(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 June 2007 (14.06.2007)

PCT

(10) International Publication Number
WO 2007/066320 A1

(51) International Patent Classification:
G06F 17/50 (2006.01)

(21) International Application Number:
PCT/IL2006/001349

(22) International Filing Date:
23 November 2006 (23.11.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/748,957 8 December 2005 (08.12.2005) US

(71) Applicant *(for all designated States except US)*: **MENTOR GRAPHICS CORPORATION** [US/US]; 8005 SW Boeckman Road, Wilsonville, Oregon 97070-9733 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **VELLER, Yossi** [IL/IL]; 9/a Hahagana Street, 46325 Herzliya (IL). **HANGA, Vasile** [IL/IL]; 3 Havradim Street, 42651 Netanya (IL). **ROZENMAN, Alexander** [IL/IL]; 13/10 Smilansky Street, 75258 Rishon Lezion (IL). **RACHAMIM, Rami** [IL/IL]; 12 Dultzin Street, 69630 Tel Aviv (IL).

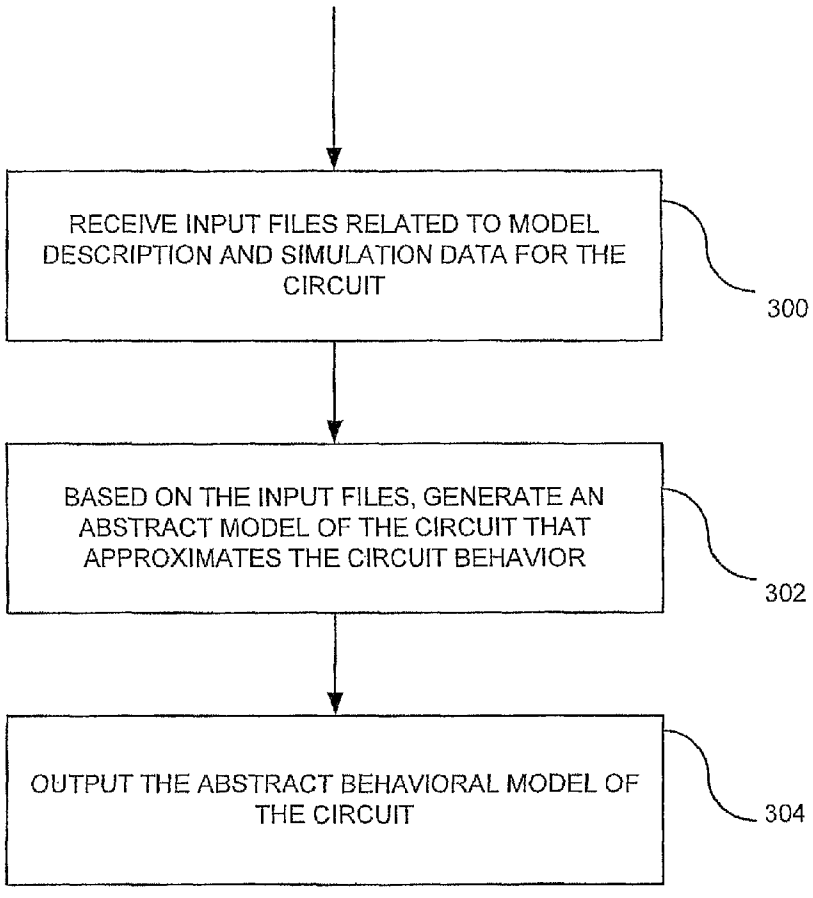
(74) Agents: **LUZZATTO, Kfir** et al.; Luzzatto & Luzzatto, Box 5352, 84152 Beersheva (IL).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: CONVERSION OF CIRCUIT DESCRIPTION TO AN ABSTRACT MODEL OF THE CIRCUIT



(57) Abstract: A system and method is disclosed for converting an existing circuit description from a lower level description, such as RTL, to a higher-level description, such as TLM, while raising the abstraction level. By changing the abstraction level, the conversion is not simply a code conversion from one language to another, but a process of learning the circuit using neural networks and representing the circuit using a system of equations that approximate the circuit behavior, particularly with respect to timing aspects. A higher level of abstraction eliminates much of the particular implementation details, and allows easier and faster design exploration, analysis, and test, before implementation. In one aspect, a model description of the circuit, protocol information relating to the circuit, and simulation data associated with the lower level description of the circuit are used to generate an abstract model of the circuit that approximates the circuit behavior.

WO 2007/066320 A1

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

CONVERSION OF CIRCUIT DESCRIPTION TO AN ABSTRACT MODEL OF THE CIRCUIT

Related application data

Priority is claimed to US provisional patent application 60/748,957, filed December 8, 2005, which is hereby incorporated by reference.

Field of the Invention

The present invention generally relates to simulation, and more particularly to converting a simulated circuit description to a higher level of abstraction.

Background of the Invention

The complexity of integrated circuits (ICs) being designed nowadays is continuously increasing and has resulted in complete system-on-chip (SoC) solutions. Even more, the complexity of such integrated systems is exploding thanks to advances in process fabrication. The limiting factor is now the ability to design, manage and verify such systems rather than the ability to fabricate them.

The typical design process begins with a software program that describes the behavior or functionality of a circuit. This software program is written in a hardware description language (HDL) that defines a behavior to be performed with limited implementation details. Logic synthesis tools convert the HDL program into a gate netlist description. The RTL description is used to verify functionality and ultimately generate a netlist that includes a list of components in the circuit and the interconnections between the components. This netlist is used to create the physical integrated circuit.

As SoC's are becoming larger, the only way to efficiently design such

dense SoC's, both from the design complexity and time-to-market aspects, is by embedding Intellectual Property (IP) cores. Standards for such cores are currently evolving. Ideally, they should be reusable, pre-characterized and pre-verified. But it often desirable to change the design to create the next generation. For example, as fabrication technology changes, it is desirable to convert or migrate the design to the new process parameters. For example, an IP core may be designed and tested for 90 nm technology, but it is desirable to convert the IP core to a new process of 60 nm technology. Or it may be desirable to update the design and incorporate changes in order to create the next generation design.

In order to test and explore such changes in the design, simulation must be performed, which is very time consuming. A few seconds of real-time simulation can take weeks or even months. If the simulation results are not desirable, then the design process must start over again by changing the high-level code and re-simulating.

Because of such delays in simulation, designers are beginning to move the design process to a higher level of abstraction (meaning less focus on design details). At the higher level of abstraction, design exploration can be performed to evaluate which performance and power consumption can be achieved, which parts to use, etc. The preferable higher level of abstraction is called Transaction Level Modeling (TLM), which refers to the evolving design and verification space called Electronic System Level (ESL) with methodologies that begin at a higher level of abstraction than the current mainstream Register Transfer Level (RTL). The main ESL design language SystemC, is driven from C/C++ rather than from hardware languages like Verilog and VHDL.

The challenge is how to rewrite or convert existing models and code at

the register transfer level to models and code at the electronic system level. There are some tools available that can make such a conversion, such as VTOC available from Tenison Corporation, but these tools do a simple code conversion without changing the level of abstraction. Thus, for example, having the same level of abstraction, including the same level of design details, means that the verification and simulation are just as slow.

A simple example is if an engineer wants to use an existing circuit, but increase the memory size. There are no guarantees that making such an update will work. For example, increased memory size may drain the battery too quickly rendering the circuit unmarketable. Using current tools, the designer must either physically implement the circuit to see if it works or modify the RTL code and simulate the design. Such simulation may take weeks or even months, and if the modification does not work, the process must be started over again.

Thus, it is desirable to convert an existing design from a lower level of code, such as RTL, to a higher level, such as ESL, while changing the abstraction level of the design in order to gain the benefits of having the code at the higher level.

Summary of the Invention

A system and method are disclosed for converting an existing circuit description, specifically its timing characteristics, from a lower level description, such as RTL, to a higher-level description, such as TLM, while raising the abstraction level. By changing the abstraction level, the conversion is not simply a code conversion from one language to another, but a process of learning the circuit using neural networks and representing the circuit using a system of equations that approximate the circuit behavior, particularly with respect to timing aspects. A

higher level of abstraction eliminates much of the particular implementation details, and allows easier and faster design exploration, analysis, and test, before implementation.

In one aspect, a model description of the circuit, protocol information relating to the circuit, and simulation data associated with the lower level description of the circuit are used to generate an abstract model of the circuit that approximates the circuit timing behavior.

In another aspect, such generation is accomplished using machine learning algorithms and/or a neural network. The neural network generates a system of weighted equations. Input patterns are used to calculate a difference between the actual output values (using the equations) and desired values (using simulated data) and extract a deterministic behavior. The weights of the equations can then be modified.

In yet another aspect, causality analysis is used in order to synthesize the model description, protocol information and simulation data. The synthesized data may then be passed more efficiently through the neural network.

In another aspect, the resulting abstract model can be simulated as-is to run pure performance analysis of a system, or can be plugged into TLM functional models and used to provide timing and functional behavior during fully functional simulation.

These features and others of the described embodiments will be more readily apparent from the following detailed description, which proceeds with reference to the accompanying drawings.

Brief Description of the drawings

Figure 1 is a high-level flowchart of a method for converting a circuit description from a lower-level description into a higher-level description, while changing the level of abstraction;

Figure 2 is a flowchart of a method for converting a description of a circuit simulation into a series of transactions through message extraction;

Figure 3 is a hardware diagram of a system used to convert the description of the circuit into transactions;

Figure 4 is a detailed example showing simulated signal data of the circuit description on numerous hardware lines;

Figure 5 is a detailed example of a state machine for some of the available transactions;

Figure 6 is a detailed flowchart of a method for converting the simulated circuit description into transactions;

Figure 7 is a flowchart of a method for converting a series of transactions into a super-transaction representation;

Figure 8 shows a transaction-based view of the simulation data that may be displayed to the user;

Figure 9 is a flowchart of a method for performing model extraction of the circuit;

Figure 10 is a flowchart of a method providing further details for generating an abstract model;

Figure 11 is a hardware diagram of a system used to convert transaction data into an abstract model;

Figure 12 is an example of a fork table used in generating the abstract model;

Figure 13 is an example of a latency table used in generating the abstract model;

Figure 14 is a flowchart of a method for performing causality analysis;

Figure 15 is a flowchart of a method performed by a neural network for generating a system of equations approximating the circuit behavior;

Figure 16 shows a network that may be used to implement the invention; and

Figure 17 is an exemplary flowchart of a method for implementing the invention over the network of Figure 16.

Detailed Description of preferred Embodiments

Figure 1 shows a high-level flowchart for converting a circuit description from a low-level description (e.g., HDL, RTL) to a higher level of abstraction, such as a transaction level model (TLM). The low-level description generally includes details at the signal level, while the TLM uses high level functions and equations to calculate output transactions based on inputs and is not concerned with the device-level implementation of the circuit. ESL is an emerging electronic design methodology, which focuses on the higher abstraction level. Electronic System Level is now an established approach at most of the world's leading System-on-a-chip (SoC) design companies, and is being used increasingly in system design. From its genesis as an algorithm modeling methodology with 'no links to implementation', ESL is evolving into a set of complementary methodologies that enable embedded system design, verification, and debugging through to the hardware and software implementation of custom SoC, system-on-FPGA, system-on-board, and entire multi-board systems. ESL can be accomplished through the use of SystemC as an abstract modeling language.

At process box 10, simulation is performed on the low-level circuit

description. At process box 12, transactions are extracted from the simulation data. The simulation and transaction extraction process are described more fully in relation to Figures 2-8, but basically the system maps signal patterns into messages using pre-defined protocols (e.g., AMBA, PCI, etc.). Then the messages are converted to transactions. At process box 14, model extraction is performed. The model extraction is described more fully in relation to Figures 9-18, but generally the system looks to repetitive correlation (i.e., deterministic behavior) between input sequences and output messages. Neural network functions are used to calculate the output message generation and extrapolate statistical behavior of a component. Additionally, data dependencies can be extracted. Finally, in process box 16, the model is output at the higher level of abstraction. The model, in a sense, is like a black box where input transactions/messages are analyzed to generate output transactions/messages, without a focus on signal levels and values, but more a focus on timing and relationships between messages. The resulting abstract model can be simulated as-is to run pure performance analysis of a system, or can be plugged into TLM functional models and used to provide timing and functional behavior during fully functional simulation.

Figure 2 shows a flowchart of a method for converting simulation data of a circuit description to a transaction-based description, which is at a higher layer of abstraction. In process box 20, simulation data of a circuit description is received. The circuit description may be in HDL or any other software language and it may be compiled and simulated as part of a system design flow or it may be separately compiled and simulated. Thus, the simulation can be run in combination with the conversion process to a transaction-based description, or it can be run on a separate machine at a separate time. Any desired simulator may be

used, such as ModelSim®, available from Mentor Graphics Corporation, or VCD (Value Change Dump) files generated by any other simulator. In process box 24, the simulated circuit is converted into a series of transactions associated with a predetermined protocol. The protocol used is typically provided as input into the system by the user. In process box 26, the simulation data is output in the form of the transactions, which is a higher level of abstraction than the received simulated circuit design. For example, Figure 4 shows a simulated circuit description, which is at a signal level including a plurality of signals on various hardware lines. Figure 8 illustrates the converted circuit description at a transaction level. The output may be achieved by a variety of techniques, such as displayed to the user on a display (not shown), output to a file, etc.

Figure 3 shows a hardware diagram of a system 38 for converting a circuit description into a circuit description at the transaction level. A storage device 40 of any desired type has stored thereon the circuit design in HDL or any other desired language that may be used to describe circuits. A compiler 42 compiles the design and a protocol library 44. The compiler 42 may be any desired compiler and is usually included as part of a simulator package. The protocol library 44 includes messages and transactions associated with a protocol used by the circuit. Messages include part of a transaction, such as a request and an acknowledge of the bus, whereas a transaction is a complete operation, such as any of a variety of types of Read or Write transactions or control or setup transactions. A simulation kernel 46 simulates the compiled design in a well-known manner, as already described. The simulation kernel 46 outputs the simulation data 48 in any desired format. Box 48 can also represent a pre-simulated design data (VCD format).

A message recognition module 50 reads the simulation data 48 and analyzes the data to convert it to messages of the protocol stored in the protocol library 44. Figures 4-6 describe this conversion more thoroughly, but generally switching signals of the simulation are compared (during various time slices) to messages within the protocol library 44 to determine what message is being processed during a particular time slice. The messages associated with the switching signals during each time slice are then stored to convert the switching signals into messages.

A transaction recognition module 52 reads the messages determined by the message recognition module 50 and converts the messages into transactions using a comparison of a series of messages to predetermined messages within the protocol library 44. If a match is found, then the transaction recognition module stores the series of messages as a transaction. The result is that the messages are converted into a series of transactions.

A transaction sequence recognition module 54 converts multiple transactions into a single super-transaction sequence. For example, several Writes can be converted into a single control operation. This conversion from multiple transactions to a super-transaction sequence is described further below in relation to Figure 7. If desired, the transaction sequence recognition module 54 may be bypassed or omitted, so that the transactions are output directly. Results 56 of the conversion are output onto a storage medium or a display.

In any event, the simulated circuit description is taken to a higher level of abstraction, as the simulation data is converted first to messages, then to transactions, and finally, if desired, to transaction sequences.

The compiler 42, simulator kernel 46, and modules 50, 52, 54, may all be run on the same computer. Alternatively, the circuit description may be compiled and simulated in a different location so that the resultant simulation data 48 is merely on a storage medium to be input into the message recognition module 50. In such a case, as shown at 58, it is desirable that the some of the protocol data from the protocol library 44 is incorporated into the simulation data in a pre-processing step.

Figure 4 shows a detailed example of part of the simulated signal data 48. Various signal data 70 on hardware lines are shown including a clock line 72, a read/write line 74, a bus request line 76, a ready line 78, address lines 80, and data lines 82. Simulation is also carried out on many more hardware lines, which are not shown for convenience. The signals being simulated follow a predetermined protocol 84. A protocol is a set of rules or standards designed to enable circuit elements to communicate together and exchange information with as little error as possible. The protocol 84 is made up of a plurality of transactions 85, such as shown at 86 (i.e., transaction A) and at 88 (i.e., transaction B). A transaction is a discrete activity, such as a Read or Write operation that moves data from one place to another in the system. The transactions 86, 88 are in turn made up of a series of messages 90. For example, transaction 86 is shown as including three messages, 92, 94, and 96. A message is a smaller unit of information electronically transmitted from one circuit element to another to facilitate the transaction. Example messages include "request for bus", "acknowledge", "ready", etc. Those skilled in the art will readily recognize that these are only examples of transactions and messages and others may be used. Each message is associated with a time-slice 98, such as those shown at 100, 102, and 104. Normally, the time-slices are based on the clock signal 72. During each time-slice, the hardware lines 70 are analyzed to determine the

message being sent in correspondence with the transactions of the protocol, as further described below. Transaction 88 is similar to transaction 86 and need not be further described.

Figure 5 shows an example part of a state machine 120 stored within the protocol library 44. Different states 122 are shown as numbered circles. Messages, such as those at 90, are shown in boxes, and cause the state machine to move from one state to another. Transactions may be defined by a path through the state machine 120 that starts at an idle state 124 (state 0) and that ends at the same idle state, although those skilled in the art will recognize that the state machine 120 may be constructed in a variety of different formats. For example, a read transaction 126 is made up of numbered states 0, 1, 2, 3, 4 and 5. The read transaction 126 is completed upon return to the idle state from state 5 to state 0, as shown by arrow 128. A write transaction 130 is made up of numbered states 0, 1, 2, 6, 7, 8, 9, and 10. The write transaction 130 is completed upon return to the idle state from state 7 to state 0, as shown by arrow 132.

Figure 6 shows a flowchart of a method preformed by the message recognition module 50 and the transaction recognition module 52 in order to convert the simulation data into a transaction-based description. At process box 150, the simulated input data (see box 48 in Figure 3) is received so that it may be used by the message recognition module 50. Such simulation data is normally within a database. In process box 152, the analysis starts by monitoring the signal data 70 on the various hardware lines upon which messages are received. Additionally, in process box 152, the protocol library 44 is read to access a state machine, such as state machine 120, associated with the protocol. In process box 154, in order to analyze a transaction, an assumption is

made that the transaction starts from the idle state 124. In process box 156, a time-slice of data is read corresponding to the clock signal on hardware line 72. For example, in Figure 4, the data may be read starting with a time-slice 100. Thus, the switching signals on the various hardware lines are read in order to be analyzed. In process box 158, the data read is analyzed by comparing the switching signals to known patterns of messages stored in the protocol library 44. Returning briefly to Figure 5, from the idle state 124, a bus request message changes the state of the state machine to state 1. A bus request message has a particular pattern of signal data on the hardware lines, which is compared to a known pattern in the protocol library 44. Thus, once a match is found between the known pattern of messages and the message analyzed during the currently analyzed time-slice, the message has been determined and is stored in process box 160. In process box 162, the current state of the state machine is updated to reflect the change of state. Continuing with the example, the new state is state 1 after a bus request message is received. In decision box 164, a determination is made whether the state machine has returned to the idle state. If yes, this indicates that a transaction is complete and the transaction is determined in process box 166 by comparing a sequence of the stored messages to a sequence of known messages in the protocol library 44. The sequence of stored messages are those received from the start of the idle state until the state machine returned to the idle state. Once a match is found between the sequence of stored messages and those in the protocol library, the transaction associated with those messages is easily obtained from the protocol library 44. The determined transaction is then stored as indicated in process box 166. In decision box 168, a check is made whether all of the input simulated signal data has been analyzed by reading whether the database including the signal data is at the end. If yes, the method ends as shown at 170. Otherwise, the

method continues at process box 156 and the next time-slice is read (e.g., time-slice 102). Once the method ends, the database of signal data is converted into a series of transactions associated with the protocol found in the protocol database 44.

Figure 7 shows a method implemented by the transaction sequence recognition module 54 (see Figure 3). It may be desirable to group transactions together in order to display to a user the circuit at an even higher level of abstraction. For example, several write / read transactions can be shown as a single control transaction as opposed to individual transactions. In process box 200, a group of transactions is selected. For example, if there are many of the same type of transactions in sequence (e.g., Reads), such a sequence may be condensed. In process box 202, the selected group is compared to predetermined groups. In decision box 204, a determination is made whether there is a match between the selected group and the predetermined groups. If there is a match, then in process box 206, the sequence of transactions is stored as a single transaction in order to convert the circuit description to an even higher level of abstraction. In decision box 208, a check is made whether all of the transactions have been read. If yes, then the method ends at 210. If not, then a new group of transactions is chosen at 212, and the process starts over at process box 202.

Figure 8 shows an example of a display showing the simulation data of Figure 3 at a higher level of abstraction. Particularly, instead of signals, the simulation data is shown as a series of transactions. Write transactions, such as at 240, are shown as dotted lines and read transactions, such as shown at 242, are shown as solid lines. Throughput is shown along the Y-axis and time is indicated along the X-

axis. Thicker lines generally mean there is a grouping of many transactions so close in time that at the current zoom level they cannot be distinguished. Of course, a zoom option may be used to focus on particular transactions. As can readily be seen, the view of Figure 8 is much easier to read than that of Figure 4 and allows the designer to obtain a better overall system view of the flow of data.

Figure 9 shows a flowchart of a method for implementing model extraction 14 (Figure 1). In process box 300, input files are received related to protocol information, model description, and simulation data for the circuit. The protocol information is provided by the user and is stored in the protocol library 44. The model description is also provided by the user and includes an interface of the circuit model describing the input/output ports and the lasting state description of the circuit model that describes the internal states elements thereof. The simulation data may be simulation data 48 (see Figure 3) or simulation data at the transaction level 56 (Figure 3). In process box 302, using the input files, an abstract model is generated that approximates the circuit behavior. Although particular values may be associated with the approximated circuit behavior, in general the timing aspects are the focal point. For example, a particular address and read data are of less importance than when the address arrives and when the data is output. Such parameters can be added manually as they are easier to model (functionality is in many cases more simple than timing behavior). In process box 304, the abstract behavioral model is output.

Figure 10 is a flowchart of showing further details of process box 302. In process box 320, a set of tables is created that is associated with the input files. As explained further below, these tables are used to combine all of the input information into a desirable format for the causality

analysis and the learning phase. In process box 322, causality analysis is performed on the tables. The causality analysis is described further in Figure 14, but generally it is an analysis on the inputs in the table and the outputs in order to find a repetitive correlation there between. When there is a high degree of repetitive correlation of particular 'events', such events are given higher importance. On the other hand, signals that are seen only once may be disregarded in order to lessen the analysis of the learning phase. In process box 324, learning is performed. The learning is described further in Figure 15, but generally "learning" is a standard term used in the industry, especially relating to neural networks. For example, an article entitled "Conditional Distribution Learning with Neural Networks", IEEE Signal Processing 1997, written by Tulay Hadah, Xiao Liu, and Kemal Sonmer describes some aspects of "learning" using neural networks. In process box 326 model checking is performed in order to compare the generated model to the desired results.

Figure 11 shows a part of the system for performing the model extraction. Some aspects in Figure 11 have been already discussed. For example, the simulation data 56 and the protocol library 44 were discussed in relation to Figure 3. Although the simulation data 56 is shown at the transaction level, it may be simulation data 48, if desired. However, simulation data at the transaction level allows much less data to be fed into the analysis, significantly speeding the process. A protocol source file 350 is passed through a compiler 352 and the result is stored in the protocol library 44. Lasting state information source file 354 contains information regarding the inner states of the circuit being analyzed (e.g., describes registers in the circuit) and is also compiled in compiler 356 and stored in a file called Model Data 358. An interface source file 360 contains information regarding the input and output

ports of the circuit being analyzed. File 360 is passed through compiler 362 and combined with the compiled lasting state file 354 within the model data file 358. The above-described compiled files are passed together with the simulation data 56 to a table generator 370. The table generator uses all of the input files to generate multiple tables, including fork tables 372, latency tables 374, and data tables 376. The fork table 372 includes information regarding which path was taken during simulation when a branch was encountered in the protocol. Figure 12 provides an example fork table and is described further below. The latency tables 374 include information regarding the delay from a change of input until the corresponding output is changed. The data tables 376 include values associated with the output. In general, data values are not needed because timing is more interesting for the overall analysis. However, some data values may be tracked depending on options set by the user.

The table generator 370 outputs the resulting tables to the causality analysis engine 380 and to a neural network 302. As described further below, the causality analysis engine performs time-based causality analysis by applying a number of algorithms to each output message to compute the most likely causality basis. The results are also statistically analyzed and reduced so that only the most pertinent information is fed to the neural network 382. The neural network 382 generates equations that approximate the circuit behavior. Those skilled in the art will recognize that the neural network can be replaced by any other machine learning or statistical algorithm. The model checker 384 performs a check by comparing the inputs and outputs using the generated equations to the simulated data.

Figure 12 shows an example fork table 400 generated by the table

generator 370. The fork table includes multiple rows 402 representing events and multiple columns 404, most of which represent lasting state parameters. Column 406 includes a fork field. The fork field may include numbers (not shown) indicating which direction a fork was taken in association with an event and the associated lasting state parameters.

Figure 13 shows an example of a latency table 410. The latency table also includes rows 412 representing events. Many columns 414 represent lasting state parameters. The last three columns 416, 418, and 420 represent the event name, the time, and the latency, respectively. Some simple examples showing possible values are shown.

As is well known, the format and fields within a table is design specific and a wide variety of different formats and fields may be used.

Figure 14 is a flowchart of a method showing the operation of the causality engine 380. In process box 440, a set of causality characters is defined. Basically, when a repetitive correlation between inputs and outputs is found, a character is assigned to such a situation. For each output message in the latency table, causality characters are defined with each character represented as a pair having the form (event, time delta). Thus, the causality character describes a situation in which the specified event causes the output message after a given period of time. In process box 442, the number of causality characters is statistically reduced. Reduction of information ultimately provided to the learning process increases the speed of the system. Elimination of some characters can be accomplished using a hypothesis algorithm that provides a probability for a character to be part of the actual causality model. Thus, characters with limited appearances are generally eliminated. In process box 444, the causality characters are further

reduced using a genetic optimization algorithm that creates a model for the least amount of causality characters possible and still allowing to choose a cause for each output message instance. In process box 446, tables are created including will and time tables. The will table relates to something that caused an output change, such as an input in combination with a lasting state. The time table relates contains the remaining character lines (after the reductions) with the latency time value.

Figure 15 is a flowchart of a method for performing "learning" 324 (Figure 10). In process box 460, the tables generated in process box 446 (Figure 14) are used as well as tables generated from the table generator 370 (Figure 11) in order to create a system of weighted equations that represent the behavior of the circuit. Thus, for example, the inputs and outputs are analyzed in conjunction with state information to generate the equations. Such a generation of equations is well known in the art using standard techniques of neural networks. In process box 462, input patterns are applied to the generated system of equations to generate actual values produced by the equations. In process box 464, an error is calculated by using a difference between the actual values (process box 462) to the desired values (determined during simulation). In process box 466, based on this difference, the weightings in the system of equations are modified in order to more closely match the desired values. In decision box 468, a check is made whether the actual values generated by the system of equations are within an acceptable limit. If so, the flowchart is exited at 470. In not, the flow returns to process box 462 in order to re-analyze the equations.

Figure 16 shows that portions of the system may be applied to a distributed network, such as the Internet. Of course, the system also

may be implemented without a network (e.g., a single computer). A server computer 480 may have an associated database 482 (internal or external to the server computer). The server computer is coupled to a network shown generally at 484. One or more client computers, such as those shown at 488 and 490, are coupled to the network to interface with the server computer using a network protocol.

Figure 17 shows a flow diagram using the network of Figure 16. In process box 500, the circuit description to be transformed is sent from a client computer, such as 488, to the server computer 480. In process box 502, the abstract model of the circuit description is generated that approximates or imitates the circuit behavior, as previously described. In process box 504, the generated abstract model is checked against simulation results. In process box 506, the results are sent though the network to the client computer 488. Finally, in process box 508, the results are displayed to the user. It should be recognized that one or more of the process boxes may be performed on the client side rather than the server side, and vice versa.

Having illustrated and described the principles of the illustrated embodiments, it will be apparent to those skilled in the art that the embodiments can be modified in arrangement and detail without departing from such principles.

In view of the many possible embodiments, it will be recognized that the illustrated embodiments include only examples of the invention and should not be taken as a limitation on the scope of the invention. Rather, the invention is defined by the following claims. We therefore claim as the invention all such embodiments that come within the scope of these claims.

Claims:

1. A method for converting a description of a circuit into an abstract model, comprising:

reading a model description of the circuit, protocol information relating to the circuit, and simulation data associated with the circuit; and

based on the model description, the protocol information, and the simulation data, generating an abstract model of the circuit that approximates the circuit behavior.

2. The method of claim 1, wherein generating the abstract model includes generating a system of equations that represent the behavior of the circuit.

3. The method of any of claims 1 and 2, wherein generating an abstract model includes performing causality analysis including determining repetitive behavior between multiple input and output signals of the simulation data.

4. The method of any of the preceding claims, further including:

performing causality analysis including defining a set causality characters, each causality character representing an association between an event and an output; and

reducing the set of causality characters through statistical analysis.

5. The method of claim 4, further reducing the set of causality characters using a genetic optimization algorithm.

6. The method of any of the preceding claims, wherein generating the abstract model further includes generating fork tables and latency tables wherein the fork tables include state information and path information associated with the circuit and the latency tables include a latency period associated with a simulated event.

7. The method of any of the preceding claims, wherein the simulation data is at a transaction level.

8. The method of any of the preceding claims, wherein generating an abstract model includes learning the circuit using neural networks or any other machine learning or statistical algorithm.

9. The method of claim 8, wherein learning includes:

generating a system of weighted equations representing the behavior of the circuit;

applying input patterns to the system of equations to generate actual output values;

calculating an error by using a difference between the actual values and desired values;

modifying the weightings in the system of equations based on the calculated error.

10. The method of any of the preceding claims, wherein the simulation data is at a transaction-level through a conversion including:

reading low-level simulation data obtained from simulation of the circuit description;

analyzing switching signals, from the low-level simulation data, on simulated hardware lines of the circuit;

determining transactions of the protocol associated with the analyzed switching signals; and

storing a converted description of the simulated circuit as a series of the determined transactions.

11. The method of any of the preceding claims, wherein the circuit description is in a register transfer level and the abstract model is at a transaction-level model, the transaction-level model having a different level of abstraction than the register transfer level.

12. The method of any of the preceding claims, wherein at least one of the model description of the circuit, the protocol information relating to the circuit, or the simulation data, is provided from a client computer coupled to a network and generating the abstract model is performed on a server computer coupled to the network.

13. A computer-readable medium including instructions stored thereon for performing the method of claim 1.

14. An apparatus to convert a description of a circuit into an abstract model, comprising:

a causality engine coupled to at least one database including a protocol library, model data, and simulation data, the causality engine to determine deterministic behavior between multiple input and output signals of the simulation data; and

a neural network coupled to the causality engine to generate an abstract model of the circuit approximating the circuit behavior based on the determination of repetitive behavior by the causality engine.

15. The apparatus of claim 14, further including a table generator coupled between the causality engine and the neural network, the table generator for receiving the protocol library, the model data, and the simulation data and for generating tables to be used in the neural network.

16. The apparatus of any of claims 14-15, further including a model checker coupled to the neural network, the model checker for testing the abstract model of the circuit.

17. The apparatus of any of claims 14-16, wherein the causality engine is located on a client computer and the neural network is located on a server computer.

18. A method for generating an abstract model of a circuit, comprising:

performing causality analysis on circuit simulation data at a transaction-level to determine an association between input signals and output signals and producing a causality output based on the analysis; and

learning the circuit behavior using the causality output in order to produce the abstract model including a sequence of equations that approximate the circuit behavior.

19. The method of claim 18, further including converting low-level simulation data to the simulation data at a transaction-level including:

reading the low-level simulation data obtained from simulation of the circuit description;

analyzing switching signals, from the low-level simulation data, on simulated hardware lines of the circuit;

determining transactions of a protocol associated with the analyzed switching signals; and

storing a converted description of the simulated circuit as a series of the determined transactions.

20. The method of any of claims 18-19, wherein performing causality analysis includes:

defining a set of causality characters based on the association between input and output signals;

statistically reducing the set of causality characters; and

generating tables based on the reduced set of causality characters.

21. The method of any of claims 18-20, wherein the abstract model is in an electronic-system-level model.

22. An apparatus for converting a circuit description into an abstract model of the circuit, comprising:

means for reading a model description of the circuit, protocol information relating to the circuit, and simulation data associated with the circuit; and

means generating an abstract model of the circuit that substantially imitates the circuit behavior using the model description of the circuit, the protocol information, and the simulation data.

23. A method for generating an abstract model of a circuit, comprising:

receiving a circuit description in a register transfer level having a first level of abstraction;

converting the circuit simulation description into a transaction level description; and

using a neural network, converting the transaction level description into transaction-level model having a second level of abstraction, different from the first level of abstraction.

FIGURE 1

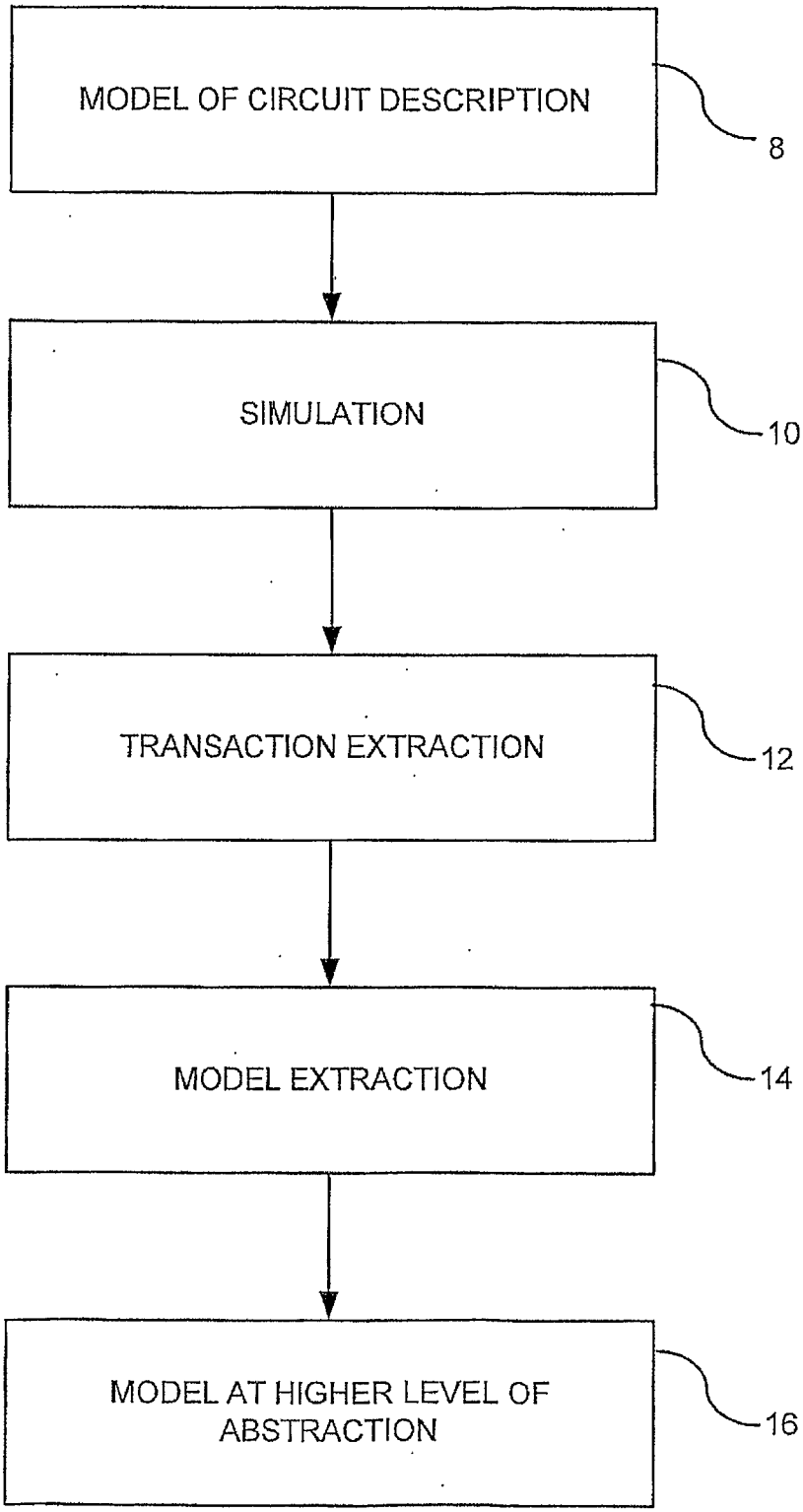

MODEL OF CIRCUIT DESCRIPTION
8
SIMULATION
10
TRANSACTION EXTRACTION
12
MODEL EXTRACTION
14
MODEL AT HIGHER LEVEL OF ABSTRACTION
16

FIGURE 2

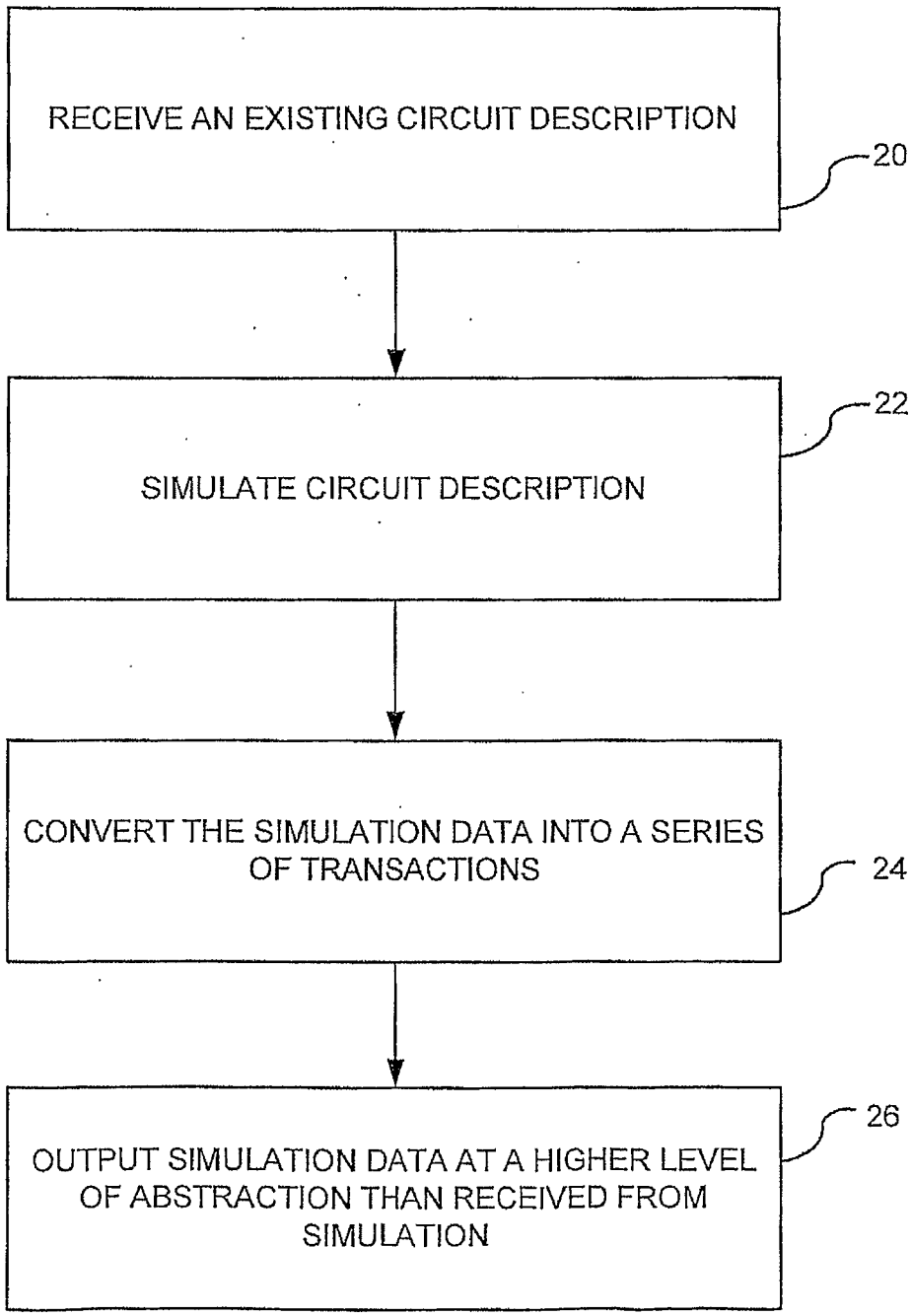
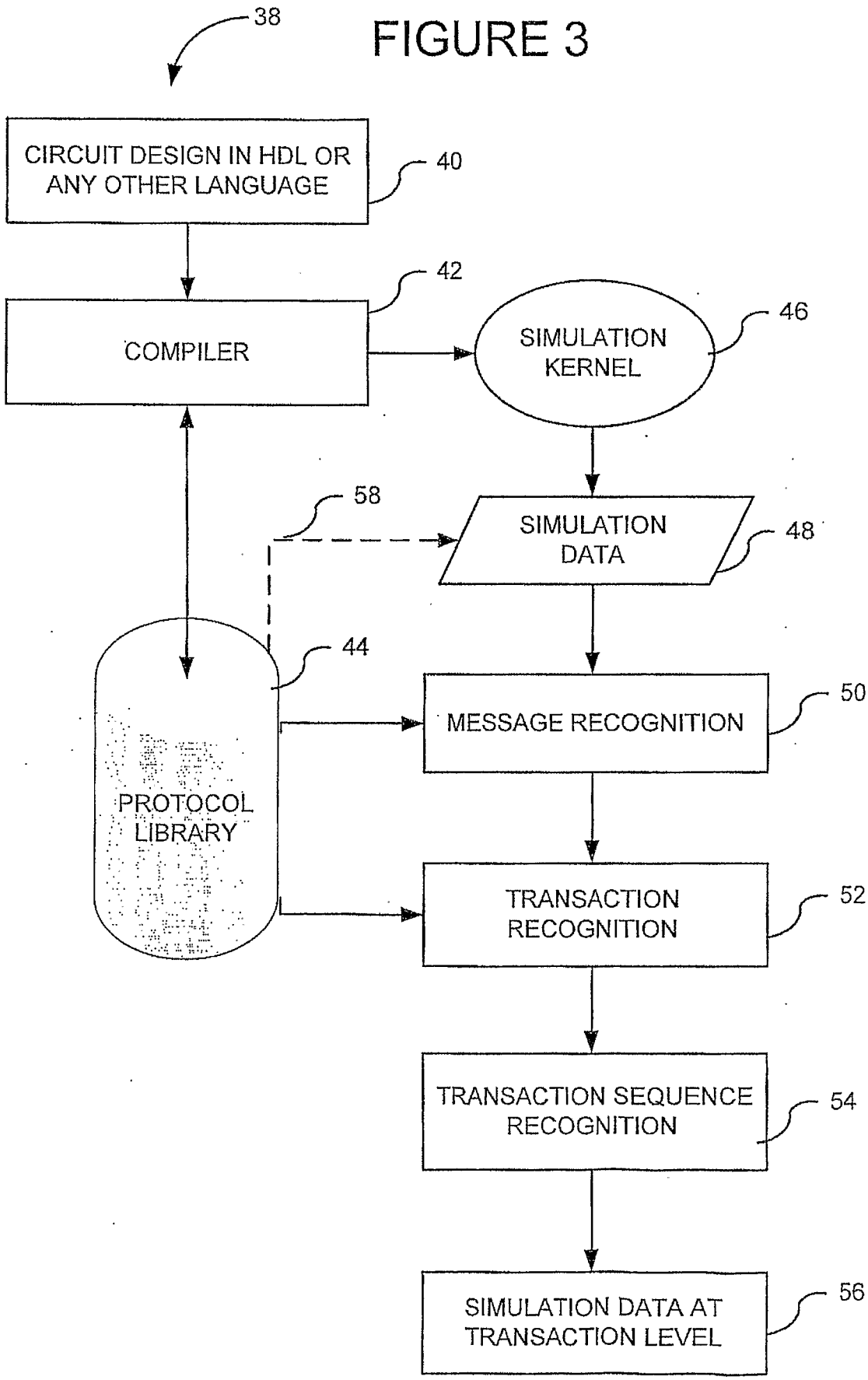

RECEIVE AN EXISTING CIRCUIT DESCRIPTION
20
SIMULATE CIRCUIT DESCRIPTION
22
CONVERT THE SIMULATION DATA INTO A SERIES OF TRANSACTIONS
24
OUTPUT SIMULATION DATA AT A HIGHER LEVEL OF ABSTRACTION THAN RECEIVED FROM SIMULATION
26

FIGURE 3


38
CIRCUIT DESIGN IN HDL OR ANY OTHER LANGUAGE
40
COMPILER
42
SIMULATION KERNEL
46
SIMULATION DATA
48
58
PROTOCOL LIBRARY
44
MESSAGE RECOGNITION
50
TRANSACTION RECOGNITION
52
TRANSACTION SEQUENCE RECOGNITION
54
SIMULATION DATA AT TRANSACTION LEVEL
56

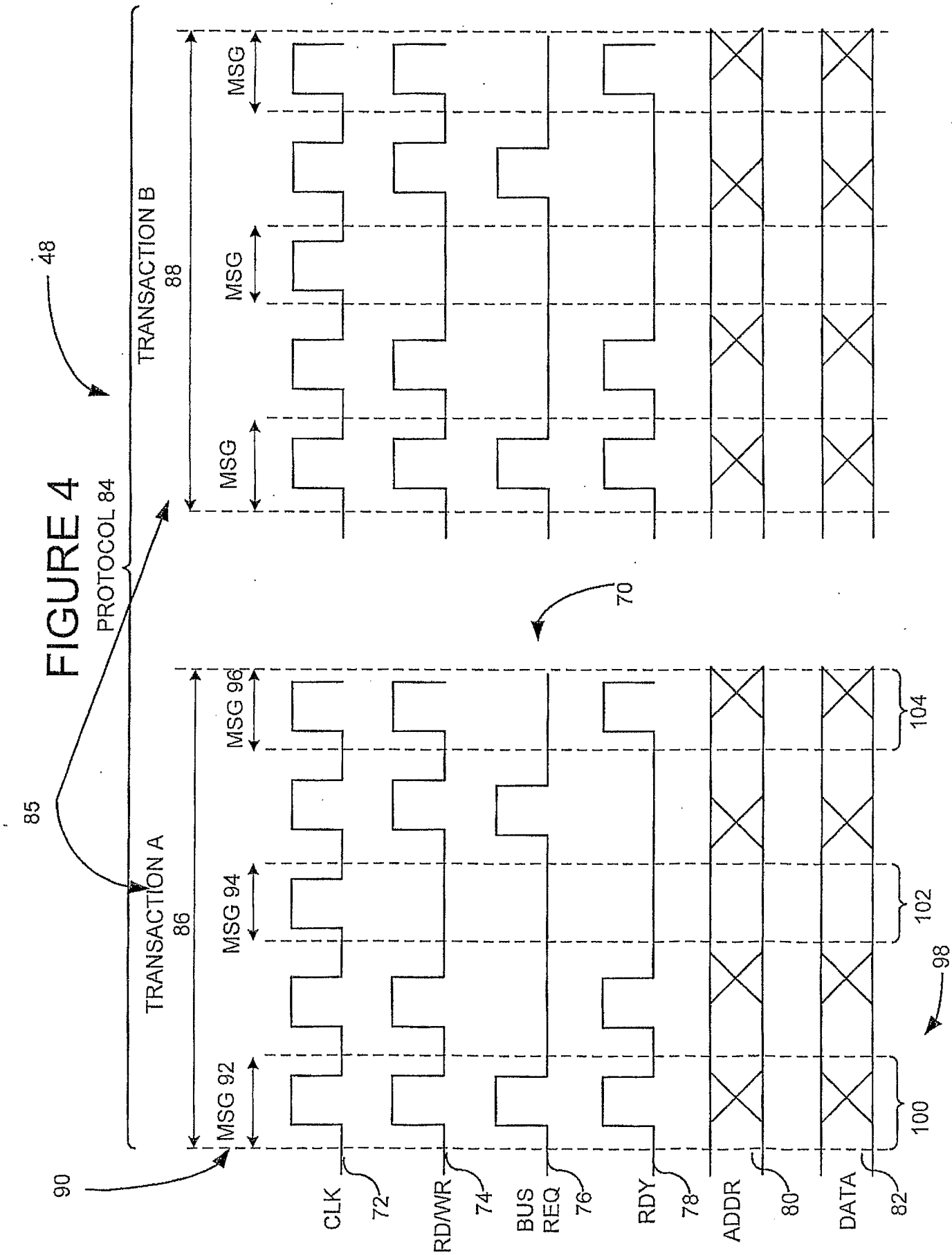


FIGURE 4
48
PROTOCOL 84
TRANSACTION A
86
TRANSACTION B
88
MSG 92
MSG 94
MSG 96
MSG
MSG
MSG
85
90
70
98
100
102
104
CLK 72
RD/WR 74
BUS REQ 76
RDY 78
ADDR 80
DATA 82

FIGURE 5

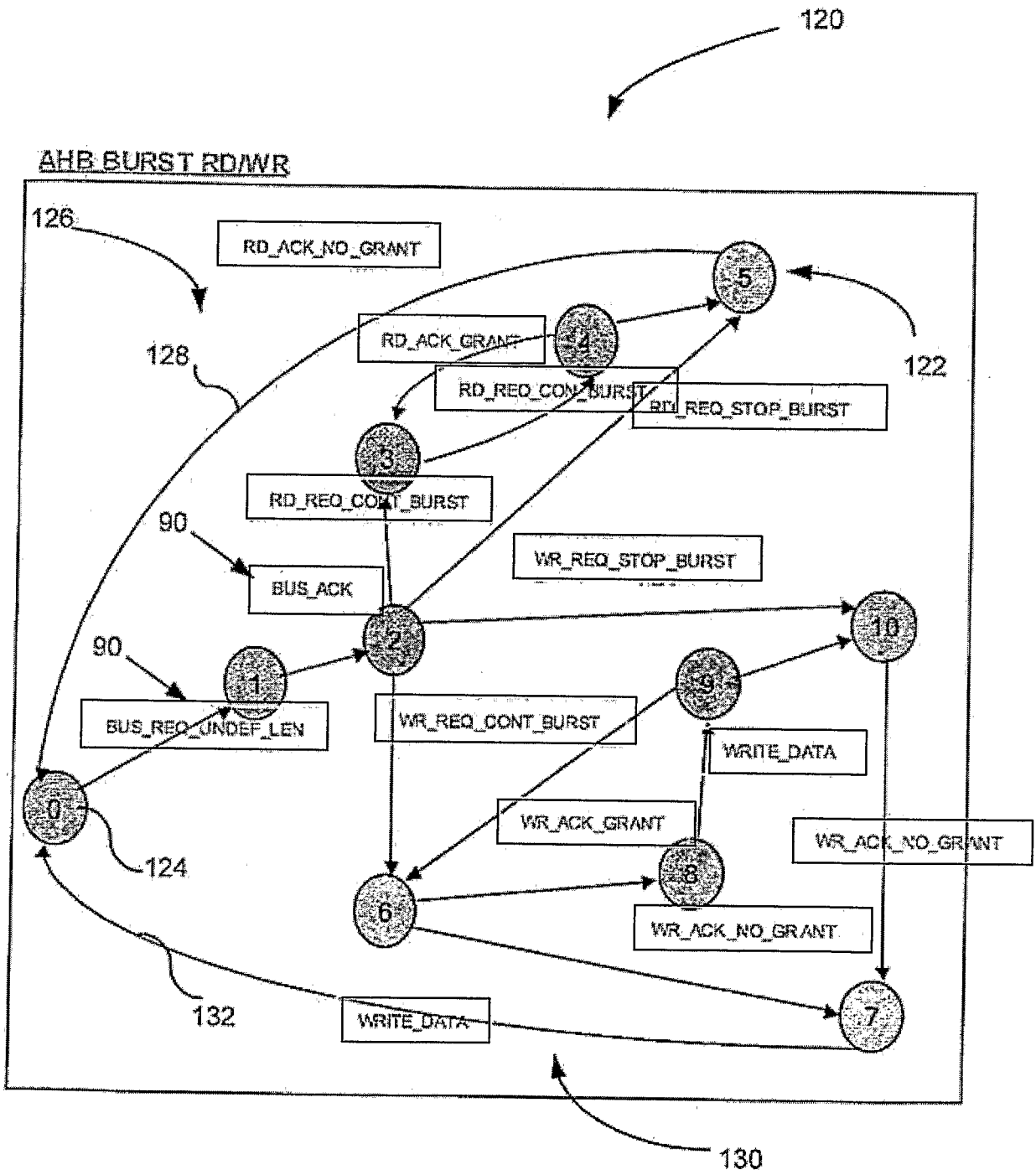

120
AHB BURST RD/WR
126
RD_ACK_NO_GRANT
5
RD_ACK_GRANT
4
128
122
RD_REQ_CONT_BURST
RD_REQ_STOP_BURST
3
RD_REQ_CONT_BURST
90
WR_REQ_STOP_BURST
BUS_ACK
2
10
90
1
9
BUS_REQ_UNDEF_LEN
WR_REQ_CONT_BURST
WRITE_DATA
0
WR_ACK_GRANT
WR_ACK_NO_GRANT
124
8
6
WR_ACK_NO_GRANT
7
132
WRITE_DATA
130

FIGURE 6

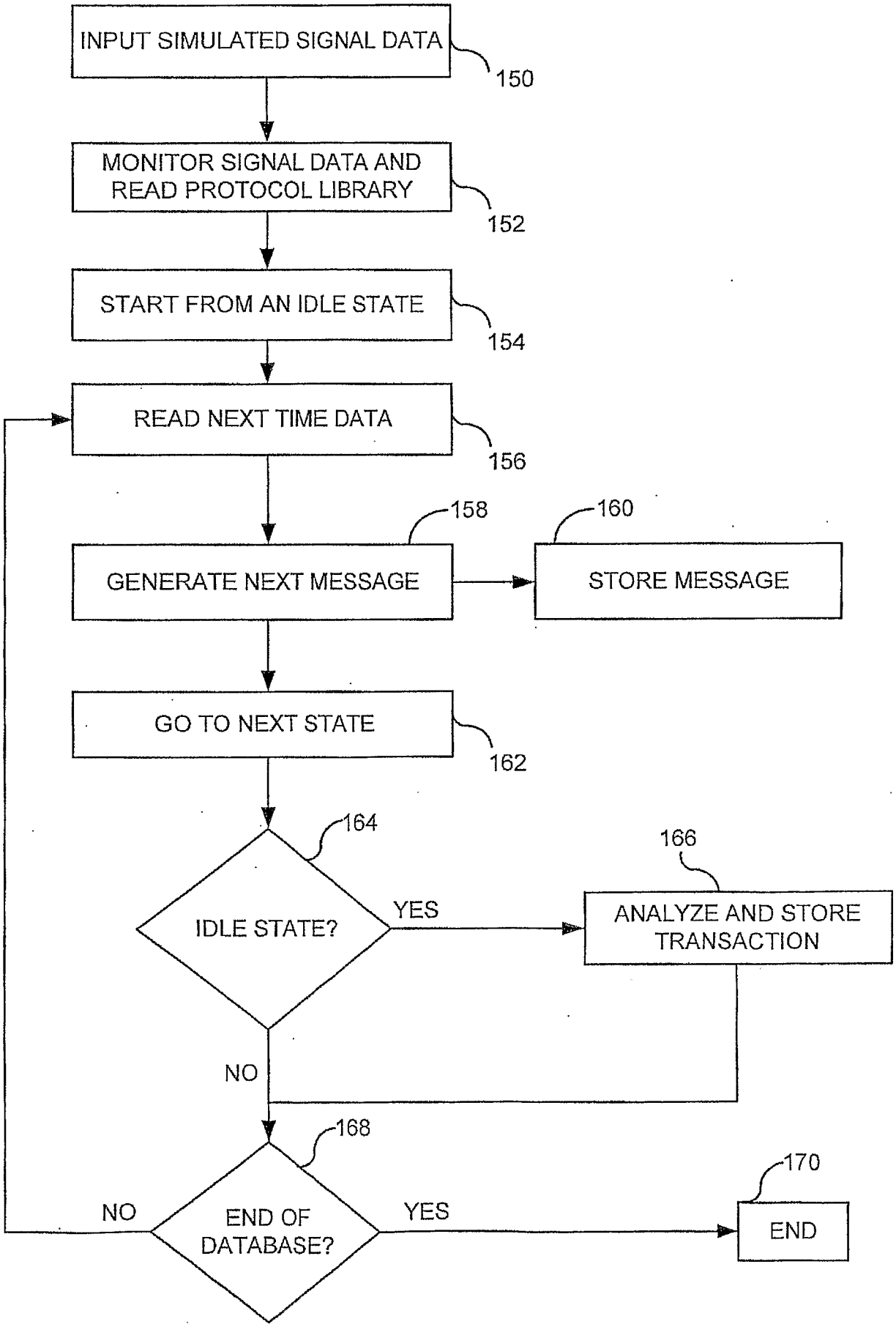

INPUT SIMULATED SIGNAL DATA
150
MONITOR SIGNAL DATA AND READ PROTOCOL LIBRARY
152
START FROM AN IDLE STATE
154
READ NEXT TIME DATA
156
158
GENERATE NEXT MESSAGE
160
STORE MESSAGE
GO TO NEXT STATE
162
164
IDLE STATE?
YES
166
ANALYZE AND STORE TRANSACTION
NO
168
END OF DATABASE?
NO
YES
170
END

FIGURE 7

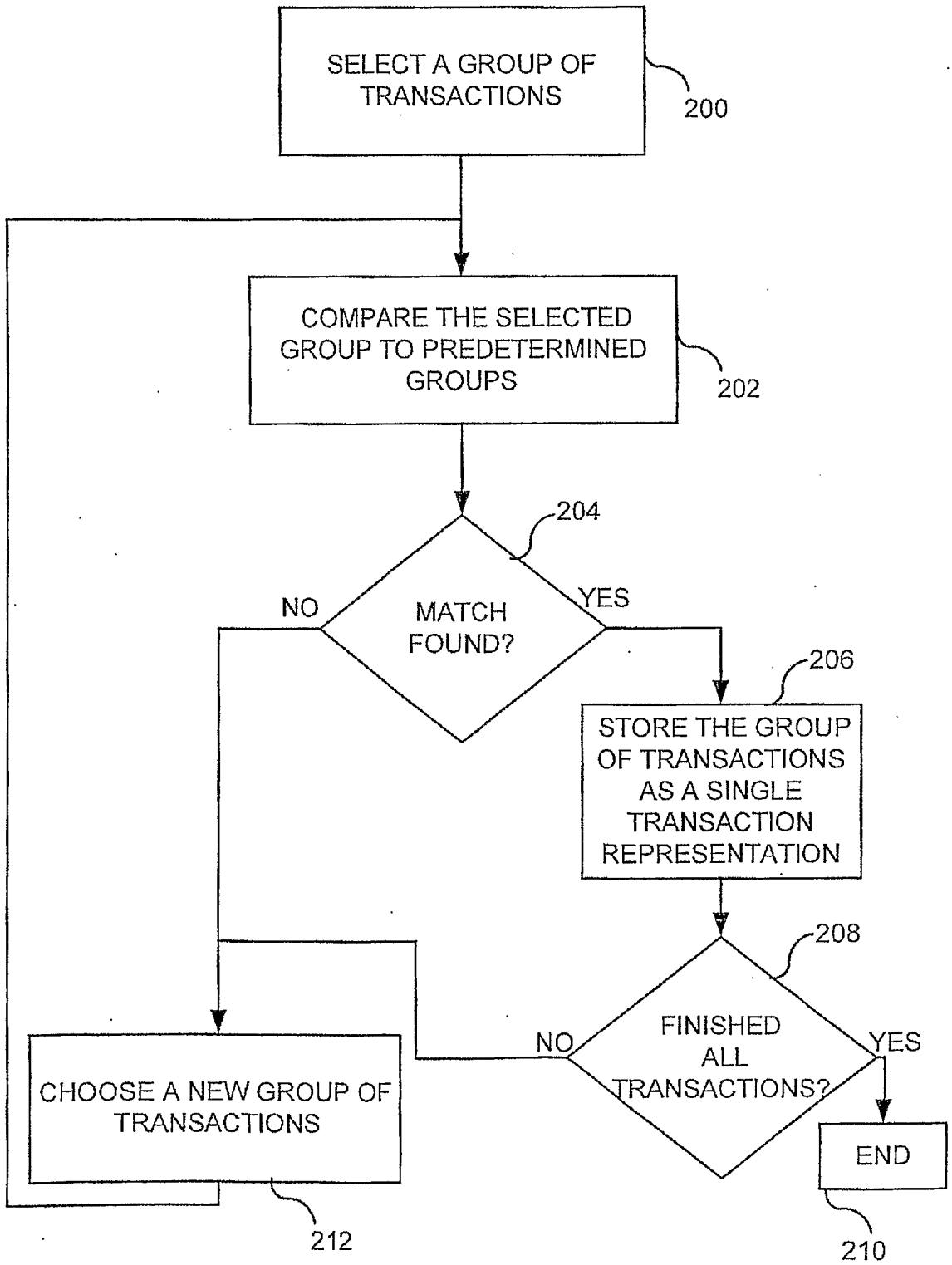

SELECT A GROUP OF TRANSACTIONS
200
COMPARE THE SELECTED GROUP TO PREDETERMINED GROUPS
202
204
MATCH FOUND?
NO
YES
206
STORE THE GROUP OF TRANSACTIONS AS A SINGLE TRANSACTION REPRESENTATION
208
FINISHED ALL TRANSACTIONS?
NO
YES
CHOOSE A NEW GROUP OF TRANSACTIONS
212
END
210

FIGURE 8

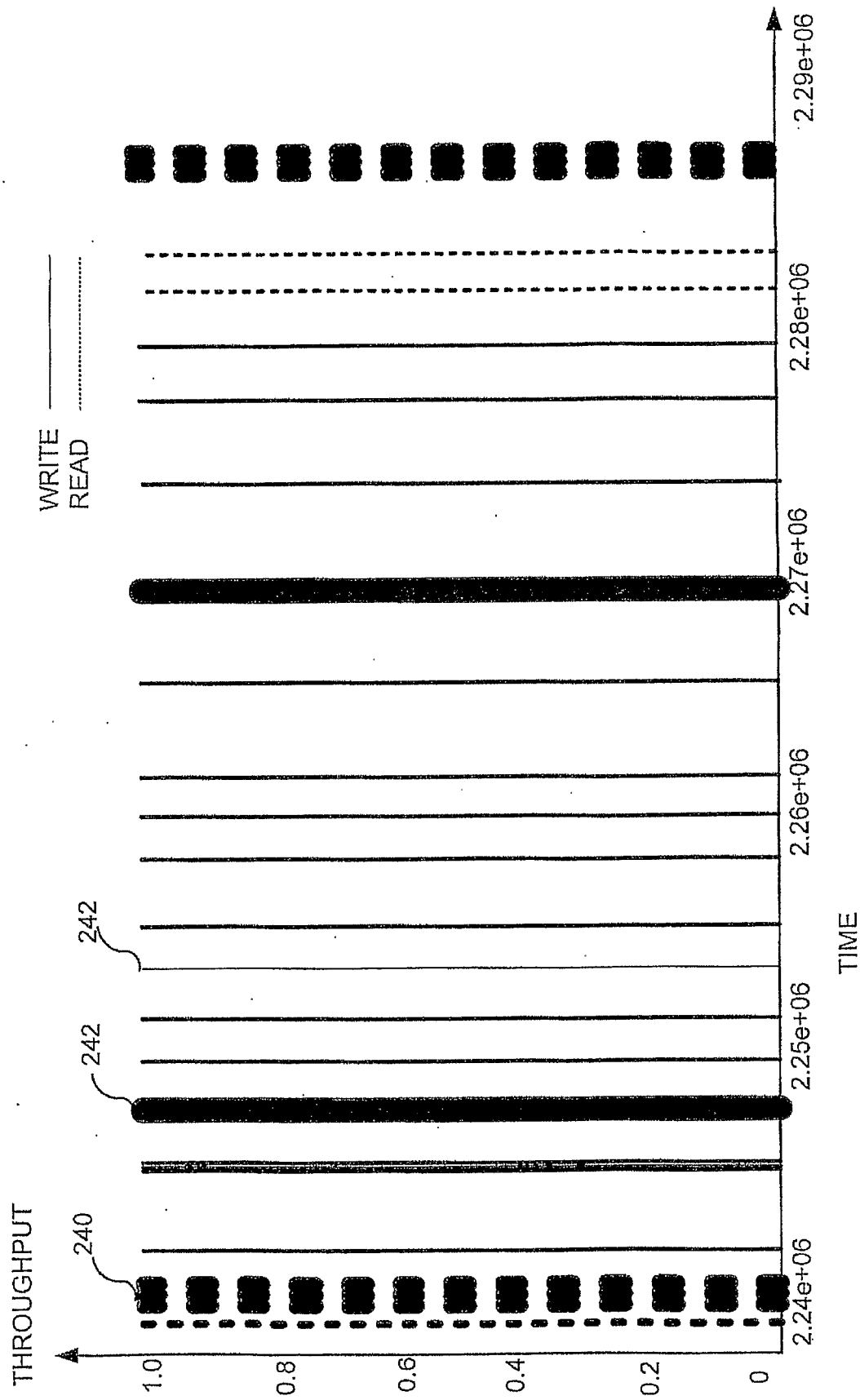

THROUGHPUT
1.0
0.8
0.6
0.4
0.2
0
240
242
242
WRITE
READ
2.24e+06
2.25e+06
2.26e+06
2.27e+06
2.28e+06
2.29e+06
TIME

FIGURE 9

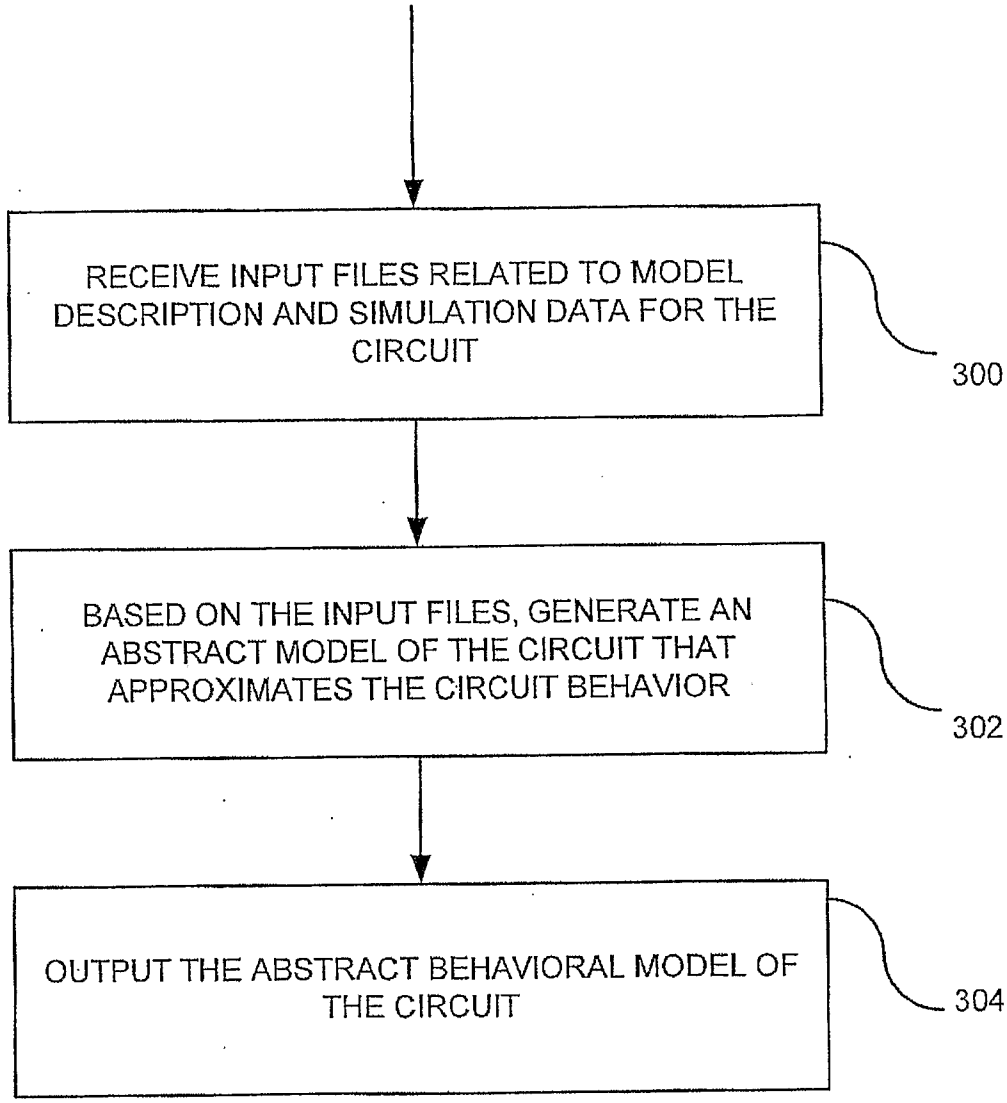

RECEIVE INPUT FILES RELATED TO MODEL DESCRIPTION AND SIMULATION DATA FOR THE CIRCUIT
300
BASED ON THE INPUT FILES, GENERATE AN ABSTRACT MODEL OF THE CIRCUIT THAT APPROXIMATES THE CIRCUIT BEHAVIOR
302
OUTPUT THE ABSTRACT BEHAVIORAL MODEL OF THE CIRCUIT
304

FIGURE 10

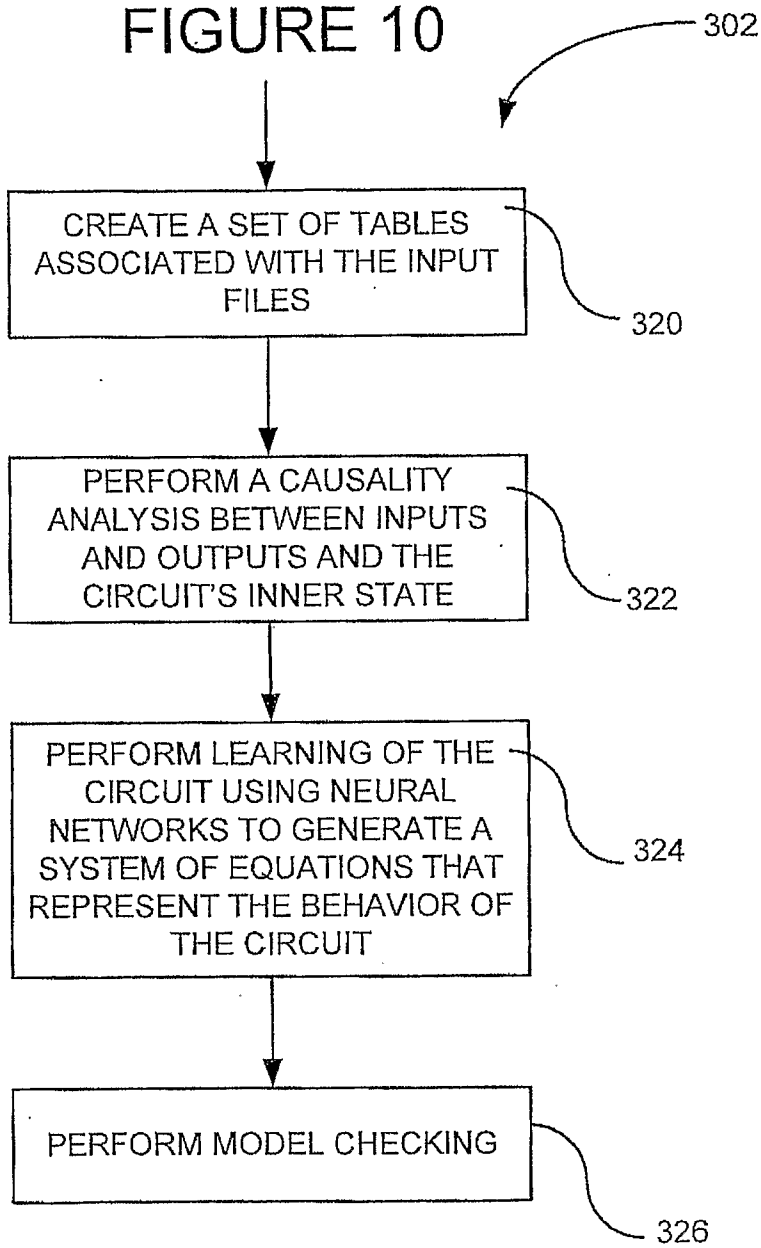
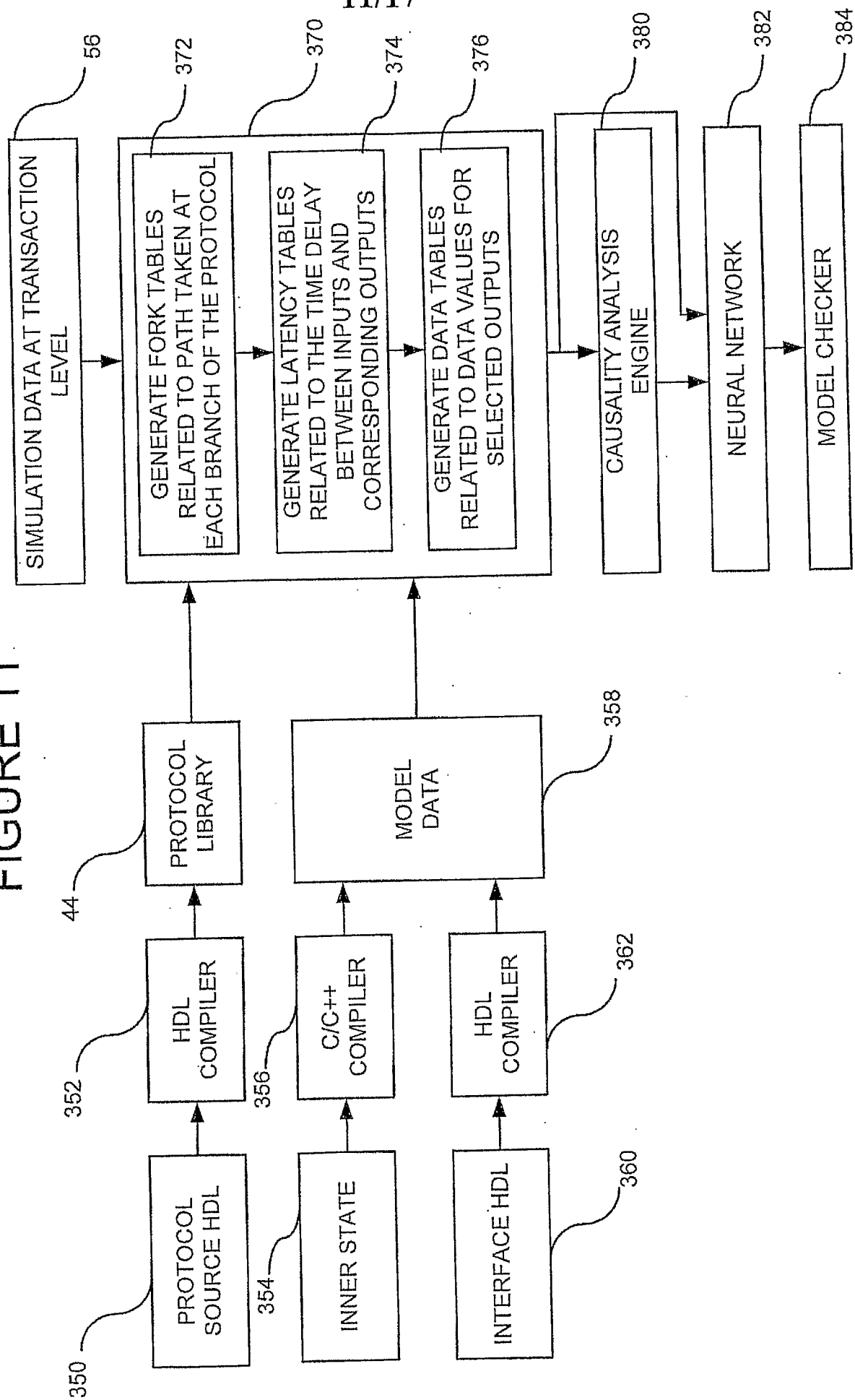

302
CREATE A SET OF TABLES ASSOCIATED WITH THE INPUT FILES
320
PERFORM A CAUSALITY ANALYSIS BETWEEN INPUTS AND OUTPUTS AND THE CIRCUIT'S INNER STATE
322
PERFORM LEARNING OF THE CIRCUIT USING NEURAL NETWORKS TO GENERATE A SYSTEM OF EQUATIONS THAT REPRESENT THE BEHAVIOR OF THE CIRCUIT
324
PERFORM MODEL CHECKING
326

FIGURE 11


SIMULATION DATA AT TRANSACTION LEVEL
56
GENERATE FORK TABLES RELATED TO PATH TAKEN AT EACH BRANCH OF THE PROTOCOL
372
GENERATE LATENCY TABLES RELATED TO THE TIME DELAY BETWEEN INPUTS AND CORRESPONDING OUTPUTS
370
374
GENERATE DATA TABLES RELATED TO DATA VALUES FOR SELECTED OUTPUTS
376
CAUSALITY ANALYSIS ENGINE
380
NEURAL NETWORK
382
MODEL CHECKER
384
PROTOCOL LIBRARY
44
MODEL DATA
358
HDL COMPILER
352
C/C++ COMPILER
356
HDL COMPILER
362
PROTOCOL SOURCE HDL
350
INNER STATE
354
INTERFACE HDL
360

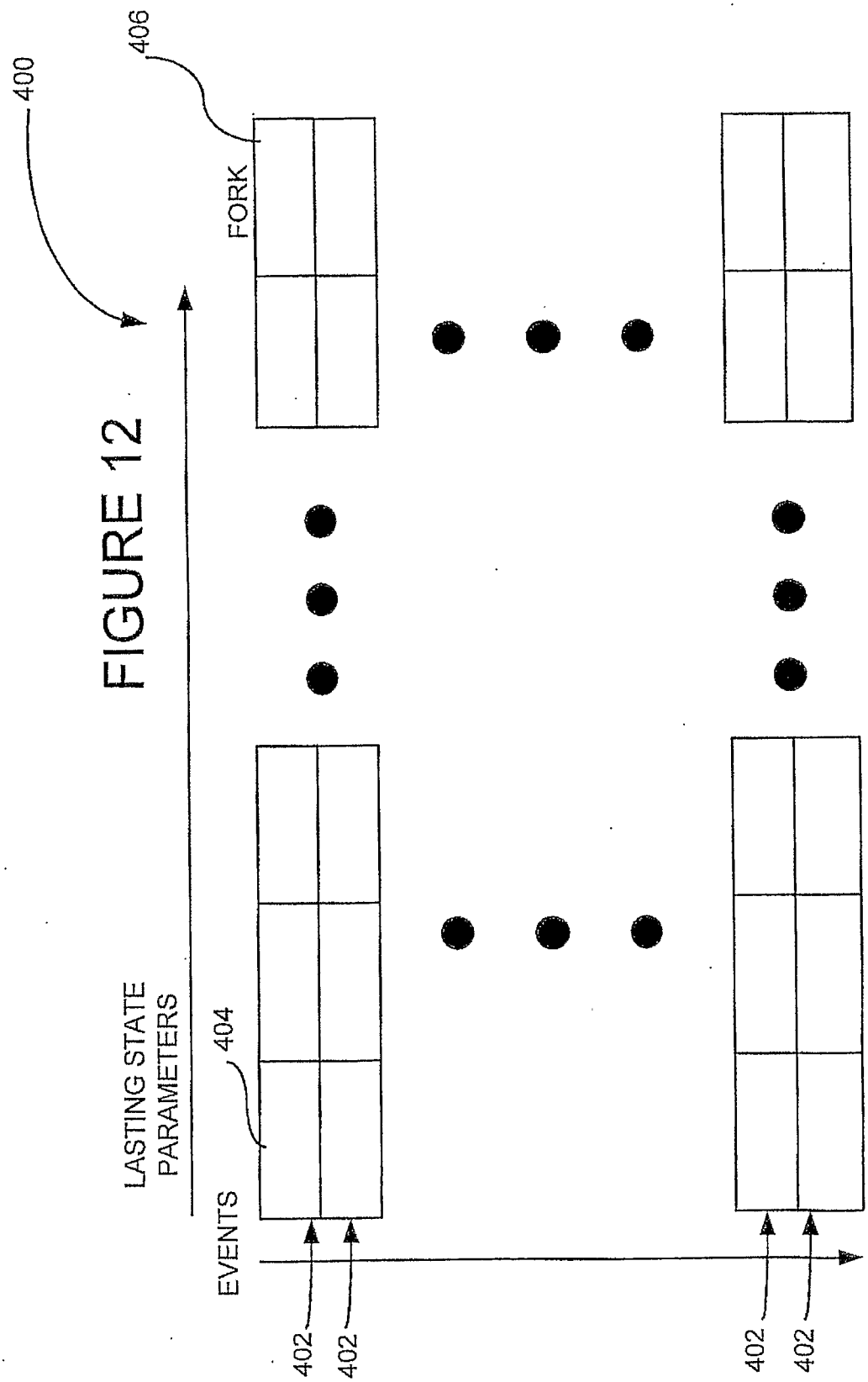

400
406
FORK
LASTING STATE PARAMETERS
404
EVENTS
402
402
402
402

FIGURE 12

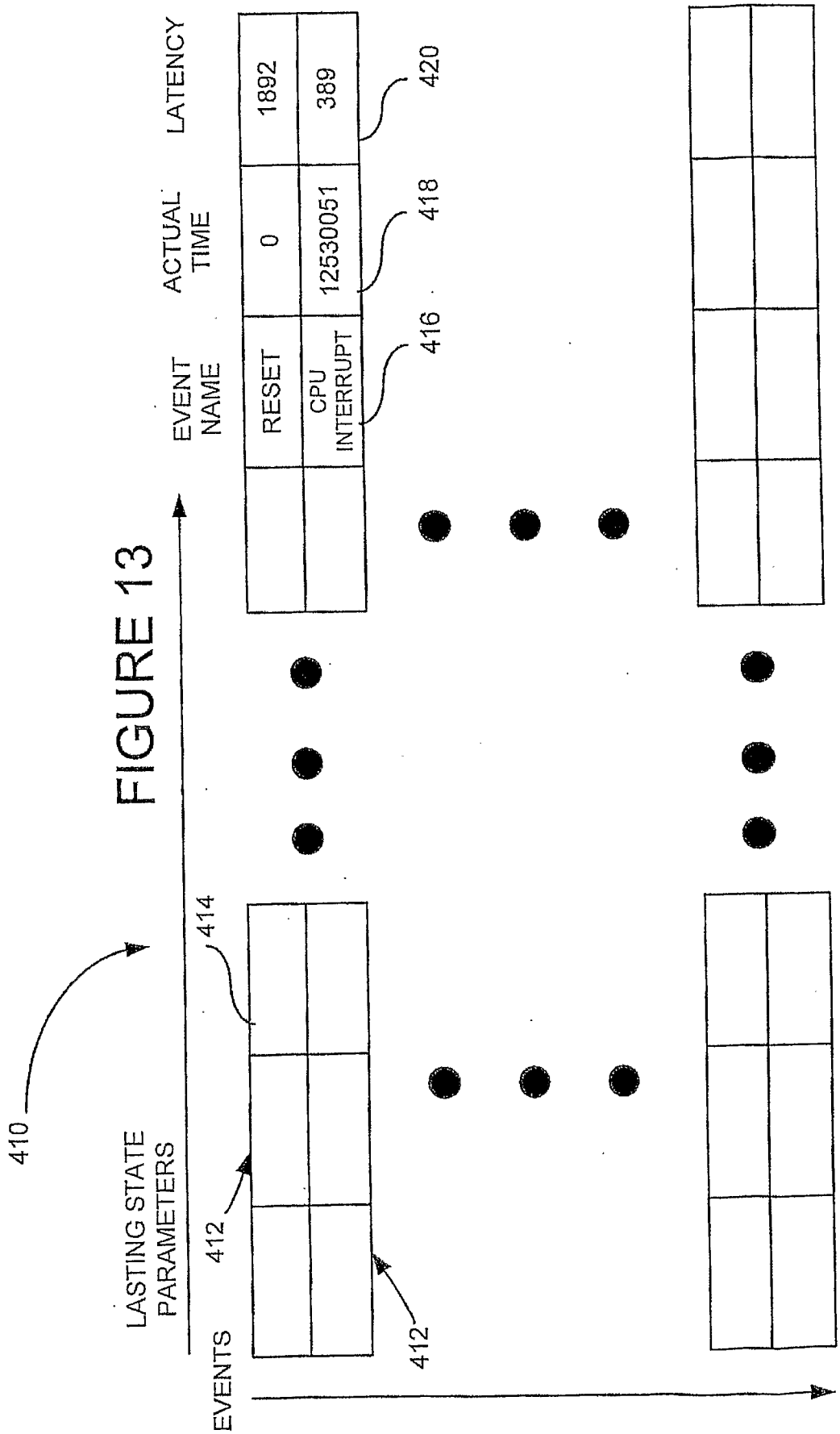

410
LASTING STATE PARAMETERS
EVENTS
412
412
414
EVENT NAME
ACTUAL TIME
LATENCY
RESET
CPU INTERRUPT
0
12530051
1892
389
416
418
420

FIGURE 13

FIGURE 14

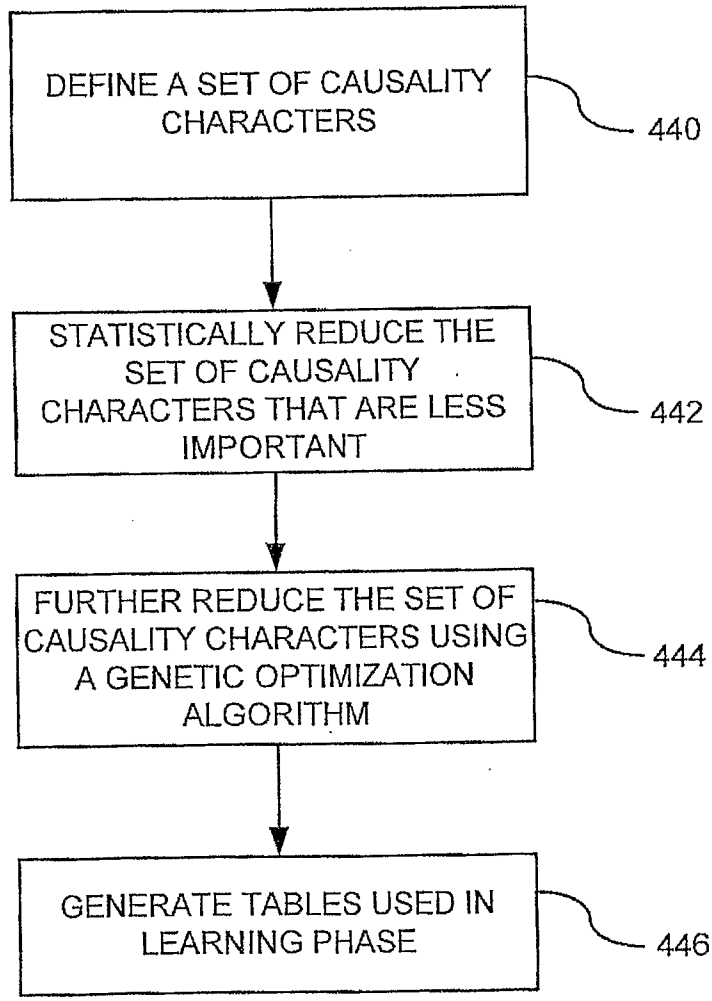

DEFINE A SET OF CAUSALITY CHARACTERS
440
STATISTICALLY REDUCE THE SET OF CAUSALITY CHARACTERS THAT ARE LESS IMPORTANT
442
FURTHER REDUCE THE SET OF CAUSALITY CHARACTERS USING A GENETIC OPTIMIZATION ALGORITHM
444
GENERATE TABLES USED IN LEARNING PHASE
446

FIGURE 15

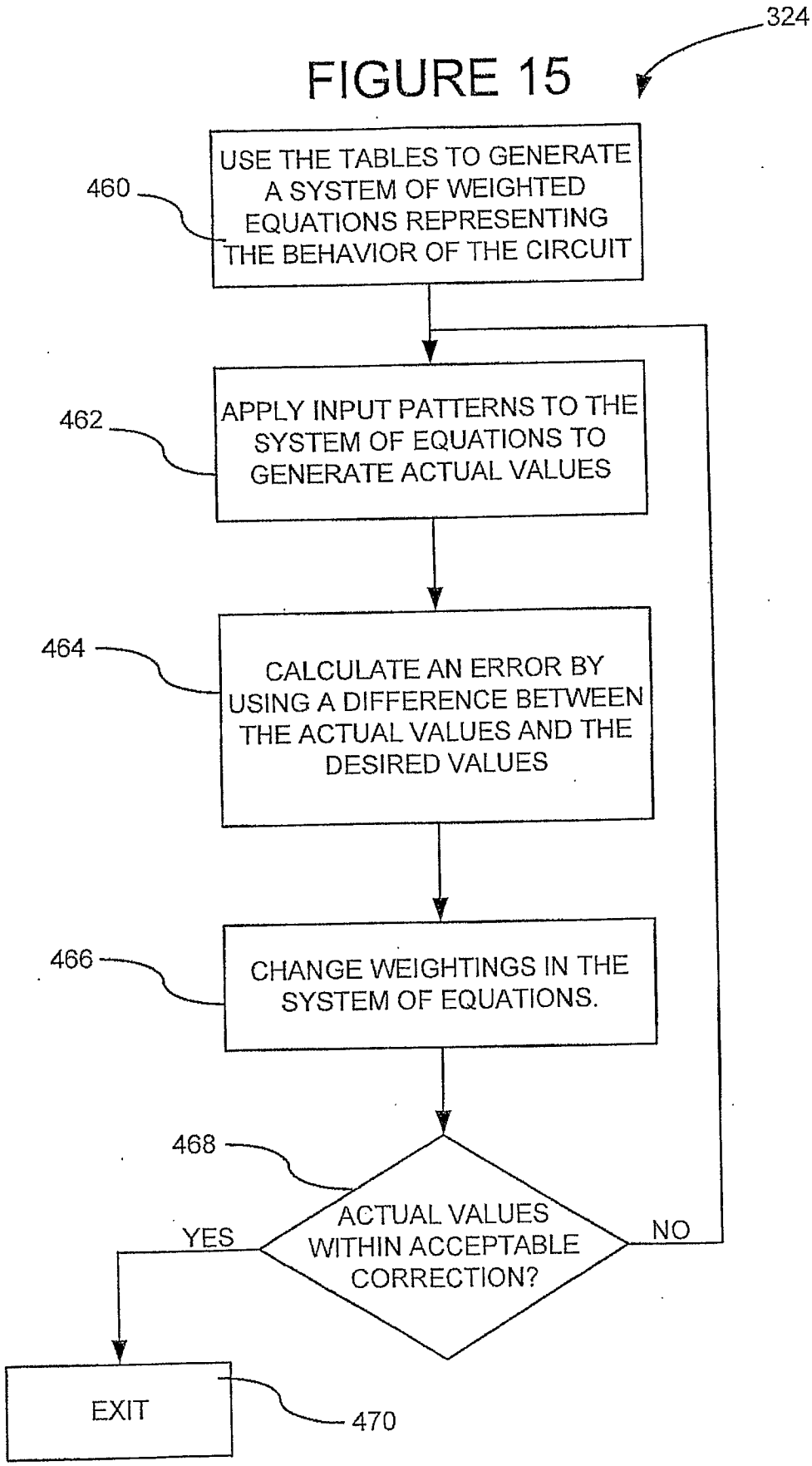

324
460
USE THE TABLES TO GENERATE A SYSTEM OF WEIGHTED EQUATIONS REPRESENTING THE BEHAVIOR OF THE CIRCUIT
462
APPLY INPUT PATTERNS TO THE SYSTEM OF EQUATIONS TO GENERATE ACTUAL VALUES
464
CALCULATE AN ERROR BY USING A DIFFERENCE BETWEEN THE ACTUAL VALUES AND THE DESIRED VALUES
466
CHANGE WEIGHTINGS IN THE SYSTEM OF EQUATIONS.
468
ACTUAL VALUES WITHIN ACCEPTABLE CORRECTION?
YES
NO
EXIT
470

FIGURE 16

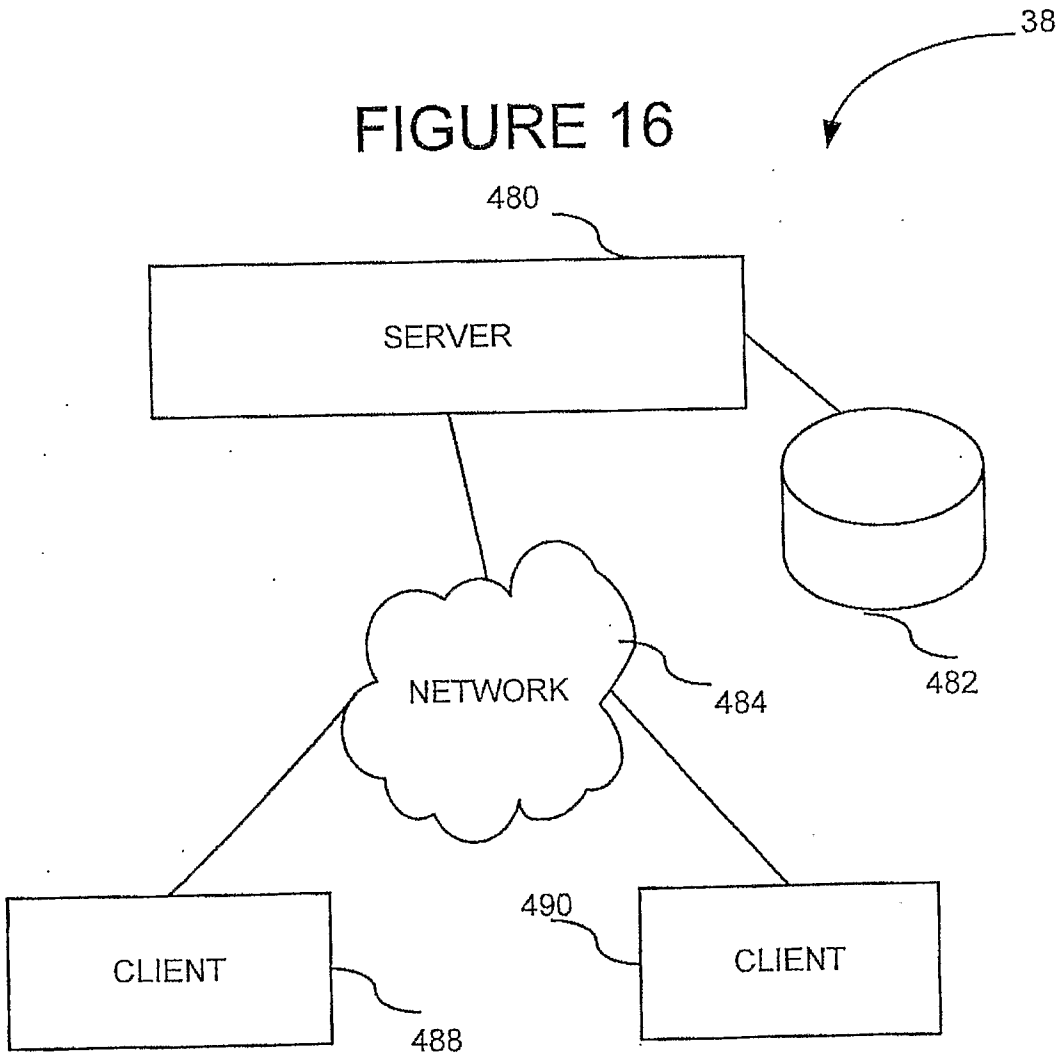
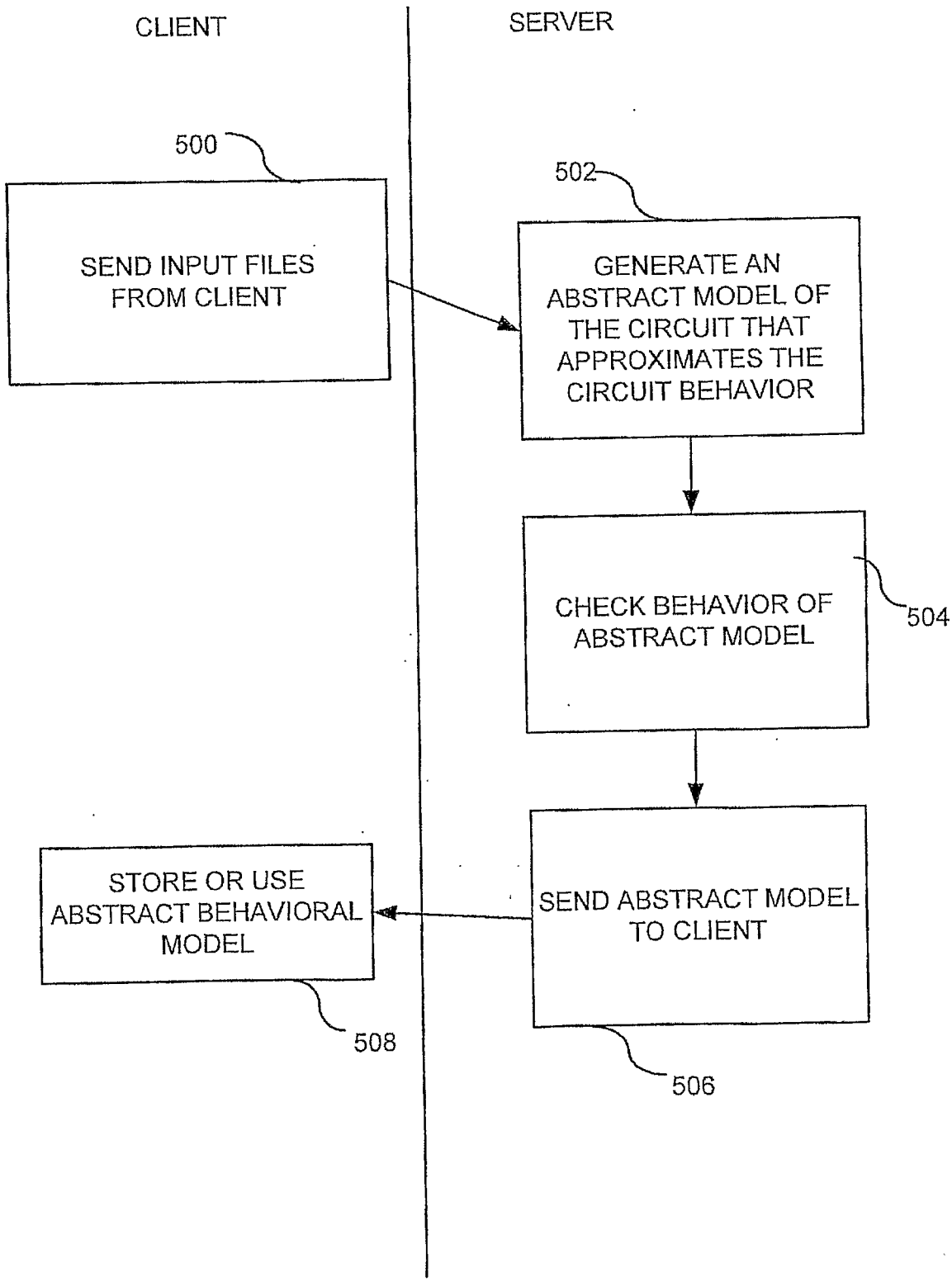


FIGURE 17


CLIENT
SERVER
500
SEND INPUT FILES FROM CLIENT
502
GENERATE AN ABSTRACT MODEL OF THE CIRCUIT THAT APPROXIMATES THE CIRCUIT BEHAVIOR
CHECK BEHAVIOR OF ABSTRACT MODEL
504
SEND ABSTRACT MODEL TO CLIENT
506
STORE OR USE ABSTRACT BEHAVIORAL MODEL
508

INTERNATIONAL SEARCH REPORT

International application No
PCT/IL2006/001349

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/50

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2005/131666 A1 (TSAI JIEN-SHEN [TW] ET AL) 16 June 2005 (2005-06-16) abstract figures 1-4 paragraph [0016] - paragraph [0040]	1-23
X	JINDAL R ET AL: "Verification of transaction-level systemc models using RTL testbenches" FORMAL METHODS AND MODELS FOR CO-DESIGN, 2003. MEMOCODE '03. PROCEEDINGS. FIRST ACM AND IEEE INTERNATIONAL CONFERENCE ON 24-26 JUNE 2003, PISCATAWAY, NJ, USA,IEEE, 24 June 2003 (2003-06-24), pages 199-203, XP010643815 ISBN: 0-7695-1923-7 abstract page 200 - page 202	1-23

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 April 2007	24/04/2007

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	ALONSO NOGUEIRO, L

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/IL2006/001349

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6 263 301 B1 (COX STEVEN G [US] ET AL) 17 July 2001 (2001-07-17) abstract figures 2-8 column 2, line 20 - column 8, line 67	1-23
X	WO 02/08966 A (TELECOM ITALIA LAB SPA [IT]; BOLLANO GIANMARIO [IT]; ETTORRE DONATO [I) 31 January 2002 (2002-01-31) abstract page 6, line 1 - line 35 page 9, line 1 - page 10, line 35 page 15, line 1 - page 17, line 35	1-23
A	BERNSTEIN A ET AL: "How to bridge the abstraction gap in system level modeling and design" COMPUTER AIDED DESIGN, 2004. ICCAD-2004. IEEE/ACM INTERNATIONAL CONFERENCE ON SAN JOSE, CA, USA NOV. 7-11, 2004, PISCATAWAY, NJ, USA,IEEE, 7 November 2004 (2004-11-07), pages 910-914, XP010763562 ISBN: 0-7803-8702-3 the whole document	1-23

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/IL2006/001349

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005131666 A1	16-06-2005	NONE	
US 6263301 B1	17-07-2001	NONE	
WO 0208966 A	31-01-2002	AU 7767801 A	05-02-2002
		EP 1301875 A2	16-04-2003
		US 2003154465 A1	14-08-2003