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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 November 2007 (15.11.2007)

PCT

(10) International Publication Number
WO 2007/128753 A1

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

(21) International Application Number:
PCT/EP2007/054255

(22) International Filing Date: 2 May 2007 (02.05.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/381,437 3 May 2006 (03.05.2006) US

(71) Applicant *(for all designated States except US)*: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; Armonk, New York 10504 (US).

(71) Applicant *(for MG only)*: **IBM UNITED KINGDOM LIMITED** [GB/GB]; Portsmouth Hampshire PO6 3AU (GB).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BOBOK, Gabor** [HU/US]; 1345 Hawthorn Road, Niskayuna, New York 12309 (US). **ROESNER, Wolfgang** [DE/US]; 10717 Chestnut Ridge Road, Austin, Texas 78726 (US). **WILLIAMS, Derek, Edward** [US/US]; 9406 Slate Creek Trail, Austin, Texas 78717 (US).

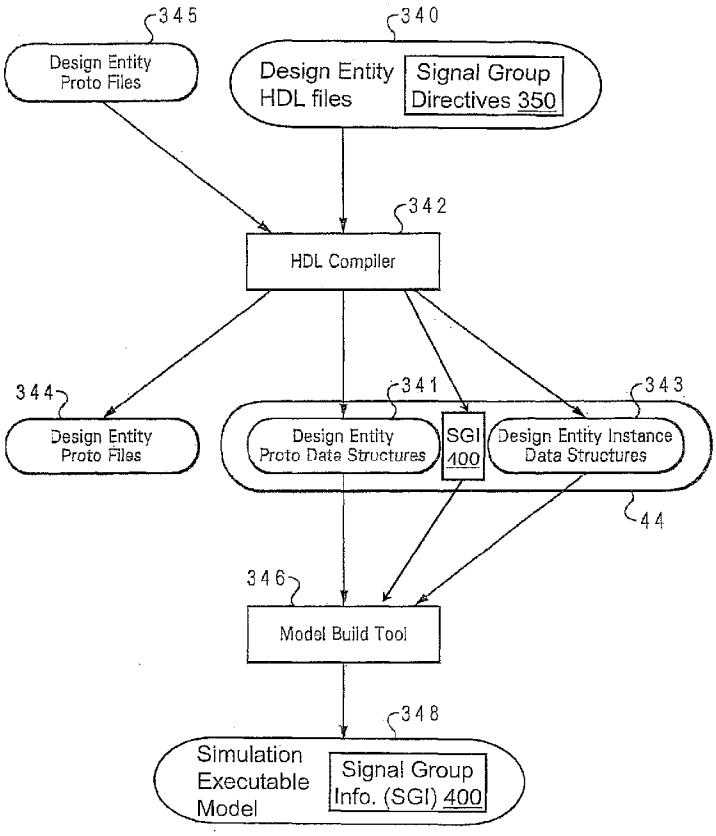
(74) Agent: **WALDNER, Philip**; IBM United Kingdom Limited, Intellectual Property Law, Winchester Hampshire SO21 2JN (GB).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, 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, 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,

[Continued on next page]

(54) Title: METHOD, SYSTEM AND PROGRAM PRODUCT SUPPORTING SPECIFICATION OF SIGNALS FOR SIMULATION RESULT VIEWING



(57) Abstract: According to a method of data processing, a data set including at least one entry specifying a signal group by a predetermined signal group name is received by a data processing system. In response to receipt of the data set, the entry in the data set is processed to identify the signal group name. Signal group information associated with an event trace file containing simulation results is accessed to determine signal names of multiple signals that are members of the signal group. Simulation results from the event trace file that are associated with instances of said multiple signals are then included within a presentation of simulation results.

WO 2007/128753 A1

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, MT, 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).

Published:

— *with international search report*

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.

METHOD, SYSTEM AND PROGRAM PRODUCT SUPPORTING SPECIFICATION OF SIGNALS FOR SIMULATION RESULT VIEWING

BACKGROUND OF THE INVENTION

Technical Field

The present invention relates in general to simulating digital devices, modules and systems, and in particular, to computer simulation of digital devices, modules and systems.

Description of the Related Art

Verifying the logical correctness of a digital design and debugging the design, if necessary, are very important steps in most digital design processes. Logic networks are tested either by actually building networks or by simulating networks on a computer. As logic networks become highly complex, it becomes necessary to simulate a design before the design is actually built. This is especially true when the design is implemented as an integrated circuit, since the fabrication of integrated circuits requires considerable time and correction of mistakes is quite costly. The goal of digital design simulation is the verification of the logical correctness of the design.

In a typical automated design process that is supported by a conventional electronic computer-aided design (ECAD) system, a designer enters a high-level description utilizing a hardware description language (HDL), such as VHDL, producing a representation of the various circuit blocks and their interconnections. The ECAD system compiles the design description into a format that is best suited for simulation. A simulator is then utilized to verify the logical correctness of the design prior to developing a circuit layout.

A simulator is typically a software tool that operates on a digital representation, or simulation model of a circuit, and a list of input stimuli (i.e., testcase) representing inputs of the digital system. A simulator generates a numerical representation of the response of the circuit, which may then either be viewed on the display screen as a list of values or further interpreted, often by a separate software program, and presented on the display screen in graphical form. The simulator may be run either on a general-purpose computer or on another piece of electronic apparatus, typically attached to a general purpose computer,

specially designed for simulation. Simulators that run entirely in software on a general-purpose computer will hereinafter be referred to as "software simulators". Simulators that are run with the assistance of specially designed electronic apparatus will hereinafter be referred to as "hardware simulators".

Usually, software simulators perform a very large number of calculations and operate slowly from the user's point of view. In order to optimize performance, the format of the simulation model is designed for very efficient use by the simulator. Hardware simulators, by nature, require that the simulation model comprising the circuit description be communicated in a specially designed format. In either case, a translation from an HDL description to a simulation format, hereinafter referred to as a simulation executable model, is required.

The result of the application of the testcase to the simulation executable model by the simulator is referred to herein as an "all events trace" (AET). The AET contains the logic values of signals and/or storage elements within the simulation executable model. An AET viewer can be utilized to present by the contents of the AET to the user for review and analysis.

As will be appreciated, for large simulation executable models, a vast amount of data will be present in the AET, not all of which will be relevant to the user. Accordingly, conventional AET viewers permit a user to input an Input/Output (I/O) list specifying signals in the simulation executable model that the user desires to view. In response, a conventional AET viewer presents to the user only those signals within the simulation executable model that are identified within the I/O list.

SUMMARY OF THE INVENTION

The present invention recognizes that user entry of the I/O list (e.g., utilizing a keyboard) is tedious and time consuming, particularly for complex simulation executable models. Accordingly, the present invention provides to a method, system and program product for simulation processing.

According to an exemplary method, a data set including at least one entry specifying a signal group by a predetermined signal group name is received by a data processing system. In

response to receipt of the data set, the entry in the data set is processed to identify the signal group name. Signal group information associated with an event trace file containing simulation results is accessed to determine signal names of multiple signals that are members of the signal group. Simulation results from the event trace file that are associated with instances of said multiple signals are then included within a presentation of simulation results.

All objects, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

FIG. 1 is a pictorial representation of a data processing system in accordance with the present invention;

FIG. 2 depicts a representative hardware environment of the data processing system illustrated in **FIG. 1**;

FIG. 3A is a simplified block diagram illustrating a digital design entity in accordance with the teachings of the present invention;

FIG. 3B is a diagrammatic representation depicting a simulation model in accordance with the teachings of the present invention;

FIG. 3C is a flow diagram illustrating of a model build process in accordance with the teachings of the present invention;

FIG. 3D is a block diagram depicting simulation model data structures representing a design in accordance with the teachings of the present invention;

FIG. 4 is a flow diagram depicting simulation of a simulation executable model and presenting the results of simulation to a user;

FIG. 5A depicts a first conventional I/O list in accordance with the prior art;

FIG. 5B depicts a second conventional I/O list in accordance with the prior art;

FIGS. 5C-5D illustrate exemplary I/O lists in accordance with the present invention;

FIG. 6A depicts an exemplary design entity HDL file including signal group descriptors in accordance with the present invention;

FIG. 6B illustrates an exemplary design entity HDL file including nested signal group descriptors in accordance with the present invention; and

FIGS. 7A-7C together form a high-level logical flowchart of an exemplary process by which an AET viewer processes an I/O list to generate a presentation of an AET file in accordance with the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

With reference now to the figures, and in particular with reference to **FIG. 1**, there is depicted a pictorial representation of a data processing system **10** with which the present invention may be advantageously utilized. As illustrated, data processing system **10** comprises a workstation **12** to which one or more nodes **13** are connected. Workstation **12** preferably comprises a high performance multiprocessor computer, such as one of the POWER line of computer systems available from International Business Machines (IBM) Corporation of Armonk, New York. Workstation **12** preferably includes nonvolatile and

volatile internal storage for storing software applications comprising an ECAD system, which can be utilized to develop and verify a digital circuit design in accordance with the method and system of the present invention. As depicted, nodes **13** include a display device **14**, a keyboard **16**, and a mouse **20**. The ECAD software applications executed within workstation **12** preferably display a graphic user interface (GUI) within display screen **22** of display device **14** with which a digital circuit designer can interact using a keyboard **16** and mouse **20**. Thus, by entering appropriate inputs utilizing keyboard **16** and mouse **20**, the digital circuit designer is able to develop and verify a digital circuit design according to the method described further hereinbelow.

FIG. 2 is a more detailed block diagram of data processing system **10**. As illustrated, data processing system **10** includes one or more Central Processing Units (CPUs) **24**, such as a conventional microprocessor, and a number of other components interconnected via system interconnect **26**. Although not depicted in **FIG. 2**, CPUs such as CPU **24** typically include a control unit that organizes data and program storage in a computer memory and transfers the data and other information between the various parts of the computer system. CPUs also generally include one or more arithmetic logic units that execute arithmetical and logical operations, such as addition, comparison, multiplication and so forth.

Data processing system **10** further includes a random-access memory (RAM) **28**, a read-only memory (ROM) **30**, a display adapter **32** supporting connection of a display device **14**, and an I/O adapter **34** for connecting peripheral devices (e.g., disk and tape drives **33**). Data processing system **10** further includes a communications adapter **42** for connecting data processing system **10** to a communications network and a user interface adapter **36** for connecting keyboard **16**, mouse **20**, speaker **38**, microphone **40**, and/or other user interface devices to system interconnect **26**.

As will be appreciated by those skilled in the art, data processing system **10** operates under the control of an operating system (e.g., AIX) and one or more other programs, which may reside in any suitable computer-readable media such as RAM **28**, ROM **30**, a magnetic disk, magnetic tape, or optical disk (the last three being located in disk and tape drives **33**).

Simulated digital circuit design models are comprised of at least one and usually many sub-units referred to hereinafter as design entities. **FIG. 3A** is a block diagram representation of an exemplary design entity **300** in which the method and system of the present invention may be implemented. Design entity **300** is defined by a number of components: an entity name, entity ports, and a representation of the function performed by design entity **300**. Each entity within a given model has a unique name (not explicitly shown in **FIG. 3A**) that is declared in the HDL description of each entity. Furthermore, each entity typically contains a number of signal interconnections, known as ports, to signals outside the entity. These outside signals may be primary input/outputs (I/Os) of an overall design or signals connecting to other entities within an overall design.

Typically, ports are categorized as belonging to one of three distinct types: input ports, output ports, and bi-directional ports. Design entity **300** is depicted in as having a number of input ports **303** that convey signals into design entity **300**. Input ports **303** are connected to input signals **301**. In addition, design entity **300** includes a number of output ports **306** that convey signals out of design entity **300**. Output ports **306** are connected to a set of output signals **304**. Bi-directional ports **305** are utilized to convey signals into and out of design entity **300**. Bi-directional ports **305** are in turn connected to a set of bi-directional signals **309**. An entity, such as design entity **300**, need not contain ports of all three types, and in the degenerate case, contains no ports at all. To accomplish the connection of entity ports to external signals, a mapping technique, known as a "port map", is utilized. A port map (not explicitly depicted in **FIG. 3A**) consists of a specified correspondence between entity port names and external signals to which the entity is connected. When building a simulation model, ECAD software is utilized to connect external signals to appropriate ports of the entity according to a port map specification.

Finally, design entity **300** contains a body section **308** that describes one or more functions performed by design entity **300**. In the case of a digital design, body section **308** contains an interconnection of logic gates, storage elements, etc., in addition to instantiations of other entities. By instantiating an entity within another entity, a hierarchical description of an overall design is achieved. For example, a microprocessor may contain multiple instances of an identical functional unit. As such, the microprocessor itself will often be modeled as a

single entity. Within the microprocessor entity, multiple instantiations of any duplicated functional entities will be present.

Referring now to **FIG. 3B**, there is illustrated a diagrammatic representation of an exemplary simulation model **329** that may be utilized in a preferred embodiment of the present invention. Simulation model **329** includes multiple hierarchical design entities. For visual simplicity and clarity, many of the ports and signals interconnecting the entities within simulation model **329** have not been explicitly shown. In any model, one and only one entity is the so-called "top-level entity". A top-level entity **320** is that entity which encompasses all other entities within simulation model **329**. That is to say, top-level entity **320** instantiates, either directly or indirectly, all descendant entities within a design. Simulation model **329** consists of top-level entity **320** which directly instantiates two instances, **321a** and **321b,** of an FXU entity **321**. Each instantiation has an associated description, which contains an entity name and a unique instantiation name. For top-level entity **320**, description **310** is labeled "TOP:TOP". Description **310** includes an entity name **312**, labeled as the "TOP" preceding the colon, and also includes an instantiation name **314**, labeled as the "TOP" following the colon.

It is possible for a particular entity to be instantiated multiple times as is depicted with instantiations **321a** and **321b** of FXU entity **321**. Instantiations **321a** and **321b** are distinct instantiations of FXU entity **321** with instantiation names FXU0 and FXU1, respectively. Top-level entity **320** is at the highest level within the hierarchy of simulation model **329**. An entity that instantiates a descendant entity will be referred to hereinafter as an "ancestor" of the descendant entity. Top-level entity **320** is therefore the ancestor that directly instantiates FXU entity instantiations **321a** and **321b**. At any given level of a simulation model hierarchy, the instantiation names of all instantiations must be unique.

Within instantiation **321a** of FXU entity **321**, single instance entities **325a** and **326a** of entity A **325** and entity B **326** respectively, are directly instantiated. Similarly, instantiation **321b** of the same FXU entity contains instantiations **325b** and **326b** of entity A **325** and entity B **326** respectively. In a similar manner, instantiation **326a** and instantiation **326b** each directly instantiate a single instance of entity C **327** as entities **327a** and **327b**, respectively.

The nesting of entities within other entities can continue to an arbitrary level of complexity provided that all entities instantiated, whether singly or multiply, have unique entity names and the instantiation names at any given level of the hierarchy are unique with respect to one another. Each entity is constructed from one or more HDL files that contain the information necessary to describe the entity.

Associated with each entity instantiation is a so called "instantiation identifier". The instantiation identifier for a given instantiation is a string consisting of the enclosing entity instantiation names proceeding from the top-level entity instantiation name. For example, the instantiation identifier of instantiation **327a** of entity C **327** within instantiation **321a** of FXU entity **321** is "TOP.FXU0.B.C". This identifier serves to uniquely identify each instantiation within a simulation model.

Within exemplary simulation model **329**, a variety of signals are instantiated (e.g., signals E, F0, F1, G, H0, H1, L, M, N, P and Q). Each signal has an associated signal name (e.g., "M") and a signal instantiation identifier, which in a preferred embodiment, is a string consisting of the enclosing entity instantiation names proceeding from the top-level entity instantiation name and terminating with the signal name. Thus, the instantiation identifier of signal M within instantiation **321a** of FXU entity **321** is "TOP.FXU0.A.M". This instantiation identifier serves to uniquely identify each signal instantiation within a simulation model. It should be noted that signals, for example, signal P(0..4), can be multi-bit signal vectors. It should also be noted that some signals (e.g., signals TOP.FXU0.E, TOP.FXU1.E, TOP.FXU0.G and TOP.FXU1.G) are renamed (as signals TOP.FXU0.F0, TOP.FXU1.F1, TOP.FXU0.H0 and TOP.FXU1.H1, respectively) as they cross design entity boundaries.

Referring now to **FIG. 3C**, there is depicted a flow diagram of a model build process which may be implemented in a preferred embodiment of the present invention. The process begins with one or more design entity HDL source code files **340** and, potentially, one or more design entity intermediate format files **345**, hereinafter referred to as "proto files" **345**, available from a previous run of an HDL compiler **342**. HDL compiler **342** processes HDL file(s) **340** beginning with the top level entity of a simulation model and proceeding in a recursive fashion through all HDL or proto file(s) describing a complete simulation model.

For each of HDL files **340** during the compilation process, HDL compiler **342** examines proto files **345** to determine if a previously compiled proto file is available and consistent. If such a file is available and consistent, HDL compiler **342** will not recompile that particular file, but will rather refer to an extant proto file. If no such proto file is available or the proto file is not consistent, HDL compiler **342** explicitly recompiles the HDL file **340** in question and creates a proto file **344** for use in subsequent compilations. Such a process will be referred to hereinafter as "incremental compilation" and can greatly speed the process of creating a simulation executable model **348**. Once created by HDL compiler **342**, proto files **344** are available to serve as proto files **345** in subsequent compilations.

In addition to proto files **344**, HDL compiler **342** also creates two sets of data structures, design entity proto data structures **341** and design entity instance data structures **343**, in memory **44** of computer system **10**. Design entity proto data structures **341** and design entity instance data structures **343**, serve as a memory image of the contents of a simulation executable model **348**. Data structures **341** and **343** are passed, via memory **44**, to a model build tool **346** that processes data structures **341** and **343** into simulation executable model **348**.

It will be assumed hereinafter that each entity is described by a single HDL file. Depending on convention or the particular HDL in which the current invention is practiced, this restriction may be required. However, in certain circumstances or for certain HDLs it is possible to describe an entity by utilizing more than one HDL file. Those skilled in the art will appreciate and understand the extensions necessary to practice the present invention if entities are permitted to be described by multiple HDL files. Furthermore, it will be assumed that there is a direct correspondence, for each entity, between the entity name and both of the following: the name of the HDL file representing the entity, and the name of the proto file for the entity.

In the following description, an HDL source code file corresponding to a given entity will be referred to by an entity name followed by ".vhdl". For example, the HDL source code file that describes top-level entity **320** will be referred to as TOP.vhdl. This labeling convention

serves as a notational convenience only and should not be construed as limiting the applicability of the present invention to HDLs other than VHDL.

Returning to **FIG. 3B**, it can be seen that each entity may instantiate, either directly or indirectly, one or more other entities. For example, the FXU entity directly instantiates A entity **325** and B entity **326**. Furthermore, B entity **326** directly instantiates C entity **327**. Therefore, FXU entity **321** instantiates, directly or indirectly, A entity **325**, B entity **326** and C entity **327**. Those entities, that are directly or indirectly instantiated by another entity, will be referred to hereinafter as "descendants". The descendants of top level entity **320** are FXU entity **321**, A entity **325**, B entity **326**, and C entity **327**. It can be seen that each entity has a unique set of descendants and that each time an entity is instantiated, a unique instance of the entity and its descendants is created. Within simulation model **329**, FXU entity **321** is instantiated twice, FXU:FXU0 **321a** and FXU:FXU1 **321b**, by top-level entity **320**. Each instantiation of FXU entity **321** creates a unique set of instances of the FXU, A, B, and C entities.

For each entity, it is possible to define what is referred to as a "bill-of-materials" or BOM. A BOM is a list of HDL files having date and time stamps of the entity itself and the entity's descendants. Referring again to **FIG. 3C**, the BOM for an entity is stored in proto file **344** after compilation of the entity. Therefore, when HDL compiler **342** compiles a particular HDL source code file among HDL files **340**, a proto file **344** is generated that includes a BOM listing the HDL files **340** that constitute the entity and the entity's descendants, if any. The BOM also contains the date and time stamp for each of the HDL files referenced as each appeared on disk/tape **33** of computer system **10** when the HDL file was being compiled.

If any of the HDL files constituting an entity or the entity's descendants is subsequently changed, proto file **344** will be flagged as inconsistent and HDL compiler **342** will recompile HDL file **340** on a subsequent re-compilation as will be described in further detail below. For example, going back to **FIG. 3B**, the HDL files referenced by the BOM of FXU entity **321** are FXU.vhdl, A.vhdl, B.vhdl and C.vhdl, each with appropriate date and time stamps. The files referenced by the BOM of top-level entity **320** are TOP.vhdl, FXU.vhdl, A.vhdl, B.vhdl, C.vhdl, and FPU.vhdl with appropriate date and time stamps.

Returning to **FIG. 3C**, HDL compiler **342** creates an image of the structure of a simulation model in main memory **44** of computer system **10**. This memory image is comprised of the following components: "proto" data structures **341** and "instance" data structures **343**. A proto is a data structure that, for each entity in the model, contains information about the ports of the entity, the body contents of the entity, and a list of references to other entities directly instantiated by the entity (in what follows, the term "proto" will be utilized to refer to the in-memory data structure described above and the term "proto file" will be utilized to describe intermediate format file(s) **344**). Proto files **344** are therefore on-disk representations of the in-memory proto data structure produced by HDL compiler **342**.

An instance data structure is a data structure that, for each instance of an entity within a model, contains the instance name for the instance, the name of the entity the instance refers to, and the port map information necessary to interconnect the entity with external signals. During compilation, each entity will have only one proto data structure, while, in the case of multiple instantiations of an entity, each entity may have one or more instance data structures.

In order to incrementally compile a model efficiently, HDL compiler **342** follows a recursive method of compilation in which successive entities of the model are considered and loaded from proto files **345** if such files are available and are consistent with the HDL source files constituting those entities and their descendants. For each entity that cannot be loaded from existing proto files **345**, HDL compiler **342** recursively examines the descendants of the entity, loads those descendant entities available from proto file(s) **345** and creates, as needed, proto files **344** for those descendants that are inconsistent with proto files **345**. Pseudocode for the main control loop of HDL compiler **342** is shown below (the line numbers to the right of the pseudocode are not a part of the pseudocode, but merely serve as a notational convenience).

```
process_HDL_file(file)                                          5
{                                                               10
        if (NOT proto_loaded(file)) {                           15
                if (exists_proto_file(file) AND check_bom(file)) { 20
                        load_proto(file);                       25
                } else {                                        30
                        parse_HDL_file(file)                    35
                        for (all instances in file) {           40
                                process_HDL_file(instance);     45
                        }                                       50
                        create_proto(file);                     55
                        write_proto_file(file);                 60
                }                                               65
        }                                                       70
        create_instance(file):                                  75
}                                                               80
```

When compiler **342** is initially invoked, no proto data structures **341** or instance data structures **343** are present in memory **44** of computer system **10**. The main control loop, routine process_HDL_file() (line 5), is invoked and passed the name of the top level entity by means of parameter "file". The algorithm first determines if a proto data structure for the current entity is present in memory **44** by means of routine proto_loaded() (line 15). Note that the proto data structure for the top level entity will never be present in memory because the process starts without any proto data structures loaded into memory **44**. If a matching proto data structure is present in memory **44**, instance data structures for the current entity and the current entity's descendants, if any, are created as necessary in memory **44** by routine create_instance() (line 75).

However, if a matching proto data structure is not present in memory **44**, control passes to line 20 where routine exists_proto_file() examines proto files **345** to determine if a proto file exists for the entity. If and only if a matching proto file exists, routine check_bom() is called to determine whether proto file **345** is consistent. In order to determine whether the proto file is consistent, the BOM for the proto file is examined. Routine check_bom() examines each HDL source code file listed in the BOM to determine if the date or time stamps for the HDL source code file have changed or if the HDL source code file has been deleted. If either condition occurs for any file in the BOM, the proto file is inconsistent and routine check_bom() fails. However, if check_bom() is successful, control is passed to line 25

where routine load_proto() loads the proto file and any descendant proto files into memory **44**, thus creating proto data structures **341** for the current entity and the current entity's descendants, if any. The construction of process_HDL_file() ensures that once a proto file has been verified as consistent, all of its descendant proto files, if any, are also consistent.

If the proto file is either non-existent or is not consistent, control passes to line 35 where routine parse_HDL_file() loads the HDL source code file for the current entity. Routine parse_HDL_file() (line 35) examines the HDL source code file for syntactic correctness and determines which descendant entities, if any, are instantiated by the current entity. Lines 40, 45, and 50 constitute a loop in which the routine process_HDL_file() is recursively called to process the descendent entities that are called by the current entity. This process repeats recursively traversing all the descendants of the current entity in a depth-first fashion creating proto data structures **341** and proto data files **344** of all descendants of the current entity. Once the descendant entities are processed, control passes to line 55 where a new proto data structure is created for the current entity in memory **44** by routine create_proto(). Control then passes to line 60 where a new proto file **344**, including an associated BOM, is written to disk **33** by routine write_proto_file(). Finally, control passes to line 75 where routine create_instance() creates instance data structures **343** for the current entity and any descendant entities as necessary. In this manner, process_HDL_file() (line 5) recursively processes the entire simulation model creating an in-memory image of the model consisting of proto data structures **341** and instance data structures **343**.

As further shown in **FIG. 3C**, the present invention further permits the designer to include within design entity HDL files **340** one or more signal group directives **350** identifying particular signals that are likely to be of interest when viewing the results of simulating simulation executable model **348**. Exemplary semantics for signal group directives **350** is described below with reference to **FIGS. 6A-6B**. HDL compiler **342**, in addition to the processing described above, preferably processes signal group directives **350** to generate signal group information (SGI) **400**, which represents the signal instantiation identifiers of the signals of interest utilizing any convenient data structure (e.g., linked list, table, etc.). Model build tool **346** then places the signal group information (SGI) **400**, optionally with some additional transformation in format, within simulation executable model **348**.

With reference now to **FIG. 3D** there is depicted a block diagram representing compiled data structures, which may be implemented in a preferred embodiment of the present invention. Memory **44** contains proto data structures **361**, one for each of the entities referred to in simulation model **329**. In addition, instantiations in simulation model **329** are represented by instance data structures **362**. Instance data structures **362** are connected by means of pointers indicating the hierarchical nature of the instantiations of the entities within simulation model **329**. Finally, memory **44** contains SGI **400**. Model build tool **346** in **FIG. 3C** processes the contents of memory **44** into memory data structures in order to produce simulation executable model **348**.

Referring now to **FIG. 4**, there is depicted a flow diagram of a process for simulating a design and viewing simulation results in accordance with the present invention. As shown, once a simulation executable model **348** has been obtained by the process of **FIG. 3C**, a software and/or hardware simulator **404** is utilized to stimulate simulation executable model **348** with a testcase **402** to simulate operation of a digital design. During the simulation, an all events trace (AET) file **406** records data representing the response of simulation executable model **348** to testcase **402**. The data within AET file **406** includes values of various signals and/or storage elements within simulation executable model **348** over time as well as SGI **400**.

In order to review the contents of AET file **406**, a user generally employs a separate or integrated viewer program, referred to herein as AET viewer **410**. For example, the user may request AET viewer **410** to present data from AET file **406** either in a graphical format within display screen **22** or in hardcopy format. As described hereinabove, the user can advantageously restrict the presentation of data by AET viewer **410** to particular signals of interest by specifying in a data set (referred to herein as an I/O list **408**) the signals of interest.

As illustrated in **FIG. 5A**, a conventional I/O list **500** in accordance with the prior art is a list containing a large number of entries each setting forth a signal instantiation identifier of one of the signals of interest, which in this case comprise all of the signals within FXU instantiation **321a** of **FIG. 3B**. Thus, as will be appreciated from the simplified example

given in **FIG. 5A**, in the prior art the user must enter each of a potentially large number of signal instantiation identifiers utilizing keyboard **16**. This conventional technique of manually keying in signal instantiation identifiers is tedious and error prone.

Additionally, some simulators **404** do not preserve the signal names of signals in descendant design entities that cross design entity boundaries into higher level design entities. Instead, to eliminate signal duplication in the AET file, such simulators **404** only identify a signal in the AET file by its signal name in the highest level design entity in which it appears. Consequently, if a simulator **404** that does not preserve signal names is employed, the user must specify a signal within the I/O list **500** utilizing a signal instantiation identifier that employs the signal name of the signal from the highest level design entity in which the signal appears. For example, as can be seen by comparing entry **502** of **FIG. 5A** with corresponding entry **504** of I/O list **500'** of **FIG. 5B**, the user of a simulator **404** that does not preserve signal names must utilize the signal instantiation identifier F0 as depicted at reference numeral **504** rather than the signal instantiation identifier FXU.E as illustrated at reference numeral **502**. As will be appreciated, a user of AET viewer **410** whose work predominantly pertains to a descendant design entity may have difficulty in determining or easily recalling signal names utilized in a higher level design entity that instantiates the lower level design entity with which the user is familiar.

In place of a conventional I/O list **500** or **500'**, the present invention permits a user of AET viewer **410** to instead filter presentation of data from AET file **406** utilizing one or more improved I/O lists **408** in accordance with **FIG. 5C**. As depicted in **FIG. 5C**, I/O list **408** is list containing one or more entries. In addition to zero or more entries containing conventional signal instantiation identifiers as depicted in **FIG. 5A** or **5B**, an entry of I/O list **408**, such as entry **510**, may identify a group of one or more signals of interest by a signal group instantiation identifier corresponding to information within SGI **400**.

As shown, the signal group instantiation identifier is formed similarly to a signal instantiation identifier and consists of a string of the enclosing entity instantiation names proceeding from the top-level entity instantiation name and terminating with the signal group name enclosed by a pair of angular brackets ("<" and ">") indicating that the

bracketed contents are a member of a separate signal group namespace. Thus, the six signals of interest within FXU entity instantiation **321a** can simply be identified by the single entry "FXU0.<FXU_Group>", rather than by individually entering the signal instantiation identifiers within I/O list **408**. As indicated above, the individual signals comprising the signal group FXU_Group are specified utilizing signal group directives **350** within design entity HDL files **340**.

With reference now to **FIG. 6A**, there is illustrated an exemplary embodiment of a design entity HDL file **340a** in accordance with the present invention. As will be appreciated by those skilled in the art, design entity HDL file **340a** includes conventional HDL source code describing a design entity, which in this case is design entity FXU **321**. The conventional HDL source code includes a port map **600** and signal assignment statements **602**. In addition, design entity HDL file **340a** includes unconventional HDL comments containing signal group directives **350** (**FIG. 3C**) in accordance with the present invention.

In design entity HDL file **340a**, the signal group directives **350** include two different types of signal group directives: a signal group declaration **610** and a signal preservation directive **620**. Signal group declaration **610** begins with an HDL comment of the form "--!! Signal Group *signal_group_name*;" and ends with an HDL comment of the form "--!! END Signal Group *signal_group_name*;", where *signal_group_name* is a signal group name (in this example, FXU_Group) that is unique for the given target design entity. Between the beginning and ending statements of signal group declaration **610**, the signal names of one or more signals of interest are listed in the desired order of presentation by AET viewer **410**. In this embodiment, signal names are specified relative to the target design entity (e.g., FXU design entity **321**). At least some embodiments of the present invention permit signal names higher in the design entity hierarchy to be specified relative to the target design entity utilizing the conventional syntax "..\" to indicate a next higher level of hierarchy.

The user is preferably permitted to further specify additional attributes related to the presentation of signals within signal group declaration **610**. For example, the user can specify a desired color for a signal, a default to a waveform or binary signal representation, a desired justification of unaligned bit vectors, etc. Thus, in statement **612** of signal group

declaration **610** the user has specified a left justification of the 5-bit signal vector B.C.P(0..4).

As further depicted in design entity HDL file **340b** of **FIG. 6B**, a signal group declaration, such as signal group declaration **630**, also preferably permits the user to specify nested signal groups to any legal depth. To specify a nested signal group comprising a portion of a larger signal group, the user simply includes a statement in a signal group declaration referring to the instantiation identifier of the nested signal group with the signal group enclosed in angular brackets (i.e., "<" and ">"). The use of angular brackets permits HDL compiler **342** to discriminate between the namespaces of signals and signal group names.

Referring back to **FIG. 6A**, a signal preservation directive **620** is utilized to instruct a simulator **404** that by default does not preserve the lower-level signal name of a renamed signal to do so for a particular renamed signal (e.g., signal G). Thus, assuming simulator **404** does not preserve the lower-level names of renamed signals by default, AET file **406** will contain data for signal instantiation identifiers TOP.FXU0.G and TOP.FXU1.G. As discussed above with reference to **FIG. 5B**, the capability to preserve a familiar signal name eliminates the need for the user to enter into an I/O list the possibly unfamiliar signal instantiation identifiers TOP.FXU1.H0 and TOP.FXU1.H1 to view the signal data of preserved signal G.

With reference now to **FIGS. 7A-7C**, there is depicted a high level logical flowchart of an exemplary process by which AET viewer **410** processes an I/O list **408** in accordance with the present invention. As a logical flowchart, operations are depicted logically rather than sequentially, and many of the illustrated operations can be performed in parallel or in an alternative order.

As illustrated, the process begins at block **700** of **FIG. 7A** and then proceeds to block **702**, which illustrates a determination of whether or not a user has entered a reference scope for the I/O list **408**, that is, a scope of simulation executable model **348** by reference to which all other I/O list entries will be parsed. By default, the reference scope is the top level design entity instance in the simulation executable model **348**, for example, top-level design entity

instance **320** of **FIG. 3B**. In one embodiment, the user is permitted to enter a command further limiting the reference scope of the form "Scope limit:*instance_string0*.[*design_entity*].*instance_string1*", where *instance_string0* and *instance_string1* are optional design entity instance strings (that for *instrance_string0* begins with the top-level design entity instance) and *design_entity* enclosed in square brackets is an optional design entity name. A legal reference scope command includes at least one of the three optional fields *instance_string0*, [*design_entity*], and *instance_string1*. The reference scope command can be communicated to AET viewer **410**, for example, via a command line or within I/O list **408**. If inserted within an I/O list **408**, the reference scope command preferably applies to all entries in that I/O list **408**.

As indicated at block **702**, if the user has not entered a different reference scope, AET viewer **410** sets the reference scope by default to the top level design entity instance of the simulation executable model **348**, as illustrated at block **704**. Thereafter, the process passes through page connector A to **FIG. 7B**. If AET viewer **410** determines at block **702** that the user has entered a different reference scope, AET viewer **410** further parses the reference scope command to determine at block **710** whether or not the reference scope command contains bracketed syntax (e.g., [*design_entity*]). If not, the process proceeds to block **713**, which illustrates AET viewer **410** determining whether of not the design entity instance specified by the reference scope command exists in the simulation executable model **348**. If not, processing terminates with an error at block **715**. If so, the process passes from block **713** to block **720**, which depicts AET viewer **410** setting the reference scope to the particular design entity instance specified within the *instance_string0* field of the reference scope command. For example, assuming simulation executable model **329** of **FIG. 3B**, the reference scope command "Scope limit:TOP.FXU0" sets the reference scope to instance **321a** of FXU design entity **321**. With this reference scope, entry **510** of I/O list **408** of **FIG. 5C** can be simplified as <FXU_Group>. Thereafter, the process passes through page connector A to **FIG. 7B**.

Returning to block **710**, in response to a determination that the reference scope command employs bracketed syntax, AET viewer **410** next determines at block **712** whether or not the named design entity exists within simulation executable model **348**. If not, processing ends

with an error at block **714**. If, however, the specified design entity is present in simulation executable model **348**, AET viewer **410** recursively searches simulation executable model **348** to identify all the instances of the design entity that are present within the portion of simulation executable model **348** defined by *instance_string0*, if present. As indicated at block **718**, if no instances of the specified design entity exist within the specified scope, processing terminates with an error at block **714**. If AET viewer **410** determines at block **718** that at least one instance of the specified design entity was located within the specified scope, AET viewer **410** further determines at block **722** if only a single instance of the design entity was found within the specified scope. If so, the process passes to block **720**, which illustrates AET viewer **410** setting that single instance of the design entity as the reference scope. Thereafter, the process passes through page connector A to **FIG. 7B**.

If AET viewer **410** determines at block **722** that more than one instance of the specified design entity was found within the specified scope, AET viewer **410** presents a list of the design entity instances to the user for selection, for example, via a graphical menu displayed within display screen **22** (block **724**). AET viewer **410** then receives user input designating a single one of the multiple design entity instances as defining the desired reference scope, as illustrated at block **726**. For example, assuming simulation executable model **329** of **FIG. 3B** and I/O list **408'** of **FIG. 5D**, the reference scope command "Scope limit:[FXU]" given at reference numeral **512** of **FIG. 5D** would cause AET viewer **410** to locate FXU instances **321a** and **321b** within simulation executable model **329** and present instance identifiers of instances **321a** and **321b** to the user for selection. Advantageously, regardless of which of FXU instance **321a** and **321b** is of interest and therefore selected by the user, the signal group can be specified as <FXU_Group>, as shown in entry **512** of I/O list **408'** of **FIG. 5D**. Thereafter, the process passes to block **720**, which has been described.

Referring now to **FIG. 7B**, the process begins at page connector A and then proceeds to block **730**, which represents a processing loop in which AET viewer **410** processes each entry within one or more I/O lists to **408** to construct a presentation of AET file **406**. If AET viewer **410** determines at block **730** that all entries within the I/O list(s) have been processed the process passes to block **732**, which is described below. If, however, at least one entry within an I/O list **408** remains to be processed, the process proceeds to block **740**, which

depicts AET viewer **410** moving to the first or next signal-identifying entry within the I/O list **408** and initializing a working scope for the entry to the reference scope. Signal-identifying entries within I/O lists **408** are processed with respect to (i.e., as impliedly limited by) the reference scope and take the general form:

instance_string2.[*design_entity* *].*instance_string3.signals*

where:

instance_string2 and *instance_string3* are optional design entity instance strings;

design_entity enclosed in square brackets (i.e., "[" and "]") is an optional design entity name;

* is an optional universal operator indicating all *design_entity* instances within the specified scope; and

signals is a required parameter that specifies a signal name, a signal group name enclosed in angular brackets (i.e., "<" and ">"), or empty angular brackets signifying all signal groups within the specified scope.

The process shown in **FIG. 7B** proceeds from block **740** to block **742**, which depicts AET viewer **410** determining whether the current signal-identifying entry has a leading design entity instance qualifier (e.g., *instance_string2*). If not, the process passes to block **752**, which is described below. If so, the process proceeds from block **742** to block **744**, which illustrates AET viewer **410** determining whether or not the specified design entity instance exists within the reference scope of simulation executable model **348**. If not, processing terminates with an error at block **746**. If so, the process proceeds to block **750**. Block **750** illustrates AET viewer **410** setting the current scope to the scope formed by appending the scope defined by the design entity instance qualifier to the reference scope. Next, AET viewer **410** determines at block **752** whether or not the next field within the signal-identifying entry of I/O list **408** is a *design_entity* qualifier enclosed in square brackets. If not, the process passes through page connector B to **FIG. 7C**. If so, processing of the signal-identifying entry continues at block **754**.

Block **754** shows AET viewer **410** recursively searching simulation executable mode l **348** to locate all design entity instances within the current scope having an entity name matching the specified design entity name. AET viewer **410** then determines at block **756** whether or not any such design entity instances exist. If not, processing ends with an error condition at block **746**. If so, AET viewer **410** further determines at block **760** whether or not the bracketed syntax includes an asterisk signifying the inclusion of all design entity instances within the specified scope. If so, AET viewer **410** narrows the working scope(s) of the entry to the one or more design entity instance(s) located at block **754** (block **761**). Thereafter, the process passes through page connector B to **FIG. 7C**.

Returning to block **760**, in response to a negative determination, the process proceeds to block **762**, which illustrates AET viewer **410** determining whether a single design entity instance was discovered at block **754**. If so, the working scope is set at block **764** to the single design entity instance. Thereafter, the process passes through page connector B to **FIG. 7C**. If, on the other hand, AET viewer **410** determines at block **762** that more than one design entity instances was discovered at block **754**, AET viewer **410** presents a list of the design entity instances to the user for selection, for example, via a graphical menu displayed within display screen **22** (block **770**). AET viewer **410** then receives user input designating one or more of the multiple design entity instances as defining the desired working scope, as illustrated at block **772**. Thereafter, the process passes to block **764**, which depicts AET viewer **410** establishing one or more working scopes in accordance with the user's selection. Thereafter, the process passes through page connector B to **FIG. 7C**.

With reference now to **FIG. 7C**, the process begins at page connector B and then proceeds to block **774**, which depicts AET viewer **410** further parsing the signal-identifying entry to determine whether the entry contains a further instance qualifier (e.g., *instance_string3*) to limit the working scope(s). If not, the process passes to block **780**, which is described below. If so, AET viewer **410** determines at block **776** whether or not the specified design entity instance(s) exist. If not, the process ends with an error at block **786**. If the specified design entity instance(s) exist, AET viewer **410** narrows the working scope(s) of the current signal-identifying entry to the design entity instance(s) indicated by the second instance qualifier (block **778**). Thereafter, the process passes to block **780**.

Block **780** depicts AET viewer **410** further parsing the signal-identifying entry of I/O list **408** to determine if the terminal *signals* field of the entry contains a single signal name or a signal group name. If the *signals* field contains a signal name, AET viewer **410** next determines at block **781** if the specified signal exists within the working scope(s). If not, processing terminates at block **786** with an error condition. If AET viewer **410** determines at block **781** if the specified signal exists within the working scope(s). If not, processing terminates at block **786** with an error condition. If so, AET viewer **410** adds the specified signal(s) to the presentation of AET file **406** (block **782**). Thereafter, the process returns to block **730** of **FIG. 7B** via page connector C.

Referring again to block **780**, in response to determining that the *signals* field of the signal-identifying entry does not contain a signal name, the process passes to block **783**, which depicts AET viewer **410** determining whether the *signals* field contains empty angle brackets. If so, AET viewer **410** recursively locates all signal group instances within the design entity instances within the working scope(s), as shown at block **784**. If AET viewer determines that none exists at block **785**, processing ends with an error at block **786**. If, on the other hand, AET viewer **410** determines at block **785** that one or more signal groups are present, AET viewer **410** determines the individual signals comprising the signal groups from SGI **400** and adds all such signals to the presentation at block **790**. Thereafter, the process returns through page connector C to **FIG. 7B**.

Returning to block **783**, if AET viewer **410** determines that the *signals* field of the signal-identifying entry of I/O list **408** does not contain empty angle brackets, but instead specifies a signal group name within angle brackets, the process passes to block **787**. Block **787** illustrates AET viewer **410** recursively locating instances of the specified signal group instances within the design entity instances within the working scope(s). As represented by block **788**, if no signal group instances are located, processing ends with an error condition at block **786**. Alternatively, if at least one instance of the named signal group is located, processing proceeds to block **790** and following blocks which have been described.

As has been described, the present invention provides a method, system and program product for processing simulation results for presentation. In accordance with the present

invention, the amount of user input required to filter the presentation of simulation results is substantially reduced through the use of predetermined signal groups and, optionally, the use of scope commands. In addition, the ease of understanding simulation results is enhanced through support of signal preservation directives that enable a designer to designate signals for which signal names are to be preserved in the presence of signal renaming.

While the invention has been particularly shown as described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention. For example, one of the embodiments of the invention can be implemented with program code resident in random access memory **28** or non-volatile storage of one or more computer systems configured generally as described in **FIG. 1** and **FIG. 2**. Until required by computer system **10**, the set of program code may be stored in another computer readable storage device, such as disk drive **33** or CD-ROM, or in data storage of another computer and transmitted over a local area network or a wide area network, such as the Internet, when desired by the user. The program code embodied in a computer usable medium may be referred to as a computer program product.

CLAIMS

1. A method of data processing in a data processing system, said method comprising:

receiving as an input a data set including at least one entry specifying a signal group by a predetermined signal group name;

in response to receipt of the data set:

processing the entry in the data set to identify the signal group name;

accessing signal group information associated with an event trace file containing simulation results to determine signal names of multiple signals that are members of the signal group; and

including within a presentation of those simulation results from the event trace file that are associated with instances of said multiple signals.

2. The method of Claim 1, and further comprising establishing membership of the signal names in the signal group by reference to an HDL source code file describing a design entity instantiated within the simulation executable model.

3. The method of Claim 1 or 2, wherein said signal group comprises a first signal group and said first signal group comprises second and third signal groups that collectively have said multiple signals as members.

4. The method of Claim 1, 2 or 3 and further comprising:

receiving a preservation directive identifying a particular signal for which an associated signal name is to be preserved from renaming; and

in response to receipt of said preservation directive, identifying said signal by said associated signal name in the presentation.

5. The method of any one of claims 1 to 4, and further comprising:

receiving as an input a scope command indicating a reference scope in terms of a design entity instance in the simulation executable model; and

in response to receipt of the scope command, interpreting said at least one entry in said data set in relation to said reference scope.

6. The method of any one of Claims 1 to 5, and further comprising:

processing the entry in the data set to identify within the entry a design entity name of having multiple design entity instances within the simulation executable model; and

in response to identifying the design entity name, presenting instance identifiers of the multiple design entity instances for user selection; and

in response to user selection of one of the instance identifiers, narrowing a scope in which instances of signals belonging to said signal group are located.

7. A program product, comprising:

a computer readable medium; and

program code within the computer readable medium that causes a data processing system to perform a method including:

receiving as an input a data set including at least one entry specifying a signal group by a predetermined signal group name;

in response to receipt of the data set:

processing the entry in the data set to identify the signal group name;

accessing signal group information associated with an event trace file containing simulation results to determine signal names of multiple signals that are members of the signal group; and

including within a presentation of those simulation results from the event trace file that are associated with instances of said multiple signals.

8. The program product of Claim 7, wherein the method further includes establishing membership of the signal names in the signal group by reference to an HDL source code file describing a design entity instantiated within the simulation executable model.

9. The program product of Claim 7 or 8, wherein said signal group comprises a first signal group and said first signal group comprises second and third signal groups that collectively have said multiple signals as members.

10. The program product of Claim 7, 8 or 9, wherein the method further includes:

receiving a preservation directive identifying a particular signal for which an associated signal name is to be preserved from renaming; and

in response to receipt of said preservation directive, identifying said signal by said associated signal name in the presentation.

11. The program product of any one of Claims 7 to 10, wherein the method further includes:

receiving as an input a scope command indicating a reference scope in terms of a design entity instance in the simulation executable model; and

in response to receipt of the scope command, interpreting said at least one entry in said data set in relation to said reference scope.

12. The program product of any one of Claims 7 to 11, wherein the method further includes:

processing the entry in the data set to identify within the entry a design entity name of having multiple design entity instances within the simulation executable model; and

in response to identifying the design entity name, presenting instance identifiers of the multiple design entity instances for user selection; and

in response to user selection of one of the instance identifiers, narrowing a scope in which instances of signals belonging to said signal group are located.

13. A data processing system, comprising:

a processor; and

data storage coupled to the processor, said data storage including program code that causes the data processing system to perform a method including:

receiving as an input a data set including at least one entry specifying a signal group by a predetermined signal group name;

in response to receipt of the data set:

processing the entry in the data set to identify the signal group name;

accessing signal group information associated with an event trace file containing simulation results to determine signal names of multiple signals that are members of the signal group; and

including within a presentation of those simulation results from the event trace file that are associated with instances of said multiple signals.

14. The data processing system of Claim 13, wherein the method further includes establishing membership of the signal names in the signal group by reference to an HDL source code file describing a design entity instantiated within the simulation executable model.

15. The data processing system of Claim 13 or 14, wherein said signal group comprises a first signal group and said first signal group comprises second and third signal groups that collectively have said multiple signals as members.

16. The data processing system of Claim 13, 14 or 15, wherein the method further includes:

receiving a preservation directive identifying a particular signal for which an associated signal name is to be preserved from renaming; and

in response to receipt of said preservation directive, identifying said signal by said associated signal name in the presentation.

17. The data processing system of any one of Claims 13 to 16, wherein the method further includes:

receiving as an input a scope command indicating a reference scope in terms of a design entity instance in the simulation executable model; and

in response to receipt of the scope command, interpreting said at least one entry in said data set in relation to said reference scope.

18. The data processing system of any one of Claims 13 to 17, wherein the method further includes:

processing the entry in the data set to identify within the entry a design entity name of having multiple design entity instances within the simulation executable model; and

in response to identifying the design entity name, presenting instance identifiers of the multiple design entity instances for user selection; and

in response to user selection of one of the instance identifiers, narrowing a scope in which instances of signals belonging to said signal group are located.

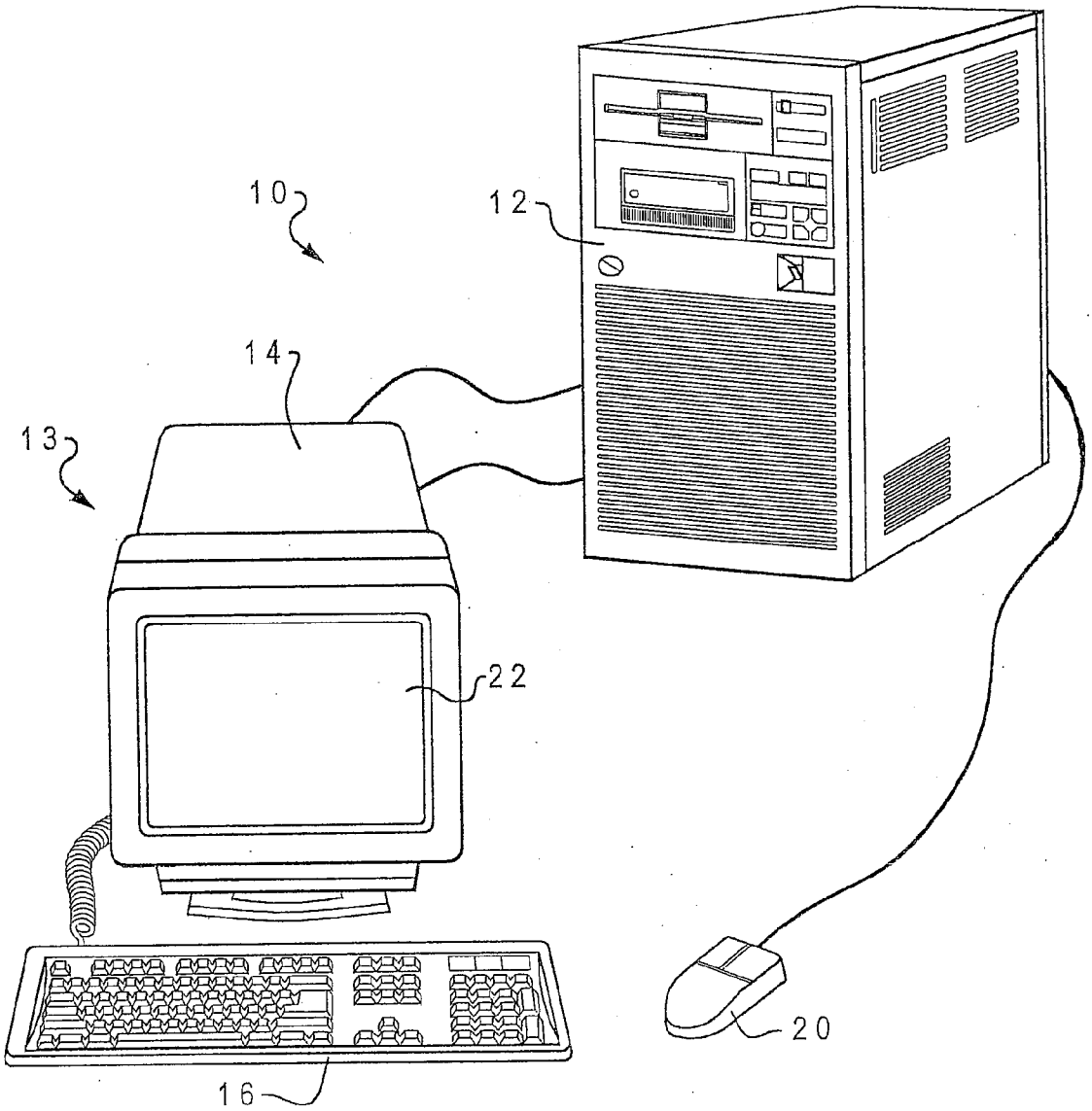


Fig. 1

Prior Art

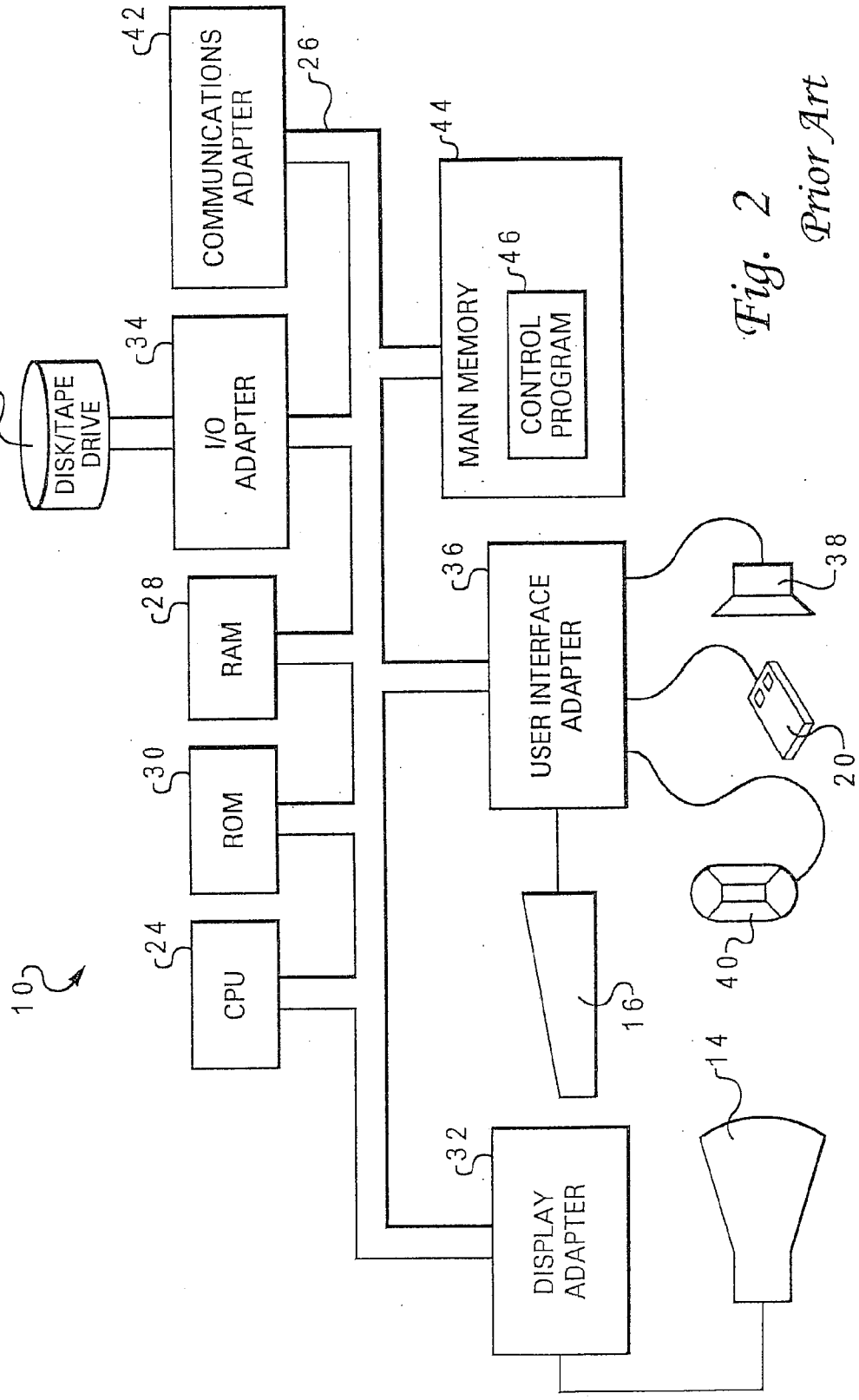


Fig. 2
Prior Art

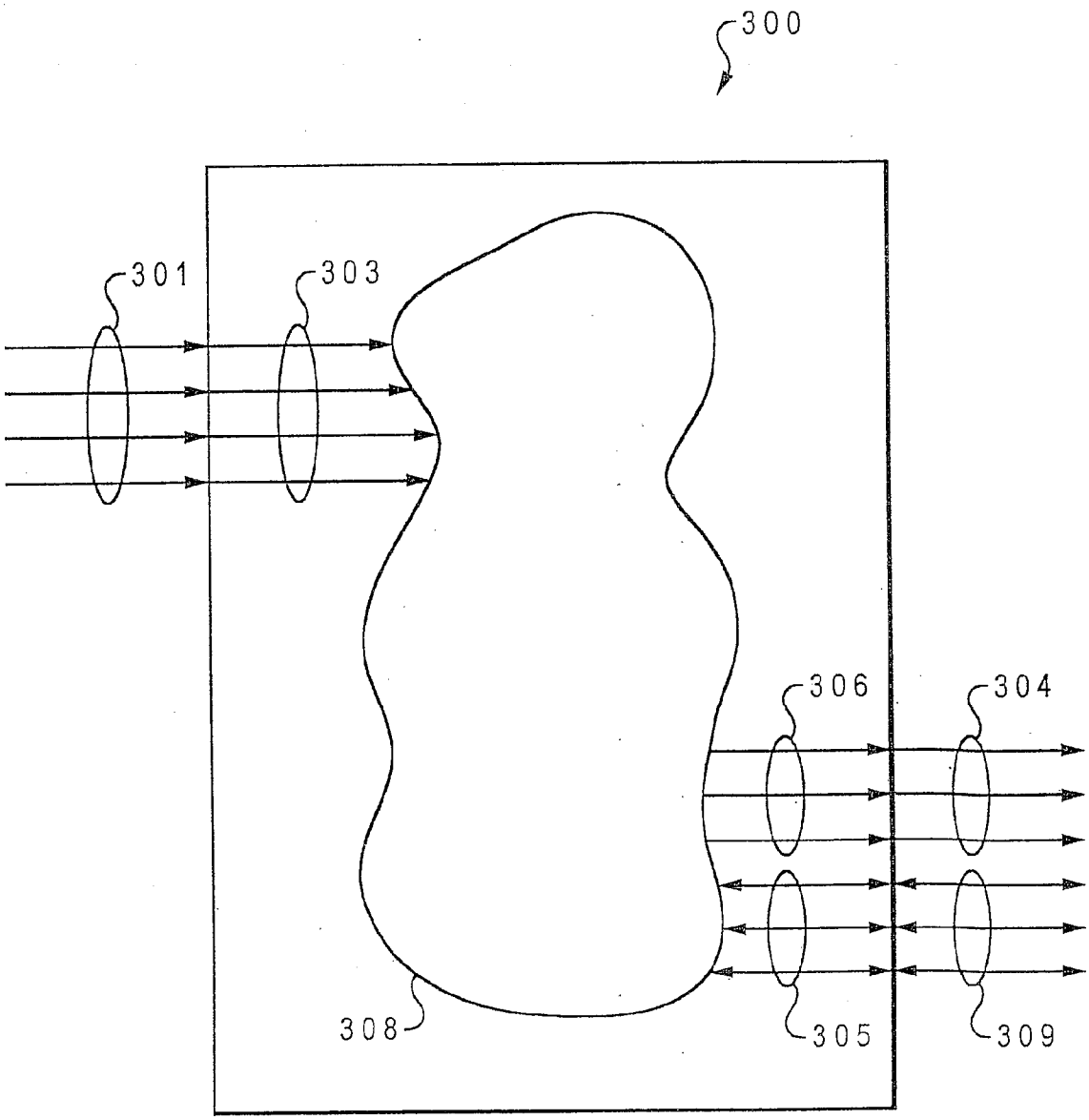


Fig. 3A

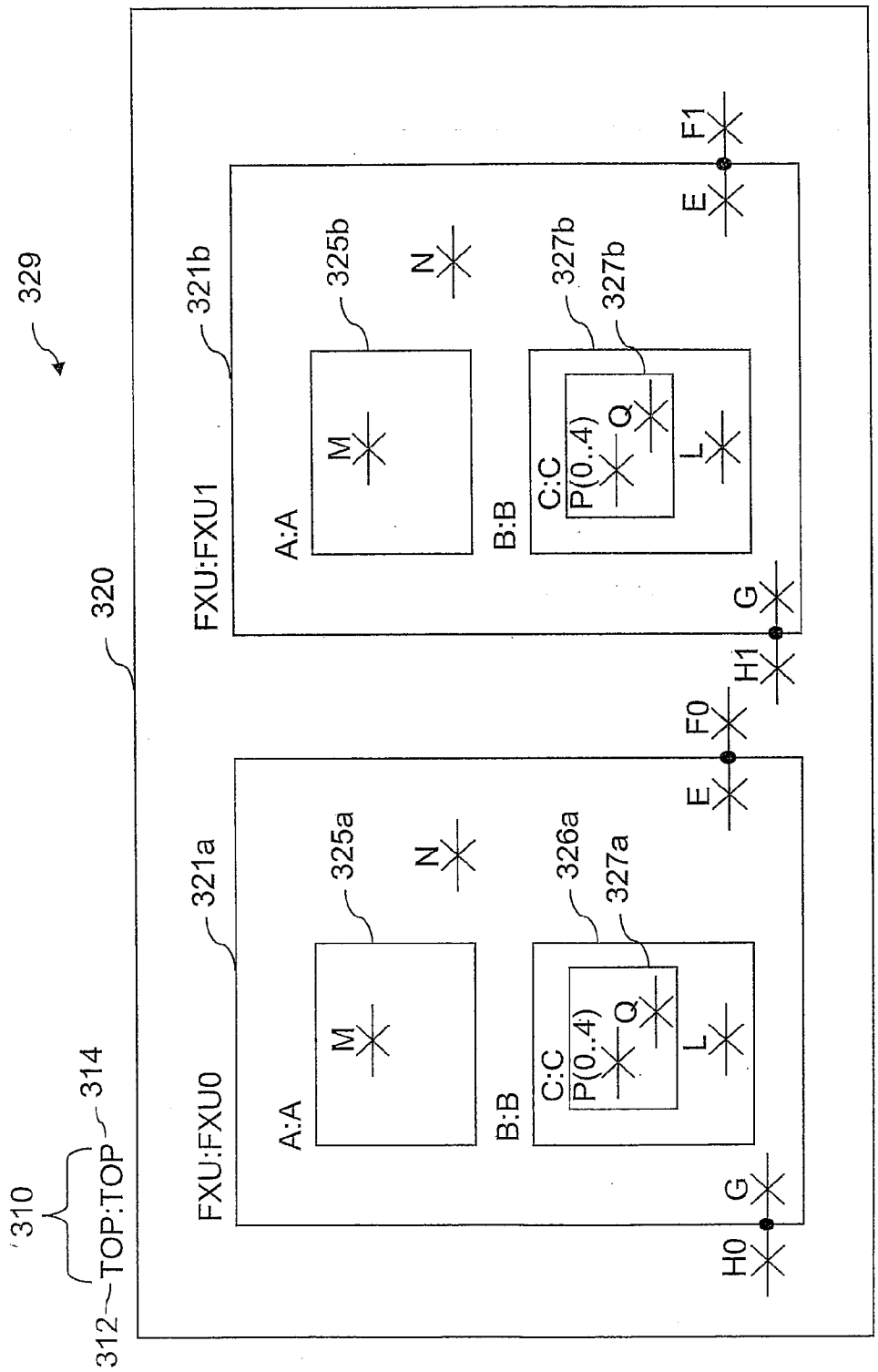

329
320
310
312
TOP:TOP
314
FXU:FXU0
321a
A:A
M
325a
N
B:B
C:C
P(0..4)
Q
L
326a
327a
G
H0
E
F0
FXU:FXU1
321b
A:A
M
325b
N
B:B
C:C
P(0..4)
Q
L
327b
327b
G
H1
E
F1

FIGURE 3B

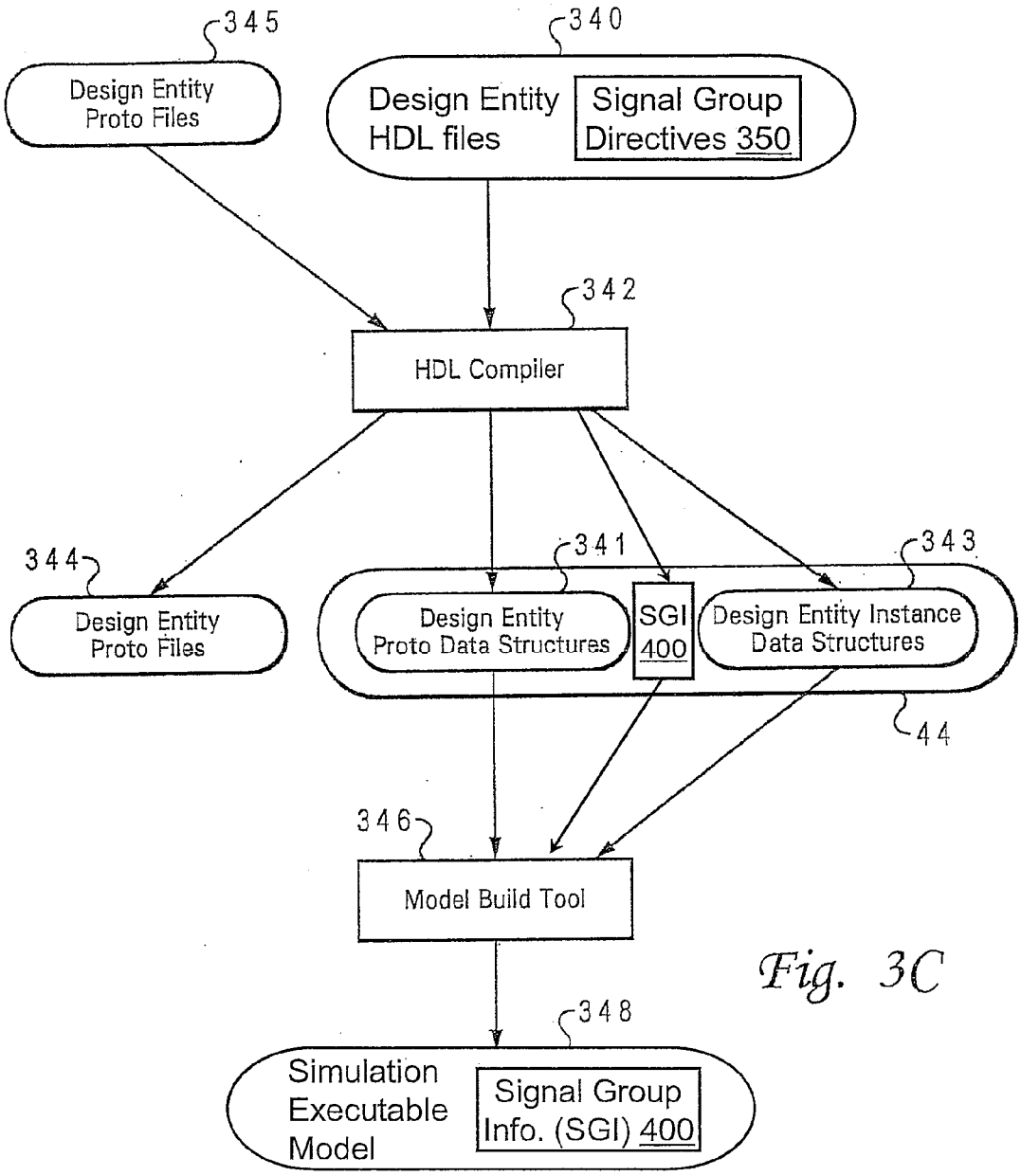

345
Design Entity Proto Files
340
Design Entity HDL files
Signal Group Directives 350
342
HDL Compiler
344
Design Entity Proto Files
341
Design Entity Proto Data Structures
SGI 400
343
Design Entity Instance Data Structures
44
346
Model Build Tool
348
Simulation Executable Model
Signal Group Info. (SGI) 400

Fig. 3C

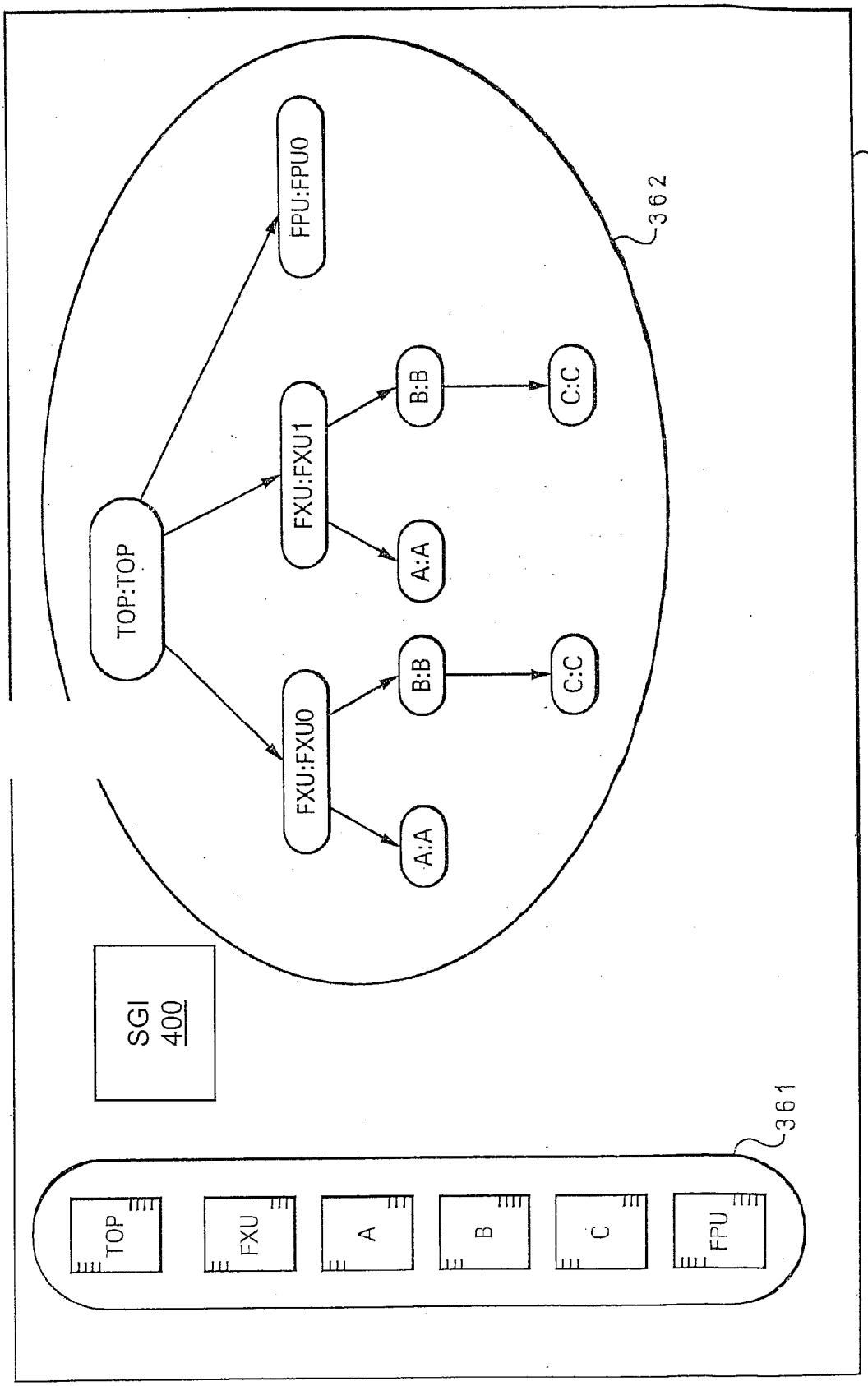

44
362
361
SGI
400
TOP:TOP
FPU:FPU0
FXU:FXU1
FXU:FXU0
A:A
B:B
C:C
A:A
B:B
C:C
TOP
FXU
A
B
C
FPU

Fig. 3D

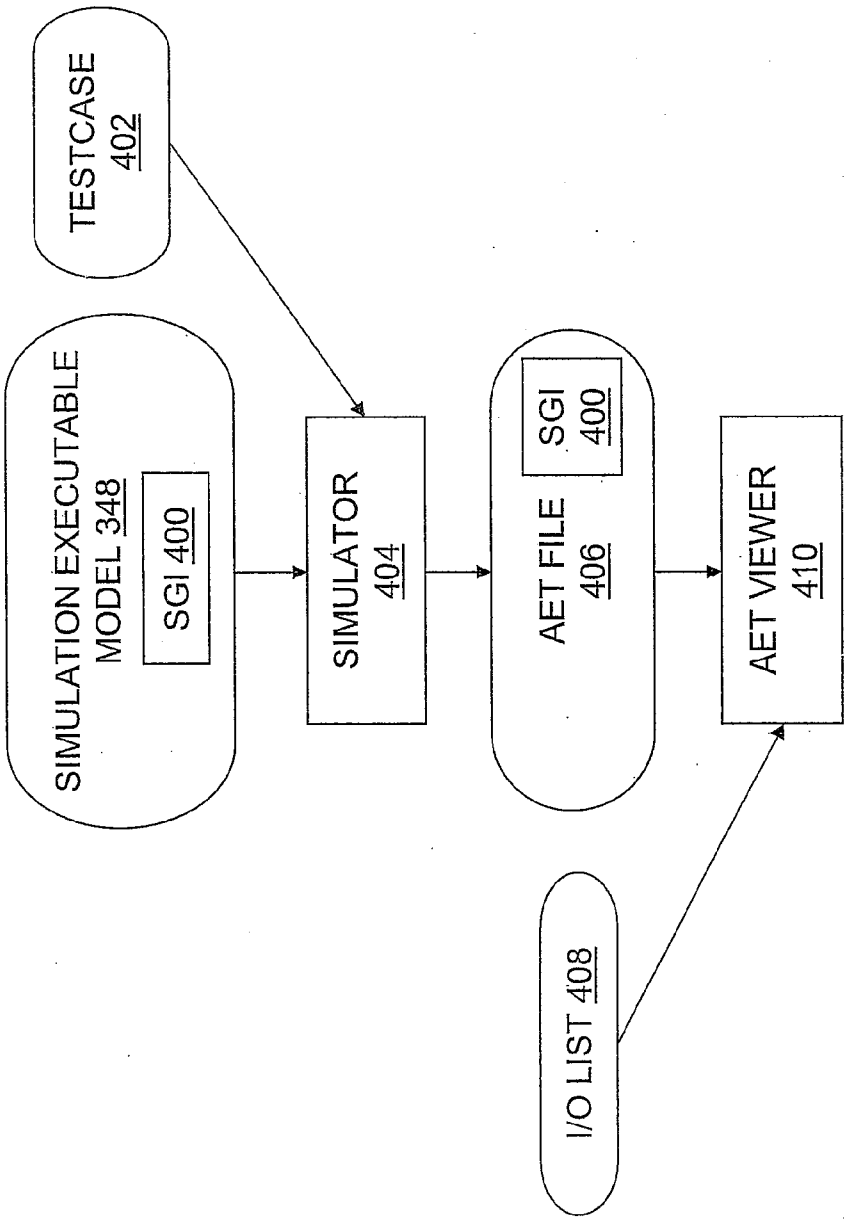
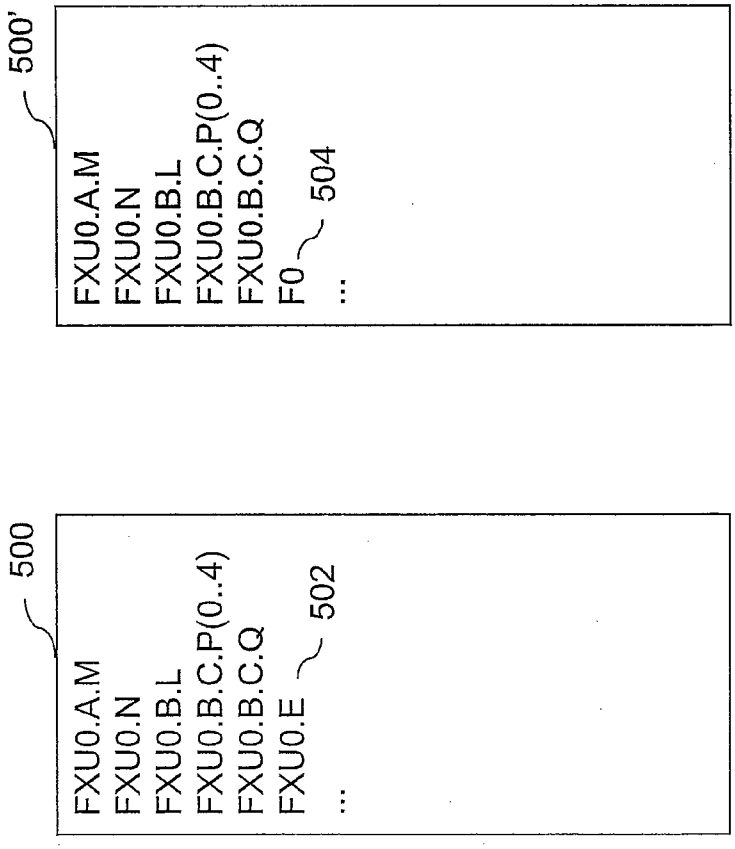

TESTCASE
402
SIMULATION EXECUTABLE
MODEL 348
SGI 400
SIMULATOR
404
AET FILE
406
SGI
400
AET VIEWER
410
I/O LIST 408

FIGURE 4



PRIOR ART

FIGURE 5A

PRIOR ART

FIGURE 5B

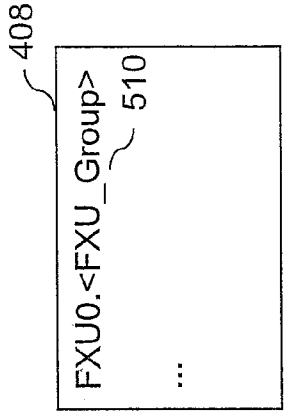


FIGURE 5C

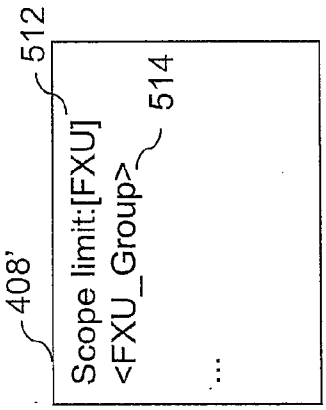


FIGURE 5D

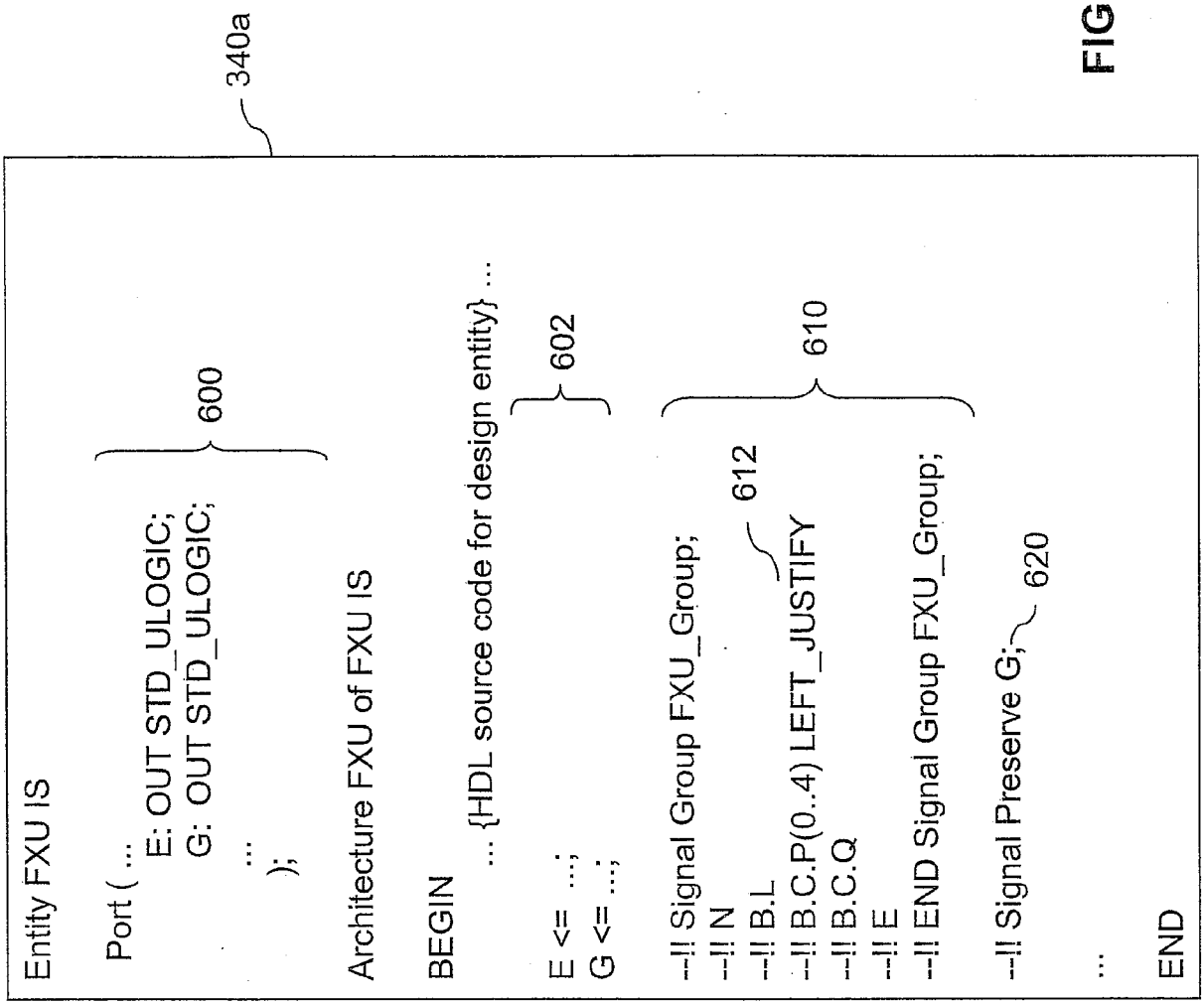

340a
Entity FXU IS
Port (...
E: OUT STD_ULOGIC;
G: OUT STD_ULOGIC;
...
);
600
Architecture FXU of FXU IS
BEGIN
... {HDL source code for design entity} ...
602
E <= ...;
G <= ...;
--!! Signal Group FXU_Group;
--!! N
612
--!! B.L
--!! B.C.P(0..4) LEFT_JUSTIFY
--!! B.C.Q
--!! E
--!! END Signal Group FXU_Group;
610
--!! Signal Preserve G;
620
...
END

FIGURE 6A

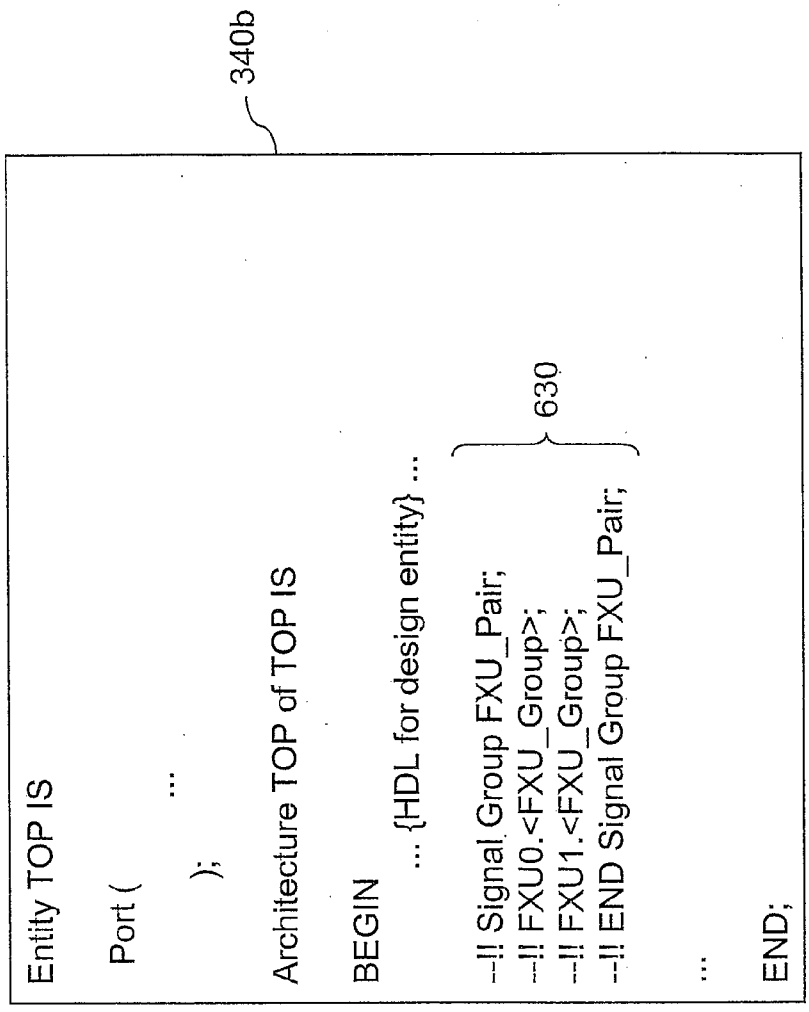

340b
Entity TOP IS
Port (
...
);
Architecture TOP of TOP IS
BEGIN
... {HDL for design entity} ...
--!! Signal Group FXU_Pair;
--!! FXU0.<FXU_Group>;
--!! FXU1.<FXU_Group>;
--!! END Signal Group FXU_Pair;
630
...
END;

FIGURE 6B

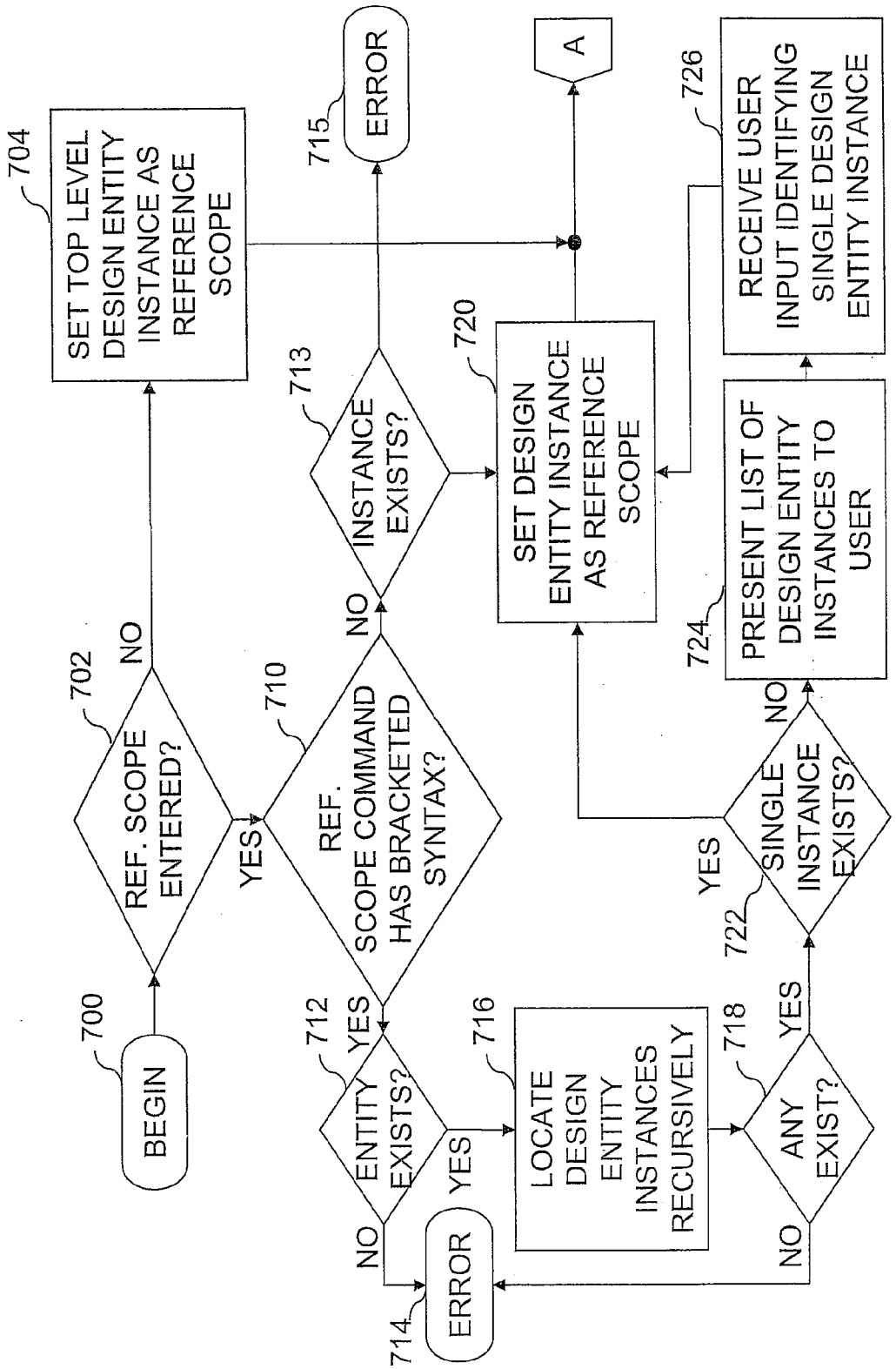

BEGIN
700
REF. SCOPE ENTERED?
702
NO
YES
SET TOP LEVEL DESIGN ENTITY INSTANCE AS REFERENCE SCOPE
704
REF. SCOPE COMMAND HAS BRACKETED SYNTAX?
710
NO
YES
ENTITY EXISTS?
712
NO
YES
ERROR
714
INSTANCE EXISTS?
713
NO
YES
ERROR
715
LOCATE DESIGN ENTITY INSTANCES RECURSIVELY
716
ANY EXIST?
718
NO
YES
SINGLE INSTANCE EXISTS?
722
NO
YES
SET DESIGN ENTITY INSTANCE AS REFERENCE SCOPE
720
A
PRESENT LIST OF DESIGN ENTITY INSTANCES TO USER
724
RECEIVE USER INPUT IDENTIFYING SINGLE DESIGN ENTITY INSTANCE
726

FIGURE 7A

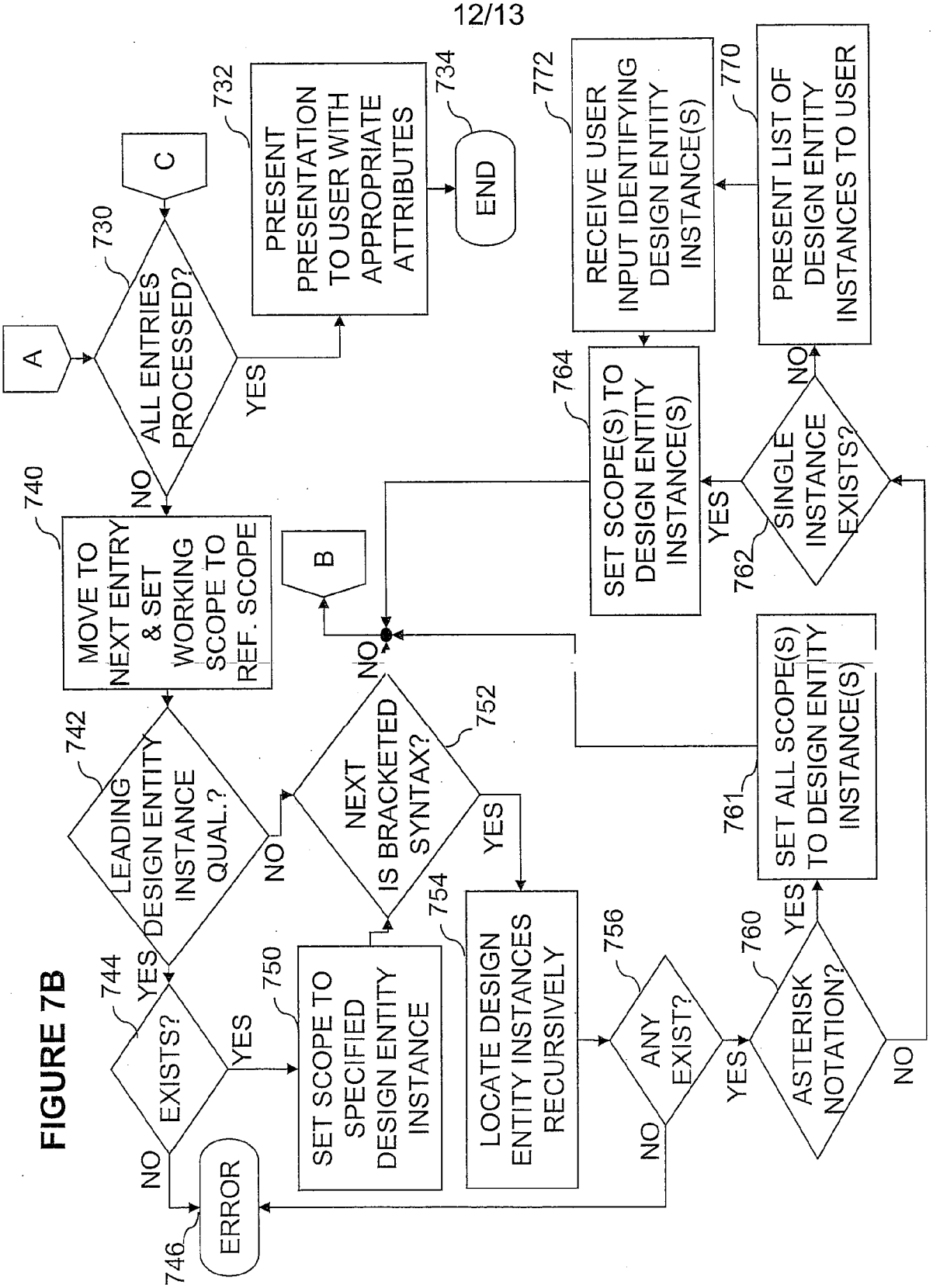

A
C
730
ALL ENTRIES PROCESSED?
NO
YES
732
PRESENT PRESENTATION TO USER WITH APPROPRIATE ATTRIBUTES
734
END
740
MOVE TO NEXT ENTRY & SET WORKING SCOPE TO REF. SCOPE
B
742
LEADING DESIGN ENTITY INSTANCE QUAL.?
744
EXISTS?
746
ERROR
750
SET SCOPE TO SPECIFIED DESIGN ENTITY INSTANCE
752
NEXT IS BRACKETED SYNTAX?
754
LOCATE DESIGN ENTITY INSTANCES RECURSIVELY
756
ANY EXIST?
760
ASTERISK NOTATION?
761
SET ALL SCOPE(S) TO DESIGN ENTITY INSTANCE(S)
762
SINGLE INSTANCE EXISTS?
764
SET SCOPE(S) TO DESIGN ENTITY INSTANCE(S)
770
PRESENT LIST OF DESIGN ENTITY INSTANCES TO USER
772
RECEIVE USER INPUT IDENTIFYING DESIGN ENTITY INSTANCE(S)

FIGURE 7B

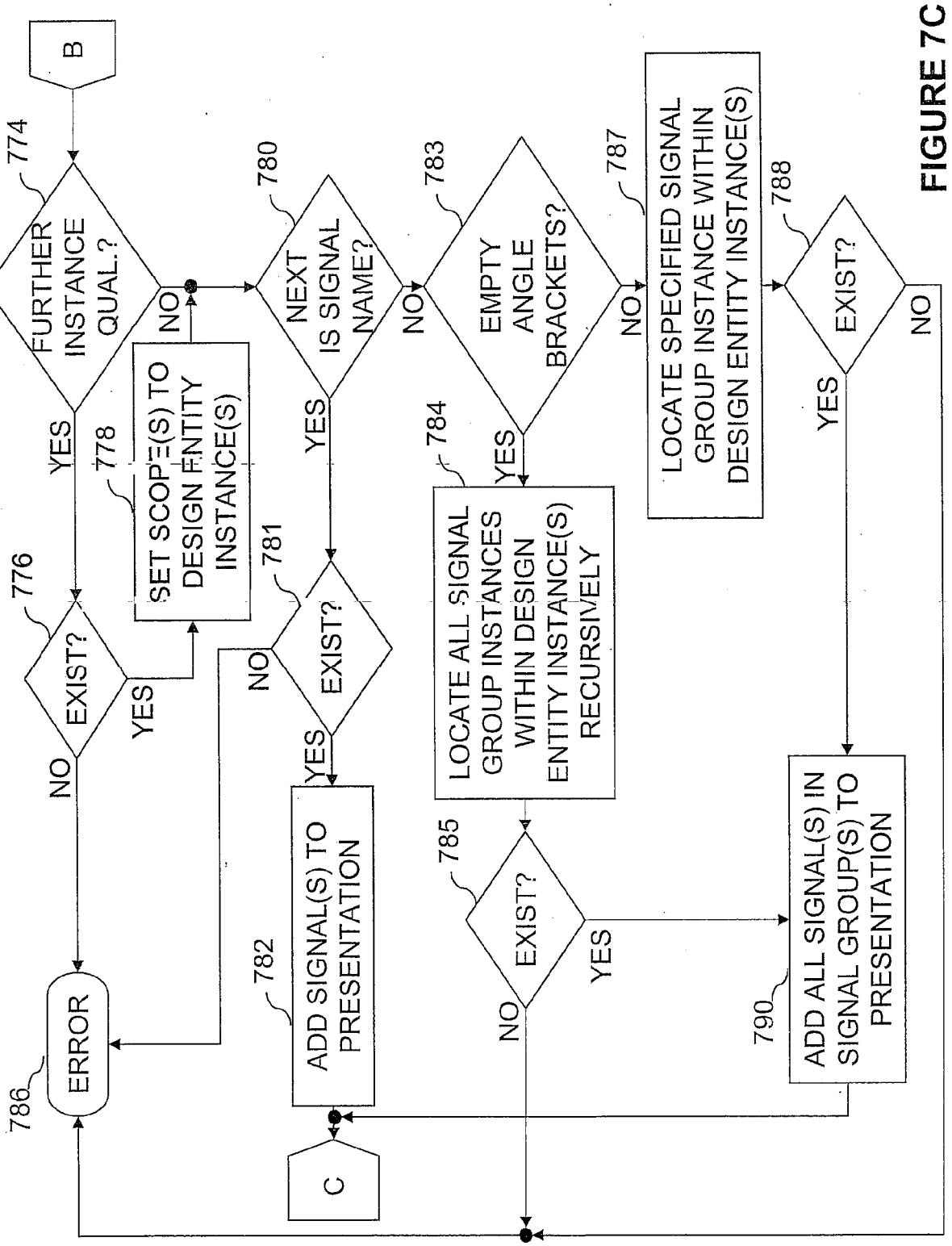

B
774
FURTHER INSTANCE QUAL.?
YES
NO
776
EXIST?
NO
YES
778
SET SCOPE(S) TO DESIGN ENTITY INSTANCE(S)
780
NEXT IS SIGNAL NAME?
YES
NO
781
EXIST?
NO
YES
782
ADD SIGNAL(S) TO PRESENTATION
783
EMPTY ANGLE BRACKETS?
YES
NO
784
LOCATE ALL SIGNAL GROUP INSTANCES WITHIN DESIGN ENTITY INSTANCE(S) RECURSIVELY
785
EXIST?
NO
YES
787
LOCATE SPECIFIED SIGNAL GROUP INSTANCE WITHIN DESIGN ENTITY INSTANCE(S)
788
EXIST?
YES
NO
790
ADD ALL SIGNAL(S) IN SIGNAL GROUP(S) TO PRESENTATION
786
ERROR
C

FIGURE 7C

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2007/054255

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, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	ADVANCED PROCESSOR TECHNOLOGIES GROUP: "GTKWave Electronic Waveform Viewer"[Online] 20 April 2005 (2005-04-20), XP002444396 Retrieved from the Internet: URL:http://intranet.cs.man.ac.uk/apt/projects/tools/gtkwave/> [retrieved on 2007-07-27] the whole document	1-18
X	INTERCONNECT SYSTEMS SOLUTION: "Wave VCD Viewer"[Online] 24 May 2005 (2005-05-24), XP002444620 Retrieved from the Internet: URL:http://web.archive.org/web/20051227174914/http://www.iss-us.com/wavevcd/index.htm> [retrieved on 2007-07-27] the whole document	1-18

-/--

[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
27 July 2007	13/08/2007

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	Anticoli, Claud

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2007/054255

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
L,X	VERIPOOL: "Dinotrace"[Online] 27 July 2007 (2007-07-27), XP002444621 Retrieved from the Internet: URL:http://www.veripool.com/dinotrace/> [retrieved on 2007-07-27] the whole document	1-18
A	US 2005/131666 A1 (TSAI JIEN-SHEN [TW] ET AL) 16 June 2005 (2005-06-16) paragraphs [0001] - [0022]	1-18
A	MASTON G A ED - INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS: "Considerations for STIL data application" PROCEEDINGS INTERNATIONAL TEST CONFERENCE 2002. ITC 2002. BALTIMORE, MD, OCT. 7-10, 2002, INTERNATIONAL TEST CONFERENCE, NEW YORK, NY : IEEE, US, 7 October 2002 (2002-10-07), pages 290-296, XP010609752 ISBN: 0-7803-7542-4 pages 1-2	1-18
A	US 2006/015314 A1 (ROESNER WOLFGANG [US] ET AL) 19 January 2006 (2006-01-19) abstract paragraphs [0001] - [0012]	1-18
A	WO 2005/001720 A (SIEMENS AG [DE]; ANDERSCH AXEL [DE]; MAENDL WOLFGANG [DE]; SEHR WOLFGA) 6 January 2005 (2005-01-06) page 1, line 1 - page 4, line 30	1-18

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2007/054255

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005131666 A1	16-06-2005	NONE	
US 2006015314 A1	19-01-2006	CN 1716264 A	04-01-2006
WO 2005001720 A	06-01-2005	DE 10329147 A1	20-01-2005
		EP 1639508 A1	29-03-2006
		US 2006178863 A1	10-08-2006