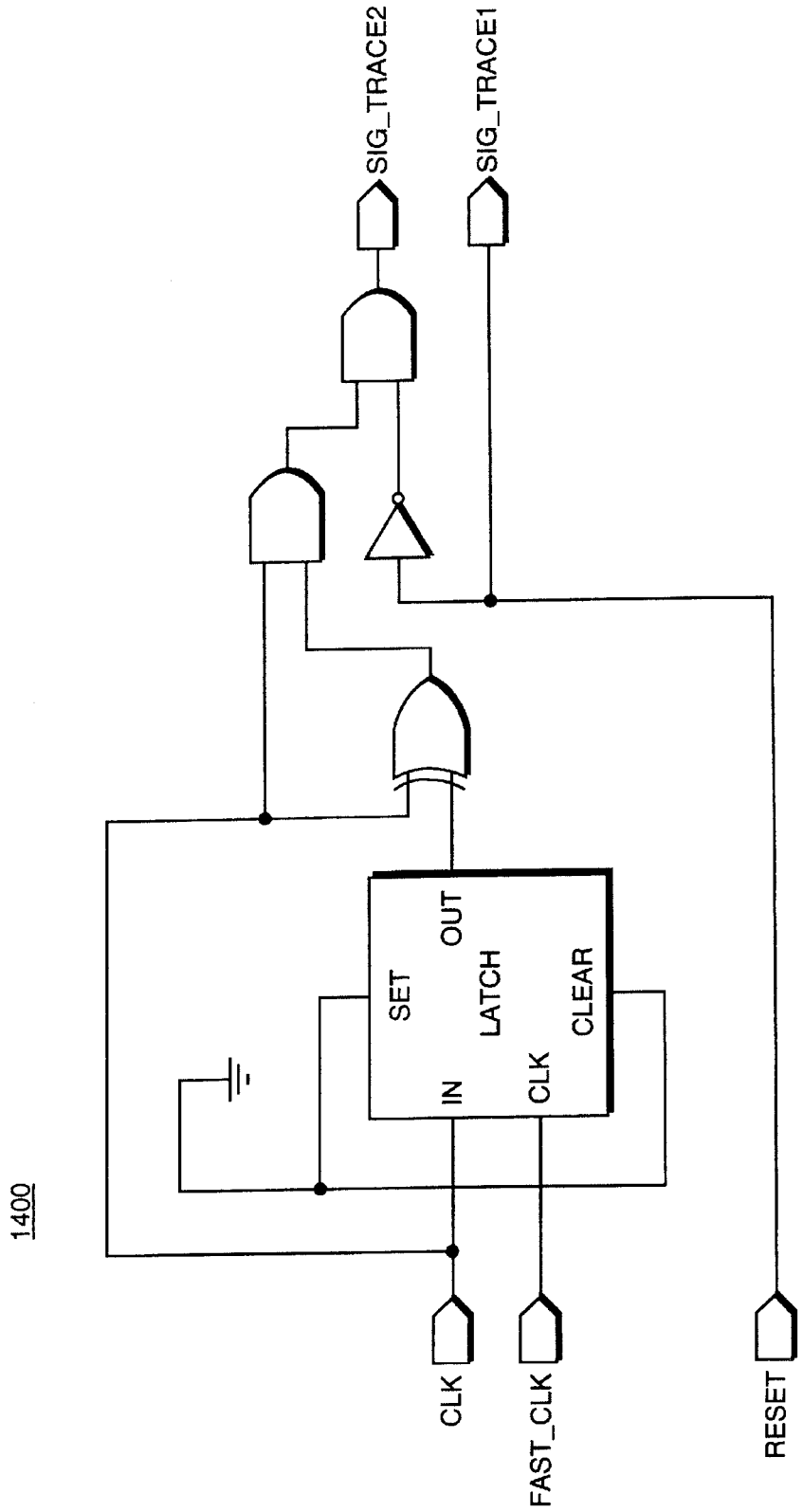

SIG_TRACE2
SIG_TRACE1
SET
OUT
LATCH
CLEAR
IN
CLK
1400
CLK
FAST_CLK
RESET

FIG. 15

1500

```
ALWAYS @ (POSEDGE CLK OR NEGEDGE RESET)
BEGIN
        IF (RESET == 0)
                Q <= 0 ;
        ELSE
                Q <= D ;
END
```

FIG. 16

1600

```
ALWAYS @ (POSEDGE FAST_CLK)
BEGIN
        SAMPLED_CLK <= CLK
        SAMPLED_RESET <= RESET ;
END
ASSIGN CLK_EDGE = SAMPLED_CLK ^ CLK ;
ASSIGN RESET_ EDGE = SAMPLED_ RESET ^ RESET;

INTEGER TRACE1, TRACE2;
REG [1:0] SIG_ TRACE ;
ALWAYS @ (CLK_EDGE OR RESET_EDGE OR CLK OR RESET)
BEGIN
        TRACE1 = 0 ; TRACE2 = 0 ;
        IF ((CLK_EDGE == 1) && (CLK == 1)II(RESET_EDGE == 1) && (RESET == 0))
                IF (RESET == 0)
                        TRACE1 =1;
                ELSE
                        TRACE2 =1;
                SIG_TRACE[0] = TRACE1 ;
                SIG_TRACE[1] = TRACE2 ;
END

ALWAYS @ (POSEDGE CLK OR NEGEDGE RESET)
BEGIN


        IF (RESET == 0)
            Q <=0 ;
        ELSE
            Q <= D ;
END
```

FIG. 17

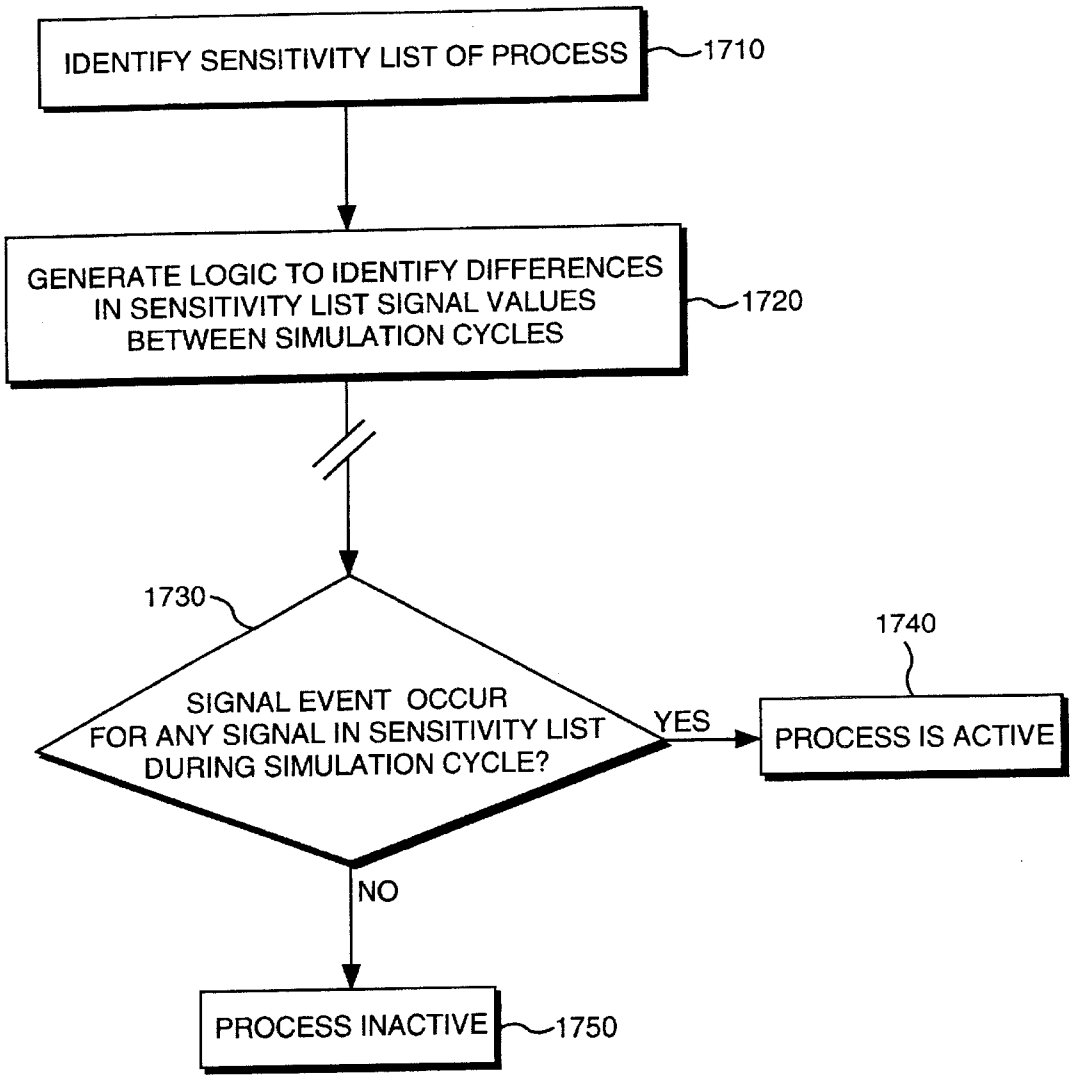

IDENTIFY SENSITIVITY LIST OF PROCESS
1710
GENERATE LOGIC TO IDENTIFY DIFFERENCES IN SENSITIVITY LIST SIGNAL VALUES BETWEEN SIMULATION CYCLES
1720
1730
SIGNAL EVENT OCCUR FOR ANY SIGNAL IN SENSITIVITY LIST DURING SIMULATION CYCLE?
YES
1740
PROCESS IS ACTIVE
NO
PROCESS INACTIVE
1750

FIG. 18

```
P1: PROCESS (A,B,C)

PROCESS (FAST_CLK)
 BEGIN
          IF (FAST_ CLK'EVENT AND FAST_ CLK='1')
           THEN
                    SAMPLED_A <=A ;
                    SAMPLED_B <=B ;
                    SAMPLED_C <=C ;
          END IF
          END PROCESS;

P1_ ACTIVE <= (SAMPLED_A /= A)
        OR (SAMPLED_B /= B)
        OR (SAMPLED_C /= C);
```

1810

1820

FIG. 19

```
CASE OPCODE IS
    WHEN "00" => TRACE1 := 1;
1910            STATE := 1;
    WHEN "01" => TRACE2 := 1;
            STATE:= 2 ;
    WHEN "10" => TRACE3 := 1;
            STATE := 2 ;
    WHEN "11" => TRACE4 := 1;
            STATE := 1 ;
END CASE ;
```

FIG. 20

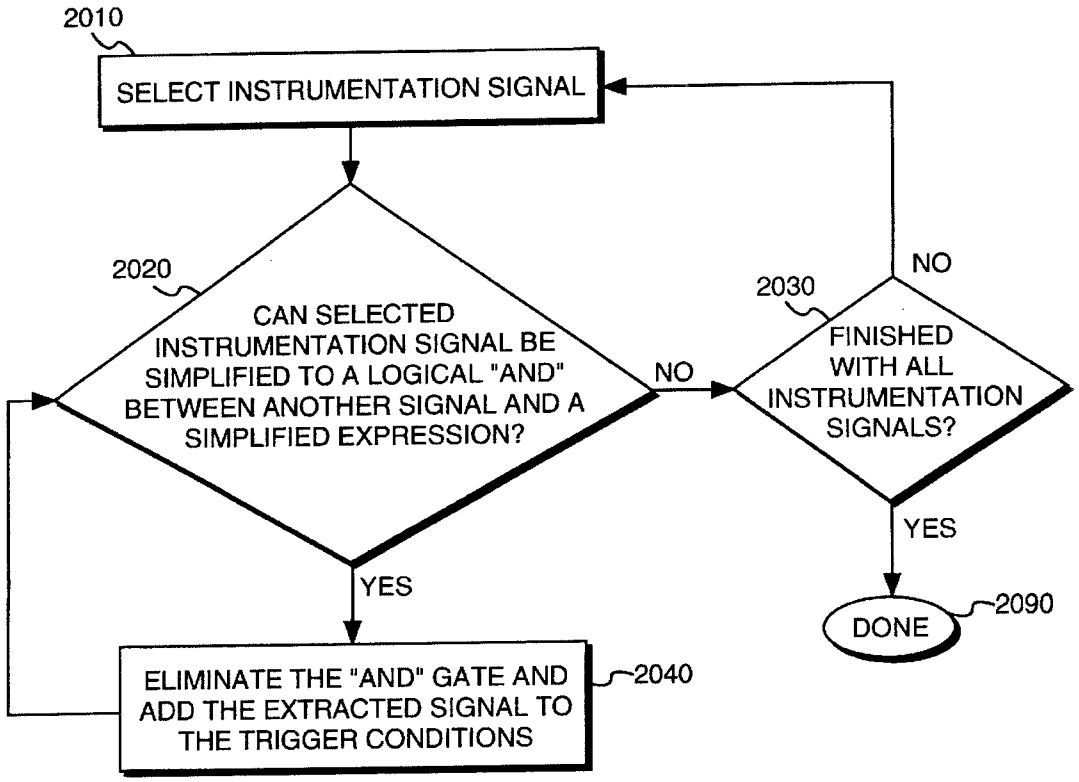


FIG. 21

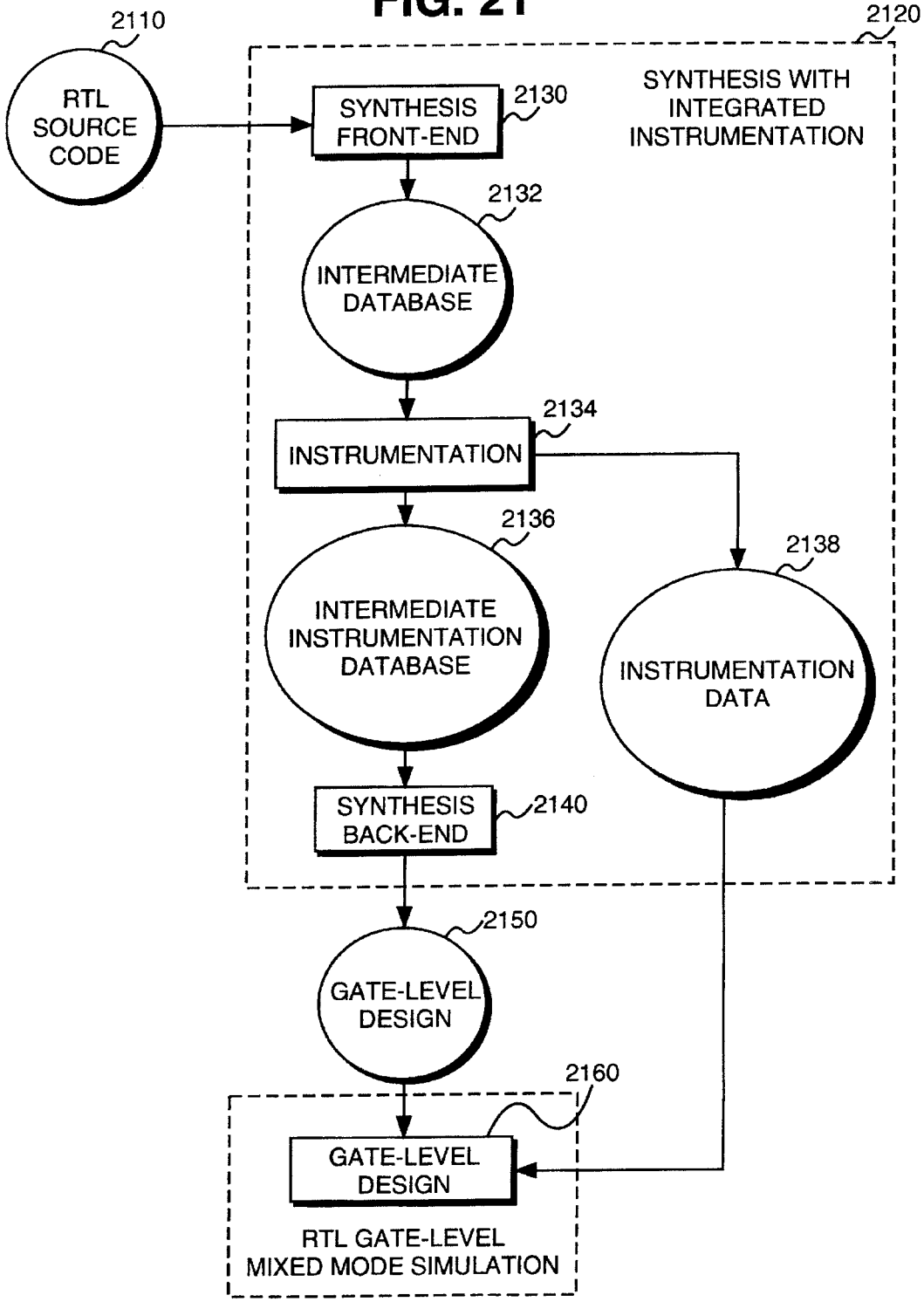
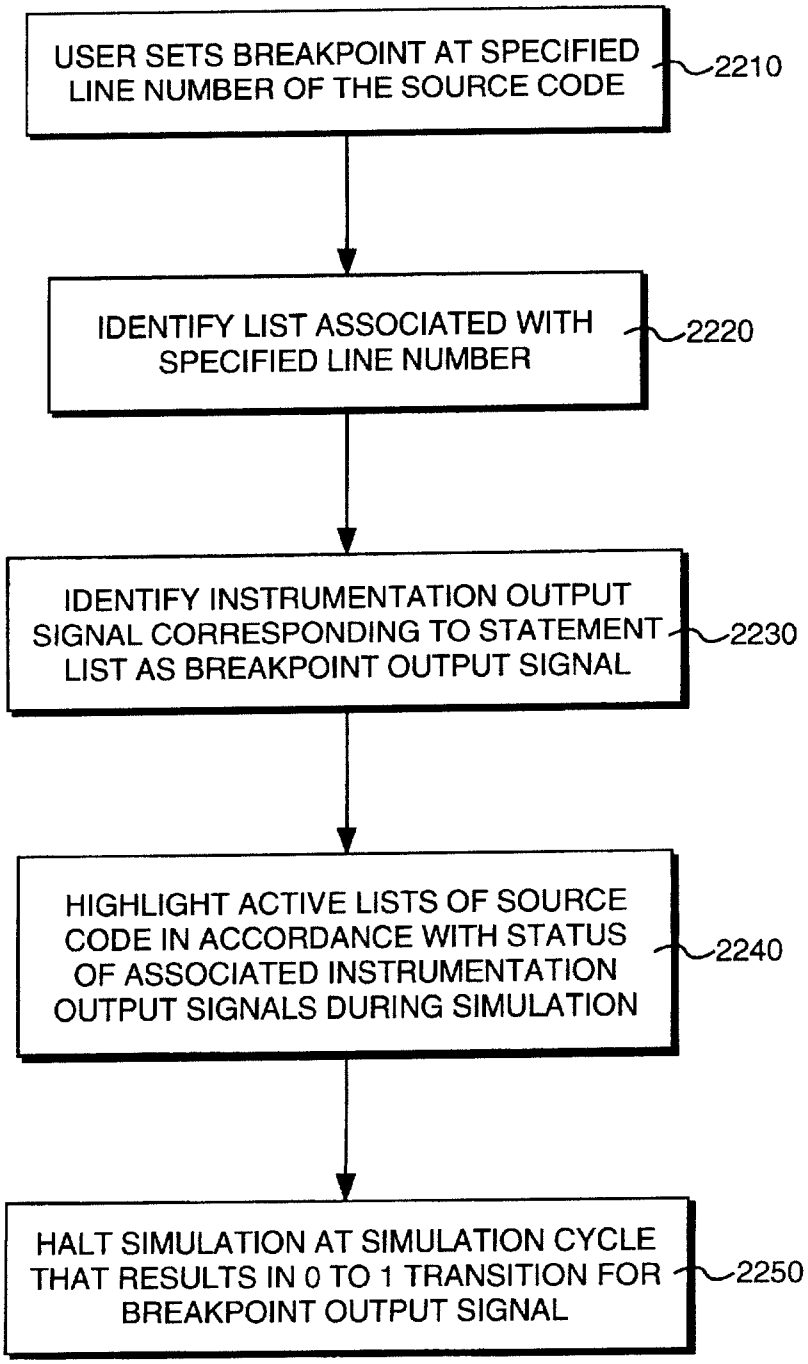

2110
RTL SOURCE CODE
2120
SYNTHESIS WITH INTEGRATED INSTRUMENTATION
SYNTHESIS FRONT-END
2130
2132
INTERMEDIATE DATABASE
2134
INSTRUMENTATION
2136
INTERMEDIATE INSTRUMENTATION DATABASE
2138
INSTRUMENTATION DATA
SYNTHESIS BACK-END
2140
2150
GATE-LEVEL DESIGN
2160
GATE-LEVEL DESIGN
RTL GATE-LEVEL MIXED MODE SIMULATION

FIG. 22


USER SETS BREAKPOINT AT SPECIFIED LINE NUMBER OF THE SOURCE CODE
2210
IDENTIFY LIST ASSOCIATED WITH SPECIFIED LINE NUMBER
2220
IDENTIFY INSTRUMENTATION OUTPUT SIGNAL CORRESPONDING TO STATEMENT LIST AS BREAKPOINT OUTPUT SIGNAL
2230
HIGHLIGHT ACTIVE LISTS OF SOURCE CODE IN ACCORDANCE WITH STATUS OF ASSOCIATED INSTRUMENTATION OUTPUT SIGNALS DURING SIMULATION
2240
HALT SIMULATION AT SIMULATION CYCLE THAT RESULTS IN 0 TO 1 TRANSITION FOR BREAKPOINT OUTPUT SIGNAL
2250

METHOD AND APPARATUS FOR GATE-LEVEL SIMULATION OF SYNTHESIZED REGISTER TRANSFER LEVEL DESIGNS WITH SOURCE-LEVEL DEBUGGING

This is a continuation in part of application Ser. No. 09/122,493, filed Jul. 4, 1998.

FIELD OF THE INVENTION

This invention relates to the fields of simulation and prototyping hen designing integrated circuits. In particular, this invention is drawn to debugging synthesizable code at the register transfer level during gate-level simulation.

BACKGROUND OF THE INVENTION

Integrated circuit designers have adopted the use of high-level hardware description languages due in part to the size and complexity of modem integrated circuits. One such description language is Very High Speed Integrated Circuit (VHSIC) Description Language, or VHDL. Further information regarding VHDL may be found in the IEEE Standard VHDL Language Reference Manual (IEEE 1076–1987, IEEE 1076–1993). Another such description language is Verilog. These high level description languages are typically generically referred to as hardware description languages (HDLs).

Synthesis is the process of generating a gate-level netlist from the high level description languages. Presently, synthesis tools recognize a subset of the high-level description language source code referred to as Register Transfer Level (RTL) source code. Further information regarding RTL source code may be found in the IEEE 1076.6/D1.10 Draft Standard for VHDL Register Transfer Level Synthesis (1997).

The RTL source code can be synthesized into a gate-level netlist. The gate-level netlist can be verified using gate-level simulation. The gate-level simulation can be performed using a software gate-level simulator. alternatively, the gate-level simulation may be performed by converting the ate-level netlist into a format suitable for programming an emulator, a hardware accelerator, or a rapid-prototyping system so that the digital circuit description can take an actual operating hardware form.

Debugging environments for high-level hardware description languages frequently include a number of functionalities for analyzing and verifying the design when performing simulation. For example, a designer can typically navigate the design hierarchy, view the RTh source code, and set breakpoints on a statement of RTL source code to stop the simulation. Statements are usually identified by their line number in the RTL source code. In addition, the debugging environment often supports viewing and tracing variables and signal values. The RTL simulation environment typically offers such RTL debugging functionalities.

RTL simulation is typically performed by using software RTL simulators which provide good flexibility. However, for complex designs, a very large number of test vectors may need to be applied in order to adequately verify the design. This can take a considerable amount of time using software RTL simulation as contrasted with hardware acceleration or emulation starting from a gate-level netlist representation (i.e., "gate-level hardware acceleration," or "gate level emulation"). Furthermore, it may be useful to perform in-situ verification, which consists of validating the design under test by connecting the emulator or hardware accelerator to the target system environment (where the design is to be inserted after the design is completed).

One disadvantage with gate-level simulation, however, is that most of the high-level information from the RTL source code is lost. Without the high-level information, many of the debugging functionalities are unavailable.

For example, the designer typically cannot set a breakpoint from the source code during gate-level simulation. Although signals can be analyzed during gate-level simulation, mapping signal values to particular source code lines can be difficult, if not impossible. If the source code is translated into a combinatorial logic netlist, for example, the designer cannot "step" through the source code to trace variable values. Instead, the designer is limited to analyzing the input vector and resulting output vector values. Although the signals at the inputs and outputs of the various gates may be traced or modified, these values are determined concurrently in a combinatorial network and thus such analysis is not readily mappable to the RTL source code.

A typical design flow will include creating a design at the RTL level, then synthesizing it into a gate-level netlist. Although simulation of this netlist can be performed at greater speeds using emulators or hardware accelerators, the ability to debug the design at the gate level is severely limited in comparison with software RTL simulation.

SUMMARY OF THE INVENTION

Methods of instrumenting synthesizable register transfer level (RTL) source code to enable debugging support akin to high-level language programming environments for gate-level simulation are provided.

One method of facilitating gate-level simulation includes the step of generating cross-reference instrumentation data including instrumentation logic indicative of the execution status of at least one synthesizable statement within the RTL source code. A gate-level netlist is synthesized from the RTL source code. Evaluation of the instrumentation logic during simulation of the gate-level netlist enables RTL debugging by indicating the execution status of the cross-referenced synthesizable statement in the RTh source code.

In one embodiment, the gate-level netlist is modified to provide instrumentation signals implementing the instrumentation logic and corresponding to synthesizable statements within the RTL source code. In various embodiments, this may be accomplished by modifying the RTL source code or by generating the modified gate-level netlist during synthesis as if the source code had been modified.

Alternatively, the gate-level netlist is not modified but the instrumentation signals implementing the instrumentation logic are contained in a cross-reference instrumentation database. In either case, the instrumentation signals indicate the execution status of the corresponding cross-referenced synthesizable statement. The instrumentation signals can be used to facilitate source code analysis, breakpoint debugging, and visual tracing of the source code execution path during gate-level simulation.

For example, a breakpoint can be set at a selected statement of the source code. A simulation breakpoint is set so that the simulation is halted at a simulation cycle where the value of the instrumentation signals indicate that the statement has become active .

With respect to visually tracing the source code during execution, the instrumentation logic is evaluated during gate-level simulation to determine a list of at least one active statement. The active statement is displayed as a highlighted statement.

With respect to source code analysis, cross-reference instrumentation data including the instrumentation signals

can be used to count the number of times a corresponding statement is executed in the source code. For example, an execution count of the cross-referenced synthesizable statement is incremented when evaluation of the corresponding instrumentation logic indicates that the cross-referenced synthesizable statement is active.

Other features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description that follows below.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

FIG. **1** illustrates the process of synthesizing RTL source code into a gate-level design.

FIG. **2** illustrates one embodiment of a modified process for generating a gate-level design.

FIG. **3** illustrates one embodiment of a method for instrumenting level-sensitive RTL source code.

FIG. **4** illustrates VHDL source code.

FIG. **5** illustrates the gate-level design synthesized from the RTL source code of FIG. **4**.

FIG. **6** illustrates the VHDL source code of FIG. **4** modified in accordance with the method of FIG. **3**.

FIG. **7** illustrates one embodiment of the gate-level logic synthesized from the modified RTL source code.

FIG. **8** illustrates sample Verilog source code before instrumentation.

FIG. **9** illustrates the Verilog source of FIG. **8** instrumented in accordance with the method of FIG. **3**.

FIG. **10** illustrates the gate-level logic synthesized from the instrumented Verilog source code of FIG. **9**.

FIG. **11** illustrates VHDL source code for a D flip-flop with asynchronous reset.

FIG. **12** illustrates one method of instrumenting event-sensitive RTL source code.

FIG. **13** illustrates the source code of FIG. **11** modified in accordance with the instrumentation process of FIG. **12**.

FIG. **14** illustrates the gate-level logic synthesized for the instrumented source code of FIG. **13**.

FIG. **15** illustrates Verilog source code for a D flip-flop with asynchronous reset.

FIG. **16** illustrates the Verilog source code of FIG. **15** after instrumentation in accordance with the method of FIG. **12**.

FIG. **17** illustrates a method of instrumenting process activation.

FIG. **18** illustrates source code modified in accordance with the method of FIG. **17**.

FIG. **19** illustrates an instrumented "case" statement.

FIG. **20** illustrates a process for decreasing the logic needed to instrument the source code.

FIG. **21** illustrates incorporating instrumentation within the synthesis process.

FIG. **22** illustrates a method of setting a breakpoint in RTL source code for use during gate-level simulation.

DETAILED DESCRIPTION

FIG. **1** illustrates a typical RTL source code synthesis process. HDL code including synthesizable RTL source code (**110**) serves as input to a synthesis process **120**. In one embodiment, the RTL source code **110** is synthesized in step **140** to produce a gate-level design **150**. The gatelevel design can be used for gate-level simulation as illustrated in step **160**.

Typically the gate-level design comprises a hierarchical or flattened gate level netlist representing the circuit to be simulated. The various signals in a design are referred to as nets. A hierarchical netlist is made of a list of blocks, whereas a flattened netlist comprises only one block. A block contains components and a description of their interconnection using nets. Components can be reduced to combinatorial or sequential logic gates, or they may be hierarchical blocks of lower level.

For example, the component may be a primitive gate denoting a single combinatorial logic function (e.g., AND, NAND, NOR, OR, XOR, NXOR, etc.) or a single storage element such as a flip-flop or latch for sequential logic. One example of a set of primitive gates is found in the generic library GTECH available from Synopsys, Inc. of Mountain View, Calif.

Alternatively the component may be an application specific integrated circuit (ASIC) library cell which can be represented by a set of primitive gates. One example of an ASIC library is the LCA300K ASIC library developed by LSI Logic, Inc., Milpitas, Calif.

A component may also be a programmable primitive that represents a set of logic functions and storage. One example of a programmable primitive is the configurable logic block (CLB) as described in The Programmable Gate Array Handbook, Xilinx Inc., San Jose, 1993.

Another example of a component is a macro block denoting a complex logic function such as memories, counters, shifters, adders, multipliers, etc. Each of these can be further reduced to primitive gates forming combinatorial or sequential logic.

Three major categories of tools are available to the designer to simulate and test the design. Software RTL simulators (such as ModelSim™ from Model Technology, Inc.) typically offer a high-level of abstraction for their debugging environment, but have limited performance in terms of speed and no in-situ capacity. Software gate-level simulators (such as QuickSim™ from Mentor Graphics Corporation) typically offer limited level of abstraction and speed as well as no in-situ capacity. Hardware gatelevel simulators (such as Cobalt™ and System Realizer™ from Quickturn Inc., Avatar™ from Ikos, and fast-prototyping systems usually built from FPGAs) typically offer very good performance in terms of speed and in-situ capacity, but a limited debugging environment.

When testing the design described by the HDL source code a designer may choose to simulate and validate the design at the RTL source code level (i.e., RTL simulation). RTL simulation typically permits the designer to set breakpoints in the source code, navigate the design hierarchy, view variables and signals and trace the value of these variables and signals.

When testing complex designs, millions or billions of test vectors may need to be applied in order to adequately test the design. Hardware accelerators or emulators can be used with the gate-level design to test the design at a much greater speed than what is typically possible through software simulation (i.e. either software RTL simulation or software gate-level simulation). Unfortunately, the gate-level design generated in step 150 typically includes none of the high-level information available in the RTL source code **110**. As a result, features available during RTL simulation such as

setting breakpoints or analyzing the source code coverage are not available during gate-level simulation.

Instrumentation is the process of preserving high-level information through the synthesis process. Instrumentation permits simulation of a gatelevel netlist at the level of abstraction of RTL simulation by preserving some of the information available at the source code level through the synthesis process.

FIG. **2** illustrates one embodiment of the instrumentation process in which instrumentation is integrated with the synthesis process. RTL source code **210** is provided to the synthesis process **220**. The synthesis process **120** of FIG. **1** has been modified to include an instrumentation step **234**. After instrumentation the instrumented code is then synthesized in step **240** as the original RTL source code was in step **140** of FIG. **1**.

In one embodiment, instrumentation results in generating a modified gate-level design to permit reconstitution of the flow of execution of the original RTL source code during gate-level simulation. Generally instrumentation logic is created for a synthesizable statement in the RTL source code either by modifying the RTL source code or by analyzing the RTL source code during the synthesis process. The instrumentation logic provides an output signal indicative of whether the corresponding synthesizable statement is active. A gate-level design including the instrumentation output signal is then synthesized. Referring to FIG. **2**, the resulting gate-level design **250** contains additional logic to create the additional instrumentation output signals referenced in instrumentation data **238**.

In an alternative embodiment, the RTL source code is analyzed to generate a cross-reference database as instrumentation data **238** without modifying the gate-level design. The cross-reference database indicates the combination of already existing signals in the form of instrumentation logic that can be evaluated during simulation to determine whether a particular line of the RTL source code is active. The cross-reference database contains a cross-reference between these instrumentation logic output signals and the position of the corresponding statement in the source code. The instrumentation data **238** is likely to contain considerably more complex logic to evaluate during simulation when the approach of not modifying the gate-level design (i.e., "pure" cross-reference database) is taken.

The two approaches have tradeoffs. The gate-level design modification technique does not require special knowledge of the target simulation environment. Moreover, the gate-level design modification technique significantly reduces or eliminates the complexity of the logic to be evaluated during simulation to the extent that emulator or accelerator hardware triggering circuitry can be used to take an action when the corresponding statement is executed.

For example, the hardware triggering circuitry may be used to halt the simulation at a particular statement or to count the number of times a particular statement is executed. The resulting gate-level design used during simulation, however, will not be the design actually used for production thus simulation may not verify accurately the behavior of the gate-level design used for production. Furthermore, simulation of modified gate-level design may require more physical resources in hardware than the original design alone if gates have been added in order to implement the instrumentation logic.

Alternatively, the pure cross-reference database technique typically results in greater complexity of instrumentation logic to evaluate during simulation, but does not otherwise affect the original gate-level design. The greater complexity, however, may prevent the use of the hardware triggering circuitry to halt the simulation or to track source code coverage. Thus the pure cross-reference database technique may result in a significantly slower simulation time. Furthermore, since the evaluation may be performed by software, direct verification of the gate-level design in the target system through in-situ verification may not be possible. The instrumentation data including the logic added for instrumentation purposes can be eliminated after testing, however, without disrupting the gate-level design.

In essence the gate-level design modification technique greatly simplifies the analysis and the instrumentation logic required for cross-referencing by modifying the gate-level design to create unique signals and therefore simpler logic to evaluate (i.e., a single signal). The resulting instrumentation logic cross-referenced in the instrumentation data **238** is easily evaluated during simulation. Various embodiments of instrumentation may combine the gate-level design modification technique or the pure cross-referencing technique in order to trade off simulation speed, density, and verification accuracy.

If the gate-level simulator, hardware accelerator, or emulator (e.g., through the use of a logic analyzer which can be external to the emulator) has the capacity to set breakpoints whenever certain signals reach a given value, then it is possible to implement breakpoints corresponding to RTL simulation breakpoints in the gate-level design. Whenever the user specifies a breakpoint in the RTL source code, the condition can be converted to a comparison with key signals in the gate-level design.

Instrumentation data **238** identifies the RTL source code statements each instrumentation output signal is associated with. Instrumentation data **238** is generated during the instrumentation process of step **234**. In one embodiment, the instrumentation data is implemented as gates that can then be simulated by the target-level simulator. By examining the state of each instrumentation output signal during gate-level simulation, the user can determine which portions of RTL source code are being simulated. This in turn permits the designer to determine RTL source code coverage. By tracking the instrumentation signal values for each cycle of execution, the designer can determine how many times each line of the RTL source code has been activated.

The instrumentation data **238** can be used during simulation to ensure every possible state transition has been tested. For example, a Finite State Machine analyzer can determine from the values of the instrumentation output signals whether every possible state transition has been tested.

The instrumentation data **238** can also be used to enhance the source code display. In one embodiment, the source code is repositioned on the display so as to indicate the execution paths that are active during a current cycle. In another embodiment, the active source code in a given cycle is highlighted to indicate that it is active. This permits the designer to visually see the process flow without having to determine the value of each signal. In one embodiment, the instrumentation data **238** is used to enhance the display of the original RTL source code rather than the source code resulting from instrumentation.

An integrated circuit design is typically built by assembling hierarchical blocks. In VHDL, a block corresponds to an entity and architecture. In Verilog, a block corresponds to a module. In both HDLs, a block typically includes a declarative portion and a statement portion. The declarative portion generally includes the list of the ports or connectors.

The statement portion describes the block's behavior and is typically where a designer needs help when debugging a design. The statement portion includes concurrent signal assignment statements and sequential statements.

Concurrent signal assignment statements assign a logic expression to a signal. The signal is typically available for viewing at all times and thus breakpoints can be set in accordance with when the signals reach a certain value.

Sequential statements assign values depending upon the execution flow of the sequence. Sequential statement analysis is typically where the designer needs the greatest aids in debugging the design.

Sequential statements are typically found in VHDL "processes" and in Verilog "always" blocks. Processes or always blocks can be built of an unlimited combination of sequential statements including loops, conditional statements, and alternatives. There are at least two classes of sequential statements: level-sensitive and event-sensitive. Level-sensitive sequential statements only depend on the value of the inputs and can be synthesized to logic networks of combinatorial gates and latches. Event-sensitive sequential statements additionally require sequential logic such as flip-flops.

In one embodiment, level-sensitive RTL source code is instrumented by creating and associating one output signal with each list of synthesizable sequential statements. A list can consist of one or more sequential statements.

In one embodiment, each statement is a list. In an alternative embodiment, each list corresponds to a branch of the RTL source code. A list corresponding to a branch typically comprises a plurality of adjacent sequential statements, but may comprise a single sequential statement. Only one output signal is needed for each list of synthesizable sequential statements in a branch rather than for every sequential statement in the source code. Examples of sequential statements that create branches in the RTL source code are conditional statements such as IF-THEN statements and SELECT-CASE statements.

FIG. **3** illustrates one method of modifying RTL source code for level-sensitive code. Generally, a unique local variable is created for each list of adjacent sequential statements in step **310**. The level sensitive code instrumentation includes the step of modifying the RTL source code to initialize each of these unique variables to zero at the beginning of the process being instrumented in step **320**. One unique variable assignment statement is inserted into each list of adjacent sequential statements corresponding to an executable branch in step **330**. The assignment statement sets the unique variable to one. At the end of the process all the unique local variables are assigned to global signals in step **340**. Steps **310** and **320** are more generically referred to as initialization. Step **330** is referred to as flow instrumentation. Step **340** is referred to as "gathering."

FIG. **4** illustrates non-instrumented VHDL source code. The VHDL source code **400** includes nine sequential statements within the process block. Eight of these nine statements are non-signal assignment sequential statements. These eight sequential statements form six statement lists or executable branches of the code. IF-THEN statement **410** comprises one list. Signal assignment statement **420** comprises a second list. Statements **430**, **440**, **450** and **490** comprise a third list because they would be executed sequentially within the same execution path. Statements **460**, **470**, and **480** form individual lists.

FIG. **5** illustrates one embodiment of the logic **500** resulting from the synthesis of the RTL source code of FIG. **4**. This figure may be used for comparison with the gate level design generated from instrumented code described below.

FIG. **6** illustrates the source code of FIG. **4** after instrumentation as described in FIG. **3**. The added statements are italicized for emphasis. For example, line **612** has been added to the source code to create six unique local variables (TRACE1 through TRACE6), one for each of the six identified lists, in accordance with step **310** of FIG. **3**.

In accordance with step **330** of FIG. **3**, a trace variable assignment statement has been added adjacent to each of the lists. Referring to FIGS. **4** and **6**, variable assignment statement **630** has been added adjacent to the first list comprising statement **410**. Variable assignment statement **632** has been added adjacent to the second list comprising statement **420**. Variable assignment statement **634** has been added adjacent to the third list comprising statements **430**, **440**, **450** and **490**. Variable assignment statement **636** has been added adjacent to the fourth list comprising statement **460**. Similarly, variable assignment statements **638** and **640** have been added adjacent to the fifth list comprising statement **470** and the sixth list comprising **480**, respectively. Each of variable assignment statements **630** through **640** assigns a unique local variable the value of one.

Code portion **620** is added to initialize the unique local variables to zero at the beginning of the process in accordance with step **320** of FIG. **3**.

Each of the local variables is assigned to a global output signal in accordance with step **340** of FIG. **3** by code portion **650**. If required by the HDL, the global signals are declared by code portion **610**. Similarly, the trace variables are declared by code portion **612**.

In one embodiment, the unique local variables can actually be a single array where each "unique variable" or trace variable corresponds to a different position in the array. Similarly, in one embodiment, the additional global signals are described by an array where each of the global signals is represented by a different index of the array.

Coding practices for VHDL generally require variables to be used within the process and a signal assignment at the end of the process to propagate the variable values at the end of the process. In one embodiment, markers such as variable assignment statements are used to track the execution paths. Markers such as variable assignment statements are not typically synthesized into logic indicating the variable values, thus the variable assignment statements are used in conjunction with signal assignment statements in order to produce signals indicating whether various portions of the synthesized code are being executed.

If permitted by the HDL, however, global signal assignments can be used in lieu of local variable assignment statements. This would simplify the process of FIG. **3** in that there would be no need to create or initialize local variables. In addition the step of assigning the local variables to global signals could be eliminated because values are assigned directly. The key is ensuring that there is a unique output signal created and associated with each list of sequential statements regardless of the coding practice used to achieve this goal.

FIG. **7** illustrates one embodiment of the logic **700** generated through instrumentation. In particular, FIG. **7** illustrates the additional gate-level logic added to generate signals SIG_TRACE1 through SIG_TRACE6 from synthesis of the modified source code.

FIG. **8** illustrates a Verilog "always" block **800**. FIG. **9** illustrates the same code after instrumentation in accordance with the process of FIG. **3**. Due to Verilog syntax

requirements, "BEGIN-END" statements were used to properly group the instrumentation variable with the other statements in each executable path.

Although the code of FIG. **8** results in a latch, application of the technique of FIG. **3** to the source code of FIG. **8** ensures that the instrumentation output signals are the result of combinatorial logic only. Thus the logic for determining which lines of code are active can be purely combinatorial even when the RTL source code is synthesized into latches.

FIG. **10** illustrates one embodiment of gate-level logic **1100** generated by synthesis of the instrumented "always" block **900** of FIG. **9**. The instrumentation signals SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, and SIG_TRACE4 are the result of combinatorial logic only.

Referring to FIG. **2**, the instrumentation data **238** can be stored in a cross-reference file. In one embodiment, the cross-reference file contains a mapping between original source code line numbers and instrumentation signals. Each time an instrumentation variable (and its associated signal) is added to the source code, all the line numbers of the statements in the list associated with the instrumentation variable are added to the file. This cross-reference file (i.e., instrumentation data **238**) can be used by the gate-level simulation environment to convert the designer's breakpoints into actual conditions on instrumentation signals.

A more sophisticated method than that illustrated in FIG. **3** is required to instrument RTL source code having references to signal events. Typically such source code is used to describe edge-sensitive devices. References to signal events typically imply flip-flops. A signal event is a signal transition. Thus any signal computed from a signal transition references a signal event.

FIG. **11** illustrates sample VHDL code **1100** with references to a signal event. VHDL code **1100** implements a D-type flip-flop with asynchronous reset. The event in this example is a transition on the clock signal (CLK) as referenced by the term "CLK'EVENT."

In accordance with VHDL specifications signals can have various attributes associated with them. A function attribute executes a named function on the associated signal to return a value. For example, when the simulator executes a statement such as CLK'EVENT, a function call is performed to check this property of the signal CLK. In particular, CLK'EVENT returns a Boolean value signifying a change in value on the signal CLK. Other classes of attributes include value attributes and range attributes.

In VHDL code **1100**, the signal CLK has a function attribute named "event" associated with it. The predicate CLK'EVENT is true if an event (i.e., signal transition) has occurred on the CLK signal. Assigning a value to a signal (i.e., a signal transaction) qualifies as an event only if the transaction results in a change in value or state for the signal. Thus the predicate CLK'EVENT is true whenever an event has occurred on the signal CLK in the most recent simulation cycle. The predicate "IF (CLK'EVENT and CLK='1')" is true on the rising edge of the signal CLK.

Depending upon the specifics of the HDL, another function such as RISING_EDGE(CLK) might be used to accomplish the same result without the use of attributes. The function RISING_EDGE(CLK) is still an event even though the term "event" does not appear in the function.

FIG. **12** illustrates a method of instrumenting source code having references to signal events. In step **1210**, every signal event is sampled using a fast clock. In other words, every signal whose state transition serves as the basis for the determination of another signal is sampled. An instrumentation signal event corresponding to the original signal event is generated in step **1220**. Any attributes of the original signal must similarly be reproduced based on the instrumentation signal if the source code uses attributes of the original signal event.

In step **1230**, every process that references a signal event is duplicated. In step **1240**, each list of sequential statements within the duplicate version of the code is replaced by a unique local variable assignment statement. In step **1250**, each time a signal event is referenced in the duplicated version of the code, it is replaced by the sampled signal event computed in step **1210**. The modified RTL source code can then be synthesized in step **1260** to generate gate-level logic including the instrumentation output signals.

FIG. **13** illustrates application of the method of FIG. **12** to the source code of FIG. **11**. In order to detect signal events properly for instrumentation, the signal events are sampled using a fast clock provided during gate-level simulation (i.e., FAST_CLK). FAST_CLK has a higher frequency than the CLK signal and thus permits detecting transition edges before signals depending upon CLK (including CLK itself) can.

The only signal event referenced in FIG. **11** is a transition in the signal CLK indicated by the term CLK'EVENT. Thus an instrumentation version of CLK'EVENT is created by sampling the signal CLK using FAST_CLK. The signal FAST_CLK has a higher frequency than the signal CLK.

Code portion **1310** samples the CLK signal on every rising edge of the signal FAST_CLK to generate a sampled version of the signal CLK named SAMPLED_CLK. The instrumentation version of CLK'EVENT is CLK_EVENT which is generated in code portion **1310** based on SAMPLED_CLK. The instrumentation signal CLX_EVENT (corresponding to CLK'EVENT) is determined by comparison of signals SAMPLED_CLK and CLK. The signal CLK_EVENT is true only when the signal SAMPLED_CLK is not the same as CLK, thus indicating a transition has occurred in the signal CLK.

Although not required for this example, code portion **1310** also illustrates the generation of instrumentation clock signal attributes based on SAMPLED_CLK. For example, the signal CLK'STABLE is the complement of CLK'EVENT. Thus code portion **1310** indicates the instrumentation version of the attribute CLK'STABLE (i.e., CLK_STABLE) computed on the instrumentation clock signal (i.e., SAMPLED_CLK). The signal CLK'LASTVALUE is a function signal attribute that returns the previous value of the signal CLK. The instrumentation version (i.e., CLK_LASTVALUE) of the attribute CLK'LASTVALUE is similarly computed on the instrumentation clock signal SAMPLED_CLK.

Although CLK_LASTVALUE is the same as the sampled clock signal, SAMPLED_CLK, code **1310** introduces the intermediate signal SAMPLED_CLK for purposes of illustrating sampling of the CLK signal. The signal CLK_LASTVALUE can be defined in lieu of SAMPLED_CLK in order to eliminate the introduction of an unnecessary intermediate signal SAMPLED_CLK and the subsequent step of assigning CLK_LASTVALUE the value of SAMPLED_CLK.

Neither CLK_LASTVALUE nor CLK_STABLE are needed in this example for code portion **1320**, however, code portion **1310** serves as an example of how to generate instrumentation versions of signal attributes typically used to describe edge-sensitive devices.

Code portion **1320** represents the instrumented duplicate of original code portion **1330**. The process of code portion

1330 references the event CLK'EVENT in the IF-ELSIF statement. In code portion 1320, all sequential statements (except the statement referencing an event) have been replaced with unique local variable assignment statements. These statements assign a local variable (i.e., TRACE1, TRACE2) the value "1." Code portion 1320 also includes statements to create and initialize these unique local variables.

In accordance with step 1240, every occurrence of a signal event is replaced with the sampled version of that event. Thus, for example, references to CLK'EVENT in code portion 1330 are replaced with references to CLK_EVENT in code portion 1320. Moreover, the process parameter list is modified to include the generated signal CLK_EVENT. FIG. 14 illustrates the gate-level logic 1400 resulting from synthesis of the code in FIG. 13.

FIG. 15 illustrates Verilog source code 1500 for a D flip-flop with asynchronous reset. FIG. 16 illustrates the code 1600 resulting from modifying source code 1500 in accordance with the method of FIG. 12.

One advantage of the instrumentation approach of FIG. 12 is that the gates generated by the synthesis tool are the same ones that would be generated if the source code had not been instrumented. The gates generated for the instrumentation logic are not intermingled with the gates generated from the non-instrumented source code. This permits design verification with gate-level logic that does not need to be re-verified after instrumentation verification. Thus the designer can verify the result of synthesis at the gate level while retaining RTL breakpoint feature. In some cases, however, the synthesis tool may not recognize that the same code appears twice. This may incur an additional relatively expensive phase of resource sharing in order to achieve the same performance results as the process illustrated in FIG. 3.

One advantage of the instrumentation process of FIG. 3 over that of FIG. 12, however, is that a synthesis tool can typically analyze the source code to detect obvious resource sharing.

The instrumentation methods of FIGS. 3 and 12 permit detecting any path that has been taken while a VHDL process or a Verilog "always" block is active. Tracking the activation of each process permits further analysis.

FIG. 17 illustrates a method of instrumenting the activation of the processes (or "always" blocks) themselves for subsequent determination of whether the process is active during gate-level simulation .

In step 1710, the sensitivity list of a process is identified. In step 1720, logic is generated to compare the signals in the sensitivity list between consecutive simulation cycles. Subsequently, during gate-level simulation in step 1730, a determination is made as to whether an event has occurred on any of the sensitivity list signals. Each simulation cycle that a signal indicates a difference (i.e., a signal event has occurred), the process is active as indicated by step 1740. Otherwise, if no events have occurred on any of the sensitivity list signals, the process is inactive as indicated by step 1750.

FIG. 18 illustrates the code added to determine if process P1 is active. The added code is italicized. The sensitivity list of process P1 includes signals a, b, and c. In accordance with step 1720 of FIG. 17, code section 1810 creates sampled versions of a, b, and c using FAST_CLK as described above. The sampled versions of a, b, and c are SAMPLED_A, SAMPLED_B, and SAMPLED_C, respectively.

Code section 1820 determines if an event has occurred on each of the sensitivity list signals. The test "(SAMPLED_A /=A)" is true if an event occurs with respect to signal A. Similarly "(SAMPLED_B /=B)" and "(SAMPLED_C /=C)" indicate whether an event has occurred with respect to signals B and C. Process P1 is active if any one of these tests is true. Thus the variable P1_ACTIVE is generated by combining each of these signal events using the logical OR function in code section 1820. Thus signal P1_ACTIVE indicates whether process P1 is active.

Process instrumentation data can be added to the instrumentation data cross-reference file in order to enhance the source code display. For example, the active process in a given cycle can be highlighted to indicate it is active. This permits the designer to visually see the active processes without having to determine the value of each signal. In one embodiment, the instrumentation data is used to enhance the display of the original RTL source code rather than the source code resulting from instrumentation.

The instrumentation techniques presented result in gate level designs providing explicit instrumentation signals to indicate that some specific portion of the source code is active. The number of instrumentation signals tends to increase with the complexity of the system being modeled.

Some optimizations may be performed to decrease the number of instrumentation signals. At least one execution path will be active any time a process is activated. As a result, the TRACE1 variable in the examples of FIGS. 6 and 9 tend to provide no additional information and thus SIG_TRACE1 is somewhat trivial as can be seen from the synthesized logic of FIGS. 7 and 10. Thus at least one trace variable (and therefore one output signal trace) can typically be eliminated.

In some cases the execution status of each branch of the code can be determined even though every branch is not explicitly instrumented. To verify the execution status of every branch, the instrumentation process need only ensure that each branch is instrumented either explicitly or implicitly through the instrumentation of other branches.

In some instances, the capacity of hardware triggers can be used to eliminate some of the instrumentation by combining several signals into one condition. The number of gates simulated can be reduced by replacing logical AND conditions that appear in the equations of instrumentation signals by simulator-specific triggers.

For example, consider the instrumented CASE statement code fragment 1910 illustrated in FIG. 19. For purposes of example, only the trace variable assignment statements are shown for the four possible cases. A synthesis tool will generate four comparisons with the vector "opcode." Each trace variable is associated with one of the possible values of opcode. Clearly, however, the additional logic is unnecessary because setting a breakpoint on any one of the case conditions corresponds to setting a trigger on the vector for the corresponding value of "opcode."

FIG. 20 illustrates a method for optimizing the instrumentation process. In particular, an instrumentation signal is selected in step 2010. In step 2020, a determination is made to whether the equation of the current signal can be expressed as a logical AND between a signal and a simplified expression. If so, then the AND gate should be eliminated in step 2030 and the extracted signal can be added to the trigger conditions during simulation in step 2040. If triggers can be activated on zeroes as well as ones, then step 2020 can also determine whether an equation can be simplified as a logical negation of a subexpression and the logical negation of the subexpression can be added to the trigger conditions during simulation in step 2040 where

appropriate. Step **2020** would then be applied recursively until the equation cannot be further simplified. This process is then applied to all of the instrumentation signals.

For example, signal TRACE4 is the result of performing a logical AND between opcode(**0**) and opcode(**1**). Thus TRACE4 is active only when opcode ="11". In accordance with FIG. **20**, the AND gate can be removed and the simulator trigger conditions would be changed from TRACE4=1 to "OPCODE(**0**)=1 AND OPCODE(**1**)=1." This process would then be applied recursively to all signals remaining in the trigger condition. Thus if OPCODE(**0**) happened to be the result of an AND between two other signals, the AND gate could again be eliminated from the synthesized gate-level design and the trigger conditions could be updated accordingly as long as no other signals used "OPCODE(**0**)" as an input. If no other logic uses "OPCODE(**0**)" as an input, then the trigger conditions can be updated to refer to the signals used to generate OPCODE (**0**) and the gate-level netlist AND gate can safely be eliminated. More generally, any optimization that consists of eliminating gates and other elements by transferring the implementation of the instrumentation logic to the logic analyzer of the target simulator can be performed.

Where permitted by the gate-level simulator, the instrumentation required for detecting activation of a process may similarly be reduced. In particular, greater efficiency may be possible by keeping a list of all the signals in the process sensitivity list and then testing whether events occurred on the signals in the sensitivity list. Further optimization may be made possible by sharing the logic for signals that appear on the sensitivity list of more than one process. The original signal can be sampled once initially. A comparison is made between the initial value and the current value of the signal to generate an event signal indicative of whether an event has occurred on that signal. The event signal can then be used for instrumentation of processes with events and for tracking process activation.

FIGS. **3**, **12**, and **17** illustrate methods of modifying the original RTL source code for instrumenting processes and level-sensitive and edge-sensitive source code. Trace variables (i.e., instrumentation variables) can be used to track the execution of any path within the source code. Additional output signals are generated from instrumentation variables in order to detect the execution paths of the source code. In the illustrated embodiments, the instrumentation variables are reset at the beginning of a process and the signals are assigned at the end of the process in order to ensure that all the signals are assigned regardless of which execution path is taken inside the process.

In an alternative embodiment, the signals might be directly assigned in the execution path of the process. Typically, this alternative embodiment would force the synthesis tool to generate complicated structures including latches due to the nature of HDLs and simulation rules.

The methods of FIGS. **3**,**12**, and **17** can be applied to the source code before the source code is synthesized. Thus in one embodiment the steps that modify the RTL source code can be performed before but entirely independently of the synthesis process itself.

FIG. **21** illustrates an embodiment in which the instrumentation data is generated entirely within the synthesis process. The process of creating output signals associated with synthesizable statements in the source code and then synthesizing the source code into a gate-level design including the output signal can be incorporated into the synthesis tool itself so that modification of the RTL source code is not required.

For example, one of the steps performed by a synthesis tool for generation of the gate-level design is parsing the RTL source code. Parsing the RTL source code results in a parser data structure that is subsequently used to generate the gate-level design. Instead of modifying the source code, the synthesis tool can simply set markers inside the parser data structure.

FIG. **22** illustrates one application of using the instrumentation signals for tracing execution flow using breakpoints. In step **2210**, the user sets a breakpoint at a specified line number of the source code. The specified line number is then associated with one of the instrumented lists of statements in step **2220**. In step **2230**, the instrumentation signal for the associated list is identified as the breakpoint output signal.

During the gate-level simulation run, the active lists (identified by transitions in their corresponding instrumentation signals) may be highlighted and displayed for the user as indicated in step **2240**. For example, the active lists may be portrayed in a different color than the inactive lists. Alternatively, the active lists may be displayed using blinking characters, for example. The instrumentation data file can be used to associate an instrumentation signal with a list of source code line numbers to be highlighted.

In response to a 0 to 1 transition in the breakpoint output signal, the simulation can be stopped as indicated in step **2250**. Thus through instrumentation the designer has the ability to effectively set breakpoints in the RTL source code which can be acted upon during RTL simulation.

The methods of instrumentation may be implemented by a processor responding to a series of instructions. In various embodiments, these instructions may be stored in a computer system's memory such as random access memory or read only memory.

The instructions may be distributed on a nonvolatile storage medium for subsequent access and execution by the processor. Typically the instructions are stored in the storage medium for distribution to a user. The instructions may exist in an application program form or as a file stored in the storage medium. The instructions are transferred from the nonvolatile storage medium to a computer system for execution by the processor.

In one embodiment, the program or file is installed from the storage medium to the computer system such that the copy of the instructions in the nonvolatile storage medium is not necessary for performing instrumentation. In another embodiment, the program or file is configured such that the original nonvolatile storage medium is required whenever the instructions are executed.

Nonvolatile storage mediums based on magnetic, optical, or semiconductor memory storage principles are readily available. Nonvolatile magnetic storage mediums include floppy disks and magnetic tape, for example. Nonvolatile optical storage mediums include compact discs, digital video disks, etc. Semiconductor-based nonvolatile memories include rewritable flash memory.

Instrumentation allows the designer to perform gate-level simulation of synthesized RTL designs with source-level debugging. In addition, the instrumentation process allows the designer to examine source code coverage during simulation.

In the preceding detailed description, the invention is described with reference to specific exemplary embodiments thereof. Various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method comprising the steps of:

a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.

2. The method of claim 1 wherein step b) includes the step of:

i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.

3. The method of claim 1 wherein step b) includes the steps of:

i) initializing a marker to a first value at the beginning of a process within the source code; and

ii) setting the marker to a second value within a same executable branch of the source code as the identified statement.

4. The method of claim 3 further comprising the step of:

iii) assigning the value of the marker to the instrumentation signal at the end of the process.

5. A method of generating a gate level design, comprising the steps of:

a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level design having the instrumentation signal.

6. The method of claim 5 wherein step a) further comprises the step of:

i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and

ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

7. The method of claim 6 wherein the variable is assigned a first value in step a)i), the method further comprising the step of:

iii) modifying the source code to initialize the unique variable to a second value.

8. The method of claim 5 wherein step a) is repeated to create a unique instrumented output signal for each list of sequential statements in the source code, wherein each list corresponds to a synthesizable executable branch of the source code.

9. The method of claim 5 further comprising the step of:

c) generating cross-reference instrumentation data mapping each statement in a selected list to the instrumented output signal associated with that list for every list in the source code.

10. The method of claim 9 further comprising the steps of:

d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.

11. The method of claim 5 further comprising the steps of:

c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.

12. A method of generating a gate-level netlist, comprising the steps of:

a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;

b) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;

c) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and

d) synthesizing the source code into a gate-level design including the instrumentation signals.

13. The method of claim 12 wherein step b) further comprises the steps of:

i) assigning each unique local variable a first value; and

ii) initializing each local variable with second value.

14. The method of claim 12 further comprising the step of

e) mapping every statement within a selected list to the corresponding instrumentation signal for that selected list as cross-reference instrumentation data.

15. The method of claim 12 further comprising the steps of:

e) setting a breakpoint at a selected statement of the source code;

f) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and

g) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in the breakpoint signal transitioning to a pre-determined value.

16. A method of generating a gate level netlist, comprising the steps of:

a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;

b) modifying the source code to generate a corresponding sampled version of each signal event in a selected process;

c) modifying the source code to duplicate the selected process;

d) replacing each occurrence of a selected signal event with the corresponding sampled version in the duplicated process;

e) replacing each list of sequential statements within an executable branch of the duplicated process with a unique variable assignment statement;

f) modifying the duplicated process to include an instrumentation signal assignment for each unique variable; and

g) synthesizing the modified source code into a gate-level design.

17. The method of claim 16 wherein step e) further comprises the steps of:

i) assigning the unique variables a first value; and

ii) initializing the unique variables with second value.

18. The method of claim 16 further comprising the step of

e) mapping every statement within each selected list to its corresponding instrumentation signal.

19. The method of claim 16 further comprising the steps of:

h) setting a breakpoint at a selected statement of the source code;

i) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and

j) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the breakpoint signal to a predetermined value.

20. A method of debugging a gate-level design including the steps of:

a) setting a breakpoint at a selected statement of a register transfer level (RTL) synthesizable source code;

b) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements, wherein the list corresponds to an executable branch of the source code including the selected statement;

c) modifying the source code to include an instrumentation signal assignment statement for the local variable; and

d) generating a gate-level design from the modified source code.

21. The method of claim **20** further comprising the steps of:

e) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the instrumentation signal to a pre-determined value.

22. The method of claim **20** wherein step b) further comprises the steps of:

i) assigning the local variable a first value; and

ii) initializing the local variable with second value.

23. The method of claim **20** further comprising the step of

e) mapping every statement within the executable branch of source code to the instrumentation signal.

24. A method of simulating a gate-level design comprising the steps of:

a) identifying a sensitivity list of a process;

b) generating logic to identify signal events for any signal in the sensitivity list; and

c) identifying the process as active during simulation when a signal event occurs for any signal in the sensitivity list.

25. The method of claim **24** wherein step c) further comprises the step of:

i) highlighting a source code description of the process displayed during simulation.

26. The method of claim **24** wherein step b) further comprises the step of:

i) sampling each signal in the sensitivity list to generate corresponding instrumented signals; and

ii) comparing each signal in the sensitivity list with its corresponding instrumented signal to test each signal in the sensitivity list for an event.

27. The method of claim **26** wherein step c) further comprises the step of:

i) generating an active process output signal defined by logically ORing the results of the comparisons.

28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.

29. The storage medium of claim **28** wherein the processor performs the steps of:

i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and

ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

30. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to perform the steps of:

a) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;

b) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and

c) synthesizing the source code into a gate-level design including the instrumentation signals.

31. The storage medium of claim **30** having stored therein further instructions to enable the processor to perform the step of:

d) mapping every statement within each selected list to its corresponding instrumentation signal.

32. A storage medium having stored therein processor executable instructions for debugging a gate level design during simulation, wherein when a breakpoint is set at a selected statement of a register transfer level (RTL) synthesizable source code the instructions enable the processor to perform the steps of:

a) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements within the source code, wherein the list corresponds to an executable branch of the source code including the selected statement;

b) modifying the source code to include an instrumentation output signal assignment statement for the local variable; and

c) generating a gate-level design from the modified source code.

33. The storage medium of claim **32** having stored therein further instructions to enable the processor to perform the step of:

d) mapping every statement within each selected list to its corresponding instrumentation signal.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,240,376 B1
DATED : May 29, 2001
INVENTOR(S) : Alain Raynaud and Luc M. Burgun

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 1,
Line 11, "hen" should read -- when --.
Line 17, "modem" should read -- modern --.
Line 37, "simulator. alternatively," should read -- simulator. Alternatively, --.
Line 38, "converting the ate-" should read -- converting the gate- --.
Line 47, "RTh" should read -- RTL --.

Column 2,
Line 39, "RTh" should read -- RTL --.

Column 10,
Line 33, "signal CLX_" should read -- signal CLK_ --.

Signed and Sealed this

Sixteenth Day of July, 2002

Attest:

Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office