



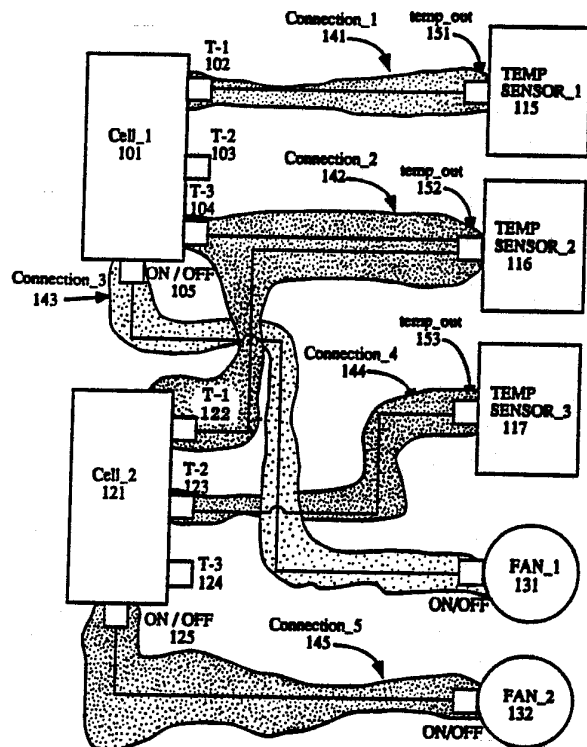
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

<p>(51) International Patent Classification <sup>5</sup> : G06F 15/40, 15/46, 15/56 G05B 19/00</p>	<p>A1</p>	<p>(11) International Publication Number: <b>WO 92/16905</b> (43) International Publication Date: 1 October 1992 (01.10.92)</p>
<p>(21) International Application Number: PCT/US92/02179 (22) International Filing Date: 17 March 1992 (17.03.92) (30) Priority data: 671,117 18 March 1991 (18.03.91) US (71) Applicant: ECHELON CORPORATION [US/US]; 4015 Miranda Avenue, Palo Alto, CA 94304 (US). (72) Inventors: DOLIN, Robert, A., Jr. ; 700 Wallea Drive, Menlo Park, CA 94025 (US). EINKAUF, Robert, L. ; 636 Pinot Blanc Way, Fremont, CA 94539 (US). RILEY, Glen, M. ; 918 Bicknell Road, Los Gatos, CA 95030 (US). (74) Agents: TAYLOR, Edwin, H. et al.; Blakely, Sokoloff, Taylor &amp; Zafman, 12400 Wilshire Boulevard, 7th Floor, Los Angeles, CA 90025 (US).</p>		<p>(81) Designated States: AT, AT (European patent), AU, BB, BE (European patent), BF (OAPI patent), BG, BJ (OAPI patent), BR, CA, CF (OAPI patent), CG (OAPI patent), CH, CH (European patent), CI (OAPI patent), CM (OAPI patent), DE, DE (European patent), DK, DK (European patent), ES, ES (European patent), FI, FR (European patent), GA (OAPI patent), GB, GB (European patent), GN (OAPI patent), GR (European patent), HU, IT (European patent), JP, KP, KR, LK, LU, LU (European patent), MC (European patent), MG, ML (OAPI patent), MR (OAPI patent), MW, NL, NL (European patent), NO, RO, RU, SD, SE, SE (European patent), SN (OAPI patent), TD (OAPI patent), TG (OAPI patent).</p> <p><b>Published</b> <i>With international search report.</i></p>

(54) Title: PROGRAMMING LANGUAGE STRUCTURES FOR USE IN A NETWORK FOR COMMUNICATING, SENSING AND CONTROLLING INFORMATION

(57) Abstract

An improved programming interface (501, 502, 503, 504, 505, 506) which provides for event scheduling improved variable declarations allowing for configuration of declaration parameters (1101, 1102, 1103, 1104) and improved handling of I/O objects (1201, 1202, 1203, 1204, 1205). Known computing devices allow for event scheduling based on the occurrence of a predefined event. However, such a system presents shortfalls in that flexibility is not provided to allow scheduling based on any arbitrary condition. Programs which run on such computing devices typically declare one or more variables; such variables having one or more parameters associated therewith. The present invention provides flexibility in allowing the states of the parameters to be changed, for example at the time a network implementing the computing device is configured. Additionally, it is disclosed to have improved declaration and control of I/O devices (115, 116, 117, 131, 132) providing for ease of use and flexibility.



**FOR THE PURPOSES OF INFORMATION ONLY**

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FI	Finland	MI	Mali
AU	Australia	FR	France	MN	Mongolia
BB	Barbados	GA	Gabon	MR	Mauritania
BE	Belgium	GB	United Kingdom	MW	Malawi
BF	Burkina Faso	GN	Guinea	NL	Netherlands
BG	Bulgaria	GR	Greece	NO	Norway
BJ	Benin	HU	Hungary	PL	Poland
BR	Brazil	IE	Ireland	RO	Romania
CA	Canada	IT	Italy	RU	Russian Federation
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland	KR	Republic of Korea	SN	Senegal
CI	Côte d'Ivoire	LI	Liechtenstein	SU	Soviet Union
CM	Cameroon	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
DE	Germany	MC	Monaco	US	United States of America
DK	Denmark	MG	Madagascar		
ES	Spain				

PROGRAMMING LANGUAGE STRUCTURES FOR USE IN A NETWORK  
FOR COMMUNICATING, SENSING AND CONTROLLING INFORMATION

1 BACKGROUND OF THE INVENTION

2

3 1. FIELD OF THE INVENTION

4 The present invention relates to the field of systems for distributed computing,  
5 communication and control and, more specifically, communication of information between  
6 devices in a distributed computing environment.

7

8 2. DESCRIPTION OF THE RELATED ART

9 In distributed computer systems it is necessary to provide for communication of  
10 information between nodes in the system. A number of methods for providing such  
11 communication are known in the art.

12 These methods include message passing techniques in which messages are passed,  
13 over a medium, from one node to another in a network. In message passing techniques,  
14 messages are built by a sender node and sent to one or more receiver nodes. The message  
15 is then parsed by the receiver node in order to correctly interpret the data. Message passing  
16 allows the advantage of passing large amounts of data in an expected format. Of course,  
17 over time the format of the message may be required to change to support new applications  
18 or features. This typically leads to compatibility issues between nodes on the network.

19 A second technique involves remote procedure calls in which a first node, requiring  
20 data which exists on a second node, calls a procedure executing on the second node where  
21 the data exists and requests the procedure to manipulate the data and provide a result to the  
22 first node. Remote procedure calls are typically suited to passing small amounts of data;  
23 however, a separate procedure call is typically required for each interchange. Therefore, it

1 is likely in any networking system that over time additional procedure calls will be required  
2 in the network as new functions are carried out by the network. The addition of new  
3 procedure calls to certain nodes of the network leads to incompatibility between nodes,  
4 because the existing nodes do not know of and cannot execute the new remote procedure  
5 calls.

6 A third technique for communication of data in a network involves data sharing.  
7 Bal, Henri E., Steiner, Jennifer G., and Tanenbaum, Andrew S., Programming Languages  
8 for Distributed Computing Systems, ACM Computing Surveys, Vol. 21, No. 3,  
9 September, 1989, pp. 261-322 (hereinafter Bal et al.) describes certain data sharing  
10 techniques. A discussion of data sharing may be found in the Bal et al. article at pages  
11 280, et seq. (It should also be noted that a discussion of messaging may be found in the  
12 Bal et al. article at pages 276, et seq. along with a general overview of interprocess  
13 communication and synchronization.)

14 Bal et al. describes how parts of a distributed program can communicate and  
15 synchronize through use of shared data. Bal et al. states that, if two processes have access  
16 to the same variable, communication can take place by one processor setting the variable  
17 and the other processor reading the variable. This communication is described as being  
18 allowed to take place whether the two processors are both running on a host where the  
19 shared data is stored and thus can manipulate the shared data directly, or if the processes  
20 are running on different hosts and access to the shared data is accomplished by sending a  
21 message to the host on which the shared data resides.

22 Two types of shared data are described: (1) shared logical variables; and (2)  
23 distributed data structures. Briefly, shared logical variables are described as facilitating a

1 communication channel between processes through a "single-assignment" property.  
2 Initially, a shared logical variable is unbound, but once a value is assigned to the variable  
3 the variable is considered to be bound. An example is provided in which the three goals of  
4 conjunction:

5 goal\_1 (X, Y), goal\_2 (X, Y), and goal\_3 (X)

6 are assumed and solved in parallel by processes P1, P2 and P3. The variable X is  
7 initially unbound and represents a communication channel between the three processes. If  
8 any of the processes binds X to a value, the other processes can use this value. Likewise,  
9 Y is a channel between P1 and P2. Processes synchronize by suspending on unbound  
10 variables. For example, if Y is to be used to communicate information from P1 to P2, then  
11 P2 may suspend until Y is bound by P1.

12 It should be emphasized that in the described concept of shared logical variables,  
13 the term binding is used to describe a process of assigning a value to a variable. As will be  
14 seen below, the term binding is also used to describe the present invention, however, the  
15 meaning of the term is significantly different and the reader is cautioned to avoid confusion  
16 between the concepts represented by these two uses of this term. Generally, in the present  
17 invention, the term binding is used to indicate a process of associating a variable of one  
18 node with a variable of at least one other node. It is not necessary that the variable of either  
19 node has yet been assigned a data value.

20 Distributed data structures are data structures which may be manipulated  
21 simultaneously by several processes. In concept, all processes share one global memory  
22 termed "tuple space" or TS. The elements of TS are ordered sequences of values, similar  
23 to records in a language such as Pascal. Three operations may take place on TS: (1)

1 "OUT" adds a tuple to TS; (2) "READ" reads a tuple from TS; and (3) "IN" reads a tuple  
2 from TS and deletes it from TS. Thus, in order to change the value of a tuple in TS it is  
3 necessary to first perform an IN operation, then to manipulate the data, and then perform an  
4 OUT operation. The requirement that a tuple must first be removed by the IN operation  
5 makes it possible to build distributed systems without conflict between accesses by the  
6 various processes.

7 Bal et al. contrasts distributed data structures with interprocess communication  
8 techniques noting that communication accomplished by distributed data structures is  
9 anonymous. A process reading a tuple from TS does not know or care which other  
10 process inserted the tuple. Further, a process executing an OUT does not specify which  
11 process the tuple is intended to be read by.

12 Bal et al. states that in concept distributed data structures utilizing the tuple space  
13 implement conceptually a shared memory, although in implementation a physical shared  
14 memory is not required. However, as can be seen, in a system utilizing such distributed  
15 data structures a single copy of the data is stored in tuple space whether or not such tuple  
16 space is implemented as a single physical shared memory. Separate copies of the data are  
17 not maintained for the various processes or on the various hosts. In fact, maintaining  
18 separate copies would lead to data conflict possibilities as the various nodes attempted to  
19 coordinate updates of the variable between the various process and hosts. Thus, the reason  
20 for requiring use of the IN command to delete a tuple before allowing manipulation of the  
21 data represented by the tuple.

22 The present invention discloses a networked communication system which is  
23 perhaps closest in certain concepts to the described distributed data structures. However, it

1 can, of course, be appreciated that certain advantages may be gained from development of a  
2 system which utilizes certain features of distributed data structures while retaining  
3 flexibility in allowing multiple copies of a data value to be stored on the various nodes.

4         The present invention discloses certain improved programming language and data  
5 structure support for communication, sensing and control as may be used by nodes of the  
6 present invention. It is known in the art to allow for scheduling of tasks through use of a  
7 programming statement such as a "when" clause or the like. However, in known systems  
8 such tasks may only be scheduled to be executed on the occurrence of a predefined event  
9 such as may be defined by the compiler writer. Examples of such events typically include  
10 expiration of a timer or input pin state changes. Such known systems do not allow for  
11 definitions of events, other than such predefined events. It has been discovered that it is  
12 useful to provide for definition of events as any valid programming language statement  
13 which may be evaluated to a true or false condition.

14         Of course, any number of known systems allow for declaration of variables and  
15 when declaring such variables certain parameters may be specified which may be set to a  
16 state indicative of a desired characteristic of the variable. For example, a variable may be  
17 declared as input or output, as a given variable type (e.g., boolean, numeric, etc.).  
18 However, once declared such characteristics are static and may only be changed by  
19 changing the source program which declares the variables. It has been discovered that it  
20 would be useful to provide for a system in which the state of at least certain parameters may  
21 be changed during system configuration allowing for greater flexibility in optimizing the  
22 system of the preferred embodiment.

1           Finally, in known systems it is necessary to call certain I/O library procedures to  
2 declare and use I/O devices. Such calls to I/O procedures may be quite complex and  
3 require significant skill on the part of the programmer to properly code and utilize the  
4 routines. The present invention discloses a system having improved methods for  
5 declaration and use of I/O devices.

6

7

OBJECTS OF THE PRESENT INVENTION

8

9

          It is a primary object of the present invention to provide for improved  
communication of information between nodes of a distributed network.

10

11

12

13

14

15

          It is more specifically an object of the present invention to provide for improved  
communication of information in a highly distributed computer system in which a problem  
may be broken down into small units in which each node accomplishes a small part of the  
entire application. In such a system, data communication may be typically accomplished in  
relatively small units of data—however, significant communication of data between nodes  
of the network is required.

16

17

18

          It is further an object of the present invention to provide for improved  
communication of information in a distributed computing system by allowing for standard  
communication techniques between nodes.

19

20

21

          It is still further an object of the present invention to provide for improved  
communication of information by providing certain facilities, structures and tools for such  
communication.



1           It is also an object of the present invention to provide improved data structures and  
2 programming language support for communication and other aspects of the present  
3 invention.

4           As one aspect of providing such improved data structures and programming  
5 language support, it is one aspect of the present invention to provide for declaration of  
6 variables having configurable parameters leading to improved ability to maintain and  
7 optimize networks of the present invention.

8           As another aspect of providing such improved data structures and programming  
9 language support, it is desired to provide for increased ease in declaring and  
10 communicating with I/O devices of the present invention.

11           As still another aspect of providing such improved data structures and programming  
12 language support, it is desired to provide for improved scheduling functions allowing for  
13 use of programmer-defined or predefined events in scheduling of tasks to be executed.

14           It is also an object of the present invention to provide simplified installation and  
15 network maintenance. Such an objective may be accomplished by storing in each node the  
16 node's application interface such that nodes may identify themselves and their application  
17 requirements to a network management node at installation time and when it is necessary to  
18 recover the complete network database.

19           To accomplish such a simplified installation and maintenance objective, it is a  
20 further objective of the present invention to define an interface file format which may  
21 efficiently store and allow retrieval of such identification and application requirement  
22 information.

- 1           These and other objects of the present invention will be better understood with
- 2   reference to the Detailed Description of the Preferred Embodiment, the accompanying
- 3   drawings, and the claims.

1 SUMMARY OF THE INVENTION

2 A network for communicating information having at least a first and second node is  
3 described in which each node includes data storage for storing data representing a variable  
4 V and further includes a processor coupled with the data storage. In the case of the first  
5 node, the processor may manipulate and write to new values to the variable V. After  
6 having updated the variable V with a new value, the processor then assembles and  
7 communicates a packet for transmission on the network. The packet includes the new data  
8 value for the variable V. The second node then receives the packet and stores the new value  
9 for the variable V in its data storage.

10 In particular, during programming of the first node, it is declared as a writer of the  
11 variable V and likewise during programming of the second node, it is declared as a reader  
12 of the variable V. During configuration of the network, a communication connection  
13 between the first node and the second node is defined and during later communication of  
14 message packets, addressing of message packets between the various nodes is  
15 accomplished through use of address tables based on the definition of such connections.

16 Further, it is disclosed to utilize a standardized set of variable types in  
17 accomplishing such communication. Use of a standardized set of variable types leads to  
18 increased compatibility between nodes of different manufacture as well as increased ease in  
19 configuring networks.

20 Finally, certain extensions are provided to standard programming languages to  
21 provide for increased ease of use of the data communication features of the present  
22 invention.

- 1            These and other aspects of the present invention will be apparent to one of ordinary
- 2   skill in the art with further reference to the below Detailed Description of the Preferred
- 3   Embodiment and the accompanying drawings.

1

BRIEF DESCRIPTION OF THE DRAWINGS

2

3           Figure 1 is a logical view of a configuration of devices as may be networked using  
4 methods and apparatus of the present invention.

5

6           Figure 2 is a diagram illustrating an embodiment of the network of Figure 1 as may  
7 be accomplished by the present invention.

8

9           Figure 3(a) is a diagram illustrating a second embodiment of the network of Figure  
10 1 as may be accomplished by the present invention.

11

12           Figure 3(b) is a second diagram illustrating the second embodiment of Figure 3(a)  
13 and including a logical view of certain connections of nodes of the network.

14

15           Figure 4 is an overall block diagram of a single node of the present invention  
16 coupled with a communication medium.

17

18           Figure 5 is an overall flow diagram illustrating a method of programming and  
19 configuring a network as may be accomplished by the present invention.

20

21           Figure 6 is a flow diagram illustrating a method for defining hardware requirements  
22 of a node as may be utilized by the present invention.

23

1           Figure 7 is a flow diagram illustrating a method for defining network and logical  
2 parameters of a node as may be utilized by the present invention.

3

4           Figure 8(a) is a flow diagram illustrating a method for defining connections among  
5 network variables as may be utilized by the present invention.

6

7           Figure 8(b) is a flow diagram illustrating a method for binding network variables as  
8 may be utilized by the present invention.

9

10          Figure 9 is an illustration of certain data structures which may be utilized by the  
11 present invention.

12

13          Figure 10 is a flow diagram illustrating a method of configuring a network using  
14 standard network variable types, as may be utilized by the present invention.

15

16          Figure 11 is a flow diagram illustrating a method of declaring and configuring a  
17 network variable as may be used by the present invention.

18

19          Figure 12 is a flow diagram illustrating a method of declaring and accessing I/O  
20 devices as may be utilized by the present invention.

1  
2           For ease of reference, it might be pointed out that reference numerals in all of the  
3 accompanying drawings typically are in the form "drawing number" followed by two  
4 digits, xx; for example, reference numerals on Figure 1 may be numbered 1xx; on Figure  
5 9, reference numerals may be numbered 9xx. In certain cases, a reference numeral may be  
6 introduced on one drawing, e.g., reference numeral 201 illustrating a communication  
7 medium, and the same reference numeral may be utilized on other drawings to refer to the  
8 same item.  
9

1                                    DETAILED DESCRIPTION OF  
2                                    THE PREFERRED EMBODIMENT

3            An improved computer network including facility for communication of information  
4 between nodes in the network is described. In the following description, numerous  
5 specific details are set forth in order to provide a thorough understanding of the present  
6 invention. It will be obvious, however, to one skilled in the art that the present invention  
7 may be practiced without these specific details. In other instances, well-known circuits,  
8 structures and techniques have not been shown in detail in order not to unnecessarily  
9 obscure the present invention.

10                                   OVERVIEW OF THE NETWORK  
11                                   OF THE PRESENT INVENTION

12            The network of the preferred embodiment is of the type which provides for  
13 sensing, control and communication. The network of the present invention and nodes  
14 utilized within the network of the present invention are described in greater detail with  
15 reference to United States Patent No. 4,918,690 Markkula et al. titled "Network and  
16 intelligent cell for providing sensing, bi-directional communications and control", which  
17 patent is assigned to the assignee of the present invention (referred to herein as the '690  
18 patent).

19            In an exemplary network, the network of the present invention may provide for  
20 sensing of current environmental factors and control of apparatus affecting the  
21 environmental factors. Further, the network may allow for communication of information  
22 packets providing information on the environmental factors between nodes in the network.  
23 The present application will utilize, as an example, a network for control of fans based on



1 sensing and communicating information regarding temperature in different zones in a  
2 controlled environment.

3 It might be worthwhile noting that in an expected scenario, various manufacturers  
4 will include a node of the type defined by the present invention in their products. For  
5 example, a thermostat manufacturer may include such a node in its thermostats. A fan  
6 manufacturer may include such a node in its fans. The various nodes may be programmed  
7 for specific applications by their respective manufacturers and, when configured in an  
8 environmental control system, are useful for communication, sensing and control between  
9 various components of the system. A node of the preferred embodiment is illustrated in  
10 block diagram form with reference to Figure 4. Such nodes may be programmed, for  
11 example, using the "C" computer programming language. As one aspect of the present  
12 invention, certain extensions have been provided to the "C" language to facilitate network  
13 communications.

14 As a further and important aspect of the present invention, network variables are  
15 described which provide for communication of information between nodes of the network.  
16 A network variable may be thought of as a data object shared by multiple nodes where  
17 some nodes are "readers" and some nodes are "writers" of the object. This will be  
18 discussed in greater detail below.

19 A network as may be implemented utilizing the present invention

20 Referring now to Figure 1, a logical view of a network as may utilize the present  
21 invention is shown. The network may, for example, include three separate temperature  
22 sensors 115-117 located in three separate zones of a building for sensing and  
23 communicating temperature information. The network may further include two control

1 cells 101 and 121 coupled to receive temperature information from the sensors 115-117 and  
2 to control two fans 131-132 (by turning the fans 131-132 on and off).

3 In the exemplary network, network variable temp\_out 151 is coupled to a first  
4 network variable temperature input 102 of control cell 101. Network variable temp\_out  
5 152 is coupled with a second network variable temperature input 104 of control cell 101.  
6 In the illustrated embodiment, a third network variable temperature input 103 is not utilized.  
7 On/Off control network variable 105 of control cell 101 is coupled to control an input  
8 network variable, On/Off, of a fan 131. Thus, in this embodiment, sensing a temperature  
9 above a given level in the zone of the building sensed by temperature sensor 115 or by  
10 temperature sensor 116 will cause fan 131 to be turned on. Likewise, when the  
11 temperature in these zones is again lowered below a given level, the fan 131 may be turned  
12 off.

13 Network variable temp\_out 152 is also coupled to a first temperature input network  
14 variable 122 of control cell 121. In addition, network variable temp\_out 153 is coupled to  
15 a second temperature input network variable 123 of control cell 121. A third temperature  
16 input 124 of control cell 121 is not utilized in this embodiment. Control cell 121 is coupled  
17 through an On/Off control output network variable 125 to control fan 132. Thus, sensing a  
18 temperature above a given level in the zone of the building sensed by temperature sensor  
19 116 or by temperature sensor 117 will cause fan 132 to be turned on. Likewise, when the  
20 temperature in these zones is again lowered below a given level, the fan 132 may be turned  
21 off. As is appreciated, in the described configuration, when temperature sensor 116  
22 detects a high temperature, both fan 131 and fan 132 are turned on.

1           Figure 1 has been labeled to illustrate logical connections between the various  
2 components. Connection 141 is illustrated as the connection between temperature sensor  
3 115 and control cell 101. Connection 142 is illustrated as the connection including  
4 temperature sensor 116, control cell 101 and control cell 121. Connection 143 is illustrated  
5 as the connection between control cell 101 and fan 131. Connection 144 is illustrated as  
6 the connection between sensor 117 and control cell 121. Connection 145 is illustrated as  
7 the connection between control cell 121 and fan 132. The connection of network variables  
8 will be discussed in greater detail below. However, it may now be useful to introduce  
9 three new terms: network variables, readers, and writers. In addition, general definitions  
10 for certain other terms used in this specification may be found with reference to Table XV.

11           As one important aspect of the present invention, the present invention provides for  
12 allocation and use of network variables by processes running on nodes in a network. As  
13 stated above, network variables may be thought of as a data object shared by multiple  
14 nodes where some nodes are "readers" of the object and other nodes are "writers" of the  
15 object. Additionally, a node may be both a reader and a writer with "turnaround". Writing  
16 with turnaround is discussed in greater detail below. Although the data object may be  
17 thought of as being shared by multiple nodes, as will be understood from the discussion  
18 below, the network variable of the preferred embodiment is not stored in shared memory  
19 but rather separate memory is provided in each of the multiple nodes to store a copy of the  
20 data object. A writer node may modify the value of the data object and all reader nodes of  
21 that network variable have their memories updated to reflect the change. Thus, for  
22 example, each of the temperature sensors 115-117 may run a process which declares a data  
23 object as follows:

1 network output boolean temp\_high.

2 Each of the controller cells 101 and 121 may declare data objects as follows:

3 network input boolean temp\_high

4 network output boolean fan\_on.

5 Each of the fans 131-132 may declare a data object as follows:

6 network input boolean fan\_on.

7 The complete syntax for declaration of network variables in the system of the  
8 preferred embodiment is given in Table VIII. The keyword "network" indicates the data  
9 object is a network variable. A network variable declared as output will result in  
10 transmission of the new value of the network variable on the network when the program  
11 stores the variable—thus, nodes having declared an output network variable are considered  
12 writer nodes for that variable. For example, each time a process running on temperature  
13 sensor 115 stores the variable temp\_high, a network message is generated communicating  
14 the new value of temp\_high. The message is communicated to all reader nodes connected  
15 in connection\_1 141, i.e., to control cell 101. In the case of temperature sensor 116  
16 changing the value of its temp\_high variable, a message is generated and transmitted to all  
17 nodes connected in connection\_2 142, i.e., to both control cell 101 and to control cell 121.  
18 The process for configuring connections as disclosed by the present invention will be  
19 discussed in greater detail below.

20 Although the preferred embodiment declares nodes as either writers or readers of  
21 network variables, it should be noted that in an alternative embodiment a node may be  
22 declared as a both a reader and writer of a particular variable. Such an embodiment may be  
23 envisioned without departure from the spirit and scope of the present invention.

1           It might be that the present invention in its preferred embodiment allows an output  
2 network variable to be initialized using an initialization command without causing a  
3 message to be transmitted on the network. Using this command, a node may be initially  
4 configured or reset without affecting other nodes on the network.

5           Network variables declared as input may change values asynchronously with  
6 program execution—this declaration is used for "reader" nodes. In the preferred  
7 embodiment, input network variables may also change values at program initialization or at  
8 other points under program control; however, the changed value will not be transmitted on  
9 the network.

10           At anytime, a reader node may force an update of its input network variables  
11 utilizing a polling function of the present invention. When this function is called, the  
12 specified network variables are updated by requesting over the network the current value  
13 from the writer node or nodes. This facility may be useful, for example, after a node reset  
14 to allow the node to determine the current value of network variables without need to wait  
15 until the next time the writer nodes update the value of those variables.

16           Thus, temperature sensors 115-117 are writer nodes of the variable temp\_high.  
17 Control cells 101 and 121 are reader nodes of the variable temp\_high and also are writer  
18 nodes of the variable fan\_on. Fans 131-132 are reader nodes of the variable fan\_on.

19           Of course, many other applications and configurations are within the scope of the  
20 teachings of the present invention and the network described with reference to Figure 1 is  
21 merely exemplary.

22           It should be noted that multiple readers and multiple writers may be provided within  
23 a single connection without departure from the spirit and scope of the present invention.

1 Multiple readers are illustrated with reference to connection\_2 142. Multiple writers have  
2 not been illustrated by Figure 1. However, variation in which multiple writers are  
3 employed will be readily apparent to one of ordinary skill in the art.

4 Turning to Figure 2, an embodiment of the network of Figure 1 is illustrated in  
5 which each of cell 101, cell 121, temperature sensor 115, temperature sensor 116,  
6 temperature sensor 117, fan 131 and fan 132 are each coupled to communicate over  
7 common communication medium 201. The communication medium 201 may be, for  
8 example, twisted pair wiring, radio frequency, power lines, or other communication  
9 channels or multiple physical channels connected together with bridges and/or routers. In  
10 this embodiment, and in order to accomplish the connections illustrated by Figure 1,  
11 temperature sensor 115 must be configured to address and communicate with cell 101;  
12 temperature sensor 116 must be configured to address and communicate with cell 101 and  
13 cell 121; temperature sensor 117 must be configured to address and communicate with cell  
14 121; control cell 101 must be configured to address and communicate with fan 131; and  
15 control cell 121 must be configured to address and communicate with fan 132.

16 Of course, providing for such addressing may be and typically is a significant task.  
17 It is appreciated that each of control cells 101 and 121, temperature sensors 115-117 and  
18 fans 131-132 may be engineered, programmed and/or manufactured by different sources.  
19 Further, although the exemplary network is, in itself, complicated having 5 separate  
20 connections, 141-145, it can of course be imagined that other networks may be  
21 substantially more complicated having even hundreds or more connections. Therefore, the  
22 present invention implements methods and apparatus which allow for straightforward and  
23 efficient configuration of nodes in a network.

1           Turning now to Figure 3(a), a modified embodiment of the configuration of Figure  
2 2 is illustrated. In this embodiment, controller cells 101 and 121 have been removed from  
3 the configuration and each of temperature sensors 115-117 and fans 131-132 are illustrated  
4 as comprising nodes 301-305, respectively. These nodes are preferably of the type which  
5 are capable of sensing, communicating and controlling as have been described in the '690  
6 patent and which are shown in greater detail with reference to Figure 4. Thus, these nodes  
7 301-305 are capable of replacing certain control functions of the control cells 101 and 121,  
8 eliminating the need for separate control cells in the described embodiment. In the  
9 embodiment of Figure 3(a), and in order to accomplish the logical connections illustrated  
10 by Figure 1, node 301 must be configured to communicate with node 304; node 302 must  
11 be configured to communicate with nodes 304 and 305; and node 303 must be configured  
12 to communicate with node 305. Again it is important to note that the temperature sensors  
13 115-117 and fans 131-132 may be manufactured by different sources. It is preferable that  
14 the manufacturing sources are not required to have prior knowledge as to what devices their  
15 products will communicate with in a network. Thus, the manufacturer of temperature  
16 sensor 115 is preferably not required to be aware, during programming and manufacture of  
17 temperature sensor 115, whether temperature sensor 115 will be configured in a network to  
18 communicate with a controller cell, such as controller cell 101 (as shown in Figure 2), or to  
19 communicate directly with a fan, such as fan 131 (as shown in Figure 3(a)), or even with  
20 some other device (perhaps a heater, air conditioner, fire extinguishing equipment, etc.).  
21 Likewise, it is preferable that the manufacturer of fans 131-132 are similarly allowed to  
22 manufacture devices without requirement of prior knowledge as to the eventual uses of  
23 those devices in a network.

1           In order to allow for such flexibility in configuring networks and to allow for  
2 efficient communication between nodes in a network, the present invention provides  
3 network variables which may be used to facilitate standards of communication between  
4 nodes in the network.

5           Table I illustrates a temperature sensor control program as may be used to program  
6 nodes 301-303 coupled with temperature sensors 115-117. As can be seen, the program of  
7 Table I is written to communicate onto the medium 201 a network variable indicative of the  
8 state of temp\_in. The value of this variable may be, for example, used by a control  
9 program running on a control cell, such as control cell 101 or 121, or used directly by a  
10 control program running on a fan, such as fans 131-132.

11           Table II illustrates a fan control program which may be used for controlling a fan  
12 such as fans 131-132 by turning the fan on and off responsive to receiving changes in state  
13 of a network variable on\_off. As can be seen, the program of Table II is written to allow  
14 receiving from the medium 201 the network variable on\_off as a binary network variable  
15 regardless of the source (e.g., whether from a control cell such as control cell 101 or 121,  
16 or directly from a temperature sensor, such as temperature sensor 115-117).

17           Table III illustrates a binding set which connects temperature sensors 115-117 with  
18 fans 131-132 as illustrated by Figure 3(a). Figure 3(b) is provided to further an  
19 understanding of the binding set. As can be seen, the binding set provides for three  
20 connections illustrated as temp1\_controls 321, temp2\_controls 322, and temp3\_controls  
21 323 of Figure 3(b). The connection temp1\_controls connects the output variable  
22 temp\_high of temperature sensor 115 with the input variable fan\_on of fan\_1 131. The  
23 connection temp2\_controls connects the output variable temp\_high of temperature sensor



1 116 with the input variable fan\_on of both fan\_1 131 and fan\_2 132. Finally, the  
2 connection temp3\_controls connects the output variable temp\_high of temperature sensor  
3 117 with the input variable fan\_on of fan\_2 132.

4 It should be noted that although tables I, II and III illustrate programs which are  
5 useful for illustrative concepts of the present invention, an attempt has not been made to  
6 ensure these programs are syntactically correct. Rather, these programs are provided for  
7 the exemplary teaching of concepts of the present invention. It is understood from an  
8 examination of the programs of tables I and II that the program of Table I may write the  
9 variable temp\_high without regard to the eventual recipient of the variable and likewise the  
10 program of Table II may read the variable fan\_on without regard to the writer node of the  
11 variable. Thus, these programs work equally well in a network such as illustrated by  
12 Figure 2 including separate control cells 101 and 121 or in a network such as illustrated by  
13 Figure 3(a) which does not include such control cells. The binding set illustrated by Table  
14 III determines the relationship between the various nodes of the network. Table IV  
15 illustrates a binding set which may be used to establish connections in a network such as  
16 illustrated by Figure 2.

#### 17 A node of the present invention

18 Figure 4 illustrates a block diagram of a node such as nodes 301-305 as may be  
19 utilized by the present invention. The node 421 is coupled in communication with medium  
20 201 through control 411, clock and timer circuitry 412, and communication port 408. In  
21 addition, the node provides a general purpose I/O port 407 allowing for communication  
22 with various external devices. The node further comprises three separate processors  
23 404-406, a read only memory (ROM) 403, a random access memory 402, and an

1 EEPROM 401. The processors 404-406 are useful for executing programs such as the  
2 programs illustrated in Tables I and II, as well as other communication, control and  
3 operating programs. The ROM 403 may be useful for storing such programs. As will be  
4 seen, the EEPROM 401 may be useful for storing certain data values which, although  
5 configurable, are not subject to frequent changes in value. Each of the processors  
6 404-406, ROM 403, RAM 402, EEPROM 401, control 411, clock 412, I/O port 407, and  
7 communication port 408 are coupled in communication through internal address bus 410,  
8 internal data bus 420 and timing and control lines 430.

9 PROGRAMMING AND CONFIGURING

10 A NETWORK OF THE PRESENT INVENTION

11 Turning now to Figure 5, steps for programming and configuring a network of the  
12 present invention are illustrated. It should be noted that steps illustrated by Figure 5 are  
13 implemented in a development system which allows for development and management of  
14 networks such as may be implemented by the present invention. However, certain of these  
15 steps may also take place outside of the development environment (e.g., connection of  
16 network variables and binding). The development system is an integrated hardware and  
17 software environment that operates in conjunction with a host computer, an IBM PC/AT  
18 compatible in the currently preferred embodiment, allowing a manufacturer or other party to  
19 design and build components compatible for communication with a network of the present  
20 invention.

21 The development system includes an IBM PC/AT-compatible computer having an  
22 interface adapter card for coupling with a control processor located in a separate card cage.  
23 In addition to the control processor, the card cage may hold other cards designed to emulate

1 routing functions in a network and transceiver evaluation boards allowing evaluation of the  
2 physical interface with various media, e.g., twisted pair, power line, or radio frequency.

3 Initially certain hardware parameters are defined for each node in the network,  
4 block 501. This step includes naming or otherwise identifying the node, block 601. A  
5 node type is specified, block 602. In the development environment, the node type may be  
6 specified as the control processor, an emulator board, or a custom node type. The location  
7 of the node is then specified—the location specifies whether or not the node resides in a  
8 card cage and, if the node resides in a card cage, the card cage number and slot number,  
9 block 603. Next, the channel to which the node is connected is specified, block 604, and  
10 the channel's priority is specified, block 605. If the node has been assigned the priority  
11 privilege, then the node's priority is set at this time. Finally, certain hardware properties  
12 may be specified, block 605. Hardware properties may include model numbers for the  
13 node, clock rates, operating system revision levels, ROM size, RAM size, EEPROM size,  
14 RAM start address, and EEPROM start address. Finally, the hardware definitions are  
15 downloaded to the node, block 606.

16 Next, network and certain logical parameters are specified for each node, block  
17 502. Currently, this step involves specifying a node name, block 701, and then specifying  
18 a program file, block 702, and hardware device name, block 703 associated with the node.  
19 Hardware names were specified above in step 601. Program files will be discussed in  
20 greater detail below in connection with block 503. The definition of the node is then saved,  
21 block 704.

22 The development environment provides an editor for developing and editing  
23 program code, block 503, such as the code illustrated in tables I and II. The preferred

1 embodiment allows programming in the "C" language and, further, provides certain  
2 extensions to the "C" language which will be discussed in greater detail below. After  
3 developing program code, the programs are compiled, linked and loaded as executable  
4 programs, block 504, onto the nodes specified in definition of network and logical  
5 parameters, block 502.

6 Connections are then specified for the network, block 505. This step is better  
7 illustrated with reference to Figure 8(a). Initially, a connection name is entered (for  
8 example, the connection names specified in the binder script of Table III are  
9 temp1\_controls, temp2\_controls and temp3\_controls), block 801. In the preferred  
10 embodiment, the connection name is entered as a unique name having from one to 16  
11 characters consisting of letters, numbers and underscores; no spaces are allowed.

12 Next, a node name is selected, block 802. In the preferred embodiment, a list of  
13 defined nodes (i.e., nodes which have been previously defined as described in connection  
14 with block 502) is displayed and a valid node name may be selected from the displayed list.  
15 For example, the node temp\_sensor\_1 may be selected. After selecting a node name, block  
16 802, a network variable name is selected, block 803. Again, a list of network variable  
17 names for the selected node are preferably displayed and a network variable name is  
18 selected from the displayed list. For example, the network variable temp\_high may be  
19 selected.

20 After completing this process for a first node, a second node may be selected, block  
21 804. Again, a node list is preferably displayed and the second node is selected from the  
22 displayed node list. For example, the node fan\_1 may be selected. A network variable

1 associated with the second node is then selected, block 805, again preferably from a  
2 displayed list. Continuing the example, the selected network variable may be fan\_on.

3 Finally, certain parameters may be set, block 806. In the preferred embodiment,  
4 settable parameters include the retry count set to the maximum number of times the message  
5 will be sent, the retry timer for acknowledged services, and the repeat timer for  
6 unacknowledged/repeated messages. This aspect of the present invention will be discussed  
7 in greater detail below.

8 The connection is then added to a connection list using an add function, block 807.  
9 It is noted that if additional nodes are to be connected in the connection, they are specified  
10 in a similar manner to the first and second nodes after having specified the first and second  
11 nodes. An example of such a connection is illustrated in Table III as temp2\_controls which  
12 includes three nodes: temp\_sensor\_2, fan\_1 and fan\_2.

13 The process of Figure 8(a) is repeated for each desired connection. In the case of  
14 the binding set of Table III, the process is repeated three times: (1) once for the connection  
15 named temp1\_controls; (2) once for the connection named temp2\_controls; and (3) once for  
16 the connection named temp3\_controls. In the case of the binding set of Table IV, the  
17 process is repeated five times, once for each of connection\_1, connection\_2, connection\_3,  
18 connection\_4, and connection\_5.

19 In the preferred embodiment, the output of the connection process is a binary script  
20 file that provides commands to drive the subsequent binding process. In order to provide a  
21 textual version of what this binary file looks like, Table III and Table IV have been  
22 provided.

1           It is also within the power of one of ordinary skill in the art to develop a graphical  
2 user interface for drawing the connections between iconic representations of the nodes and  
3 creating a binder script based on such drawings.

4           Finally, the network variables are bound, block 506, to their respective nodes in  
5 order to allow communication within the connections defined during execution of the steps  
6 of Figure 8(a). The preferred method of binding network variables is described in greater  
7 detail with reference to Figure 8(b).

8           Initially, the list of connections developed during execution of the steps of Figure  
9 8(a) is read, block 821. Then, certain type checking and message constraint checking is  
10 performed for each connection, block 822. The type and message constraint checking  
11 includes the following checks:

- 12           (1) Ensure that there are at least two members in each connection;
- 13           (2) Ensure that there is at least one output member and one input member for each  
14           connection;
- 15           (3) In the preferred embodiment, no more than one input and one output network  
16           variable from the same node may appear in the same connection;
- 17           (4) A warning is given if polled output variables are not attached to at least one  
18           polled input;
- 19           (5) An estimate for message rates may be declared for network variables; a warning  
20           is given if the estimated message rates do not match for all members of a  
21           connection;

1           (6) Network variables may be synchronized or non-synchronized—a warning  
2           message is provided if synchronized variables are bound to non-synchronized  
3           variables;

4           (7) Network variables may be sent as authenticated—a warning is provided if  
5           some, but not all, members of a connection are declared as authenticated; and

6           (8) Variable types are checked field-by-field for size and sign type matching and for  
7           type definition matching. The currently preferred list of type definitions are  
8           provided in Table V.

9           After completing type and message rate constraint checking, the addressing mode  
10          for the network variable is determined, block 824. If there is only one destination (e.g.,  
11          temp1\_controls), subnet-node addressing is used using the subnetnode structure given  
12          below to create an entry in address table 901. Address table 901 will be discussed in  
13          greater detail below. The address entered in the address table 901 is the address of the  
14          destination node (e.g., in the case of temp1\_controls, the address of fan\_1 is entered in the  
15          address table of temp\_sensor\_1; conversely, the address of temp\_sensor\_1 is entered in the  
16          address table of fan\_1 to allow for such functions as polling of the current status of the  
17          network variable). The address table index entry 912 is set to correspond to the location in  
18          the address table 901 corresponding with the address entered in the address table 901. For  
19          example, in the case of the bind set of Table III, if the address of FAN\_1 is entered as a  
20          network address 913 in the address table 901 at entry 001, the address table index entry  
21          912 of the network variable table 903 corresponding to the network variable id assigned to  
22          the connection temp1\_controls is written with the address 001. In this way, whenever  
23          messages are sent on the network by temp\_sensor\_1 indicating the value of temp\_high has

1 been updated, the address table index is used to lookup the address of the destination node  
2 of such a message. A message is then sent, addressed to the destination node, including  
3 the network variable id and the new value. The destination node then receives the message  
4 and is able to update the value of its corresponding network variable "fan\_on".

5 If there is more than one destination node (e.g., temp2\_controls), group addressing  
6 is used using the above group address structure to create an entry in the address table 901.  
7 In the case of group addressing, a set of sender and destinations for the network variable is  
8 constructed. For example, in the case of the connection temp2\_controls, the set of sender  
9 and destinations includes temp\_sensor\_2, fan\_1 and fan\_2.

10 Other optimization steps are also provided by the binder of the preferred  
11 embodiment and are described in further detail below.

12 After determining an addressing mode, for each unique set of sender and  
13 destinations (unique without respect to which nodes are senders and which nodes are  
14 receivers), a group address is assigned to the set, block 825. The group address is  
15 propagated to the address table of each of the nodes in the set and stored in their respective  
16 address tables 901. The address table index value 912 for the entry corresponding to the  
17 group address is updated to index the address table 901 at the new entry. For example,  
18 group1 is defined to include temp\_sensor\_2, fan\_1 and fan\_2 and the group address is  
19 stored at entry 002 of the address table 901. Then, the address table index 912 for each of  
20 the three nodes temp\_sensor\_2, fan\_1 and fan\_2 is updated to point to the new address  
21 table entry.

22 For group address table entries, as described above, only the output network  
23 variable nodes actually set their network variable table entries to index the address table.



1 The nodes with input network variables will not index the address table. This allows the  
2 same network variable to reside in several network variable connections, and many  
3 network variable groups. When an incoming message arrives for one of these input  
4 network variables, the correct network variable table entry is found using the network  
5 variable ID (the software matches the network variable ID in the message to one in the  
6 table).

7 This "intersecting connection" ability makes the network variable concept more  
8 powerful by allowing the same variable to be updated by several groups, thus reducing  
9 both the overall network traffic and reducing network variable table space by sharing the  
10 same table entry among several connections.

11 Finally, a single network variable identification number (netvar\_ID) is assigned to  
12 each network variable in the connection, block 823. This may be better understood with  
13 reference to Figure 9 which illustrates a network variable table 902 having a network  
14 variable identification field 911 and an address table index field 912. Further, an address  
15 table 901 is illustrated having a network address field 913. It should be noted that these  
16 tables preferably reside in each individual node's EEPROM 401 and have additional fields  
17 in the preferred embodiment. However, for simplicity only the above-mentioned fields are  
18 illustrated in Figure 9. The network variable table is preferably of a structure as follows:

```
19
20 struct nv_table
21 {
22     unsigned priority:1; /*1=priority network variable, 0=non-priority nv*/
23     unsigned dir:1; /*direction 0=input, 1=output*/
24     unsigned idhi:6; /*network variable id, most significant bits*/
25     unsigned idlo; /*network variable id, least significant bits*/
26     unsigned ta:1; /*turnaround: 1= turnaround*/
27     unsigned st:2; /*service*/
28     unsigned auth:1; /*authenticated: 1=authenticated*/
29     unsigned addr:4; /*address table index*/
```

1           );

2           where the priority field indicates whether messages to update the network variable  
3 are to be sent as priority or non-priority messages; direction indicates the direction of the  
4 target ID, for example, a network variable update going from an output variable to an input  
5 variable would have the direction bit set to a 0; the network variable id is a 14 bit  
6 identification number allowing for a maximum of 16,384 unique network variables per  
7 domain in the network and corresponds to the network variable id 911 of Figure 9;  
8 turnaround indicates an output network variable may be connected to an input network  
9 variable of the same node; service indicates whether acknowledged or unacknowledged  
10 services is utilized; auth indicates whether message are authenticated prior to being accepted  
11 and processed by identifying the sender node through an authentication process; priority  
12 indicates whether messages are transmitted as priority or normal messages; and the address  
13 table index corresponds to address table index 912 and is an index into the address table  
14 901.

15           The Address Table preferably follows one of two formats given below; the first  
16 format is for group address table entries and the second format is for single destination  
17 node address table entries:

```
18
19     struct group
20     {     unsigned type:1;      /*indicates whether the structure is for a group or
21         single node*/
22         unsigned size:7;      /*group size (0 for groups > 128 members*/
23         unsigned domain:1;    /*domain reference*/
24         unsigned member:7;    /*node's member # (0 for groups > 128 members*/
25         unsigned rpttimer:4;  /*unacknowledged message service repeat timer*/
26         unsigned retry:4;     /*retry count*/
27         unsigned rcvtimer:4;  /*receive timer index*/
28         unsigned tx_timer:4;  /*transmit timer index */
29         int group;           /*group id*/
30     }
```

```
1
2 struct subnetnode
3 {   unsigned type;      /*indicates whether the structure is for a group or
4                               single node*/
5     unsigned domain:1;  /*domain reference*/
6     unsigned node:7;    /*node's #*/
7     unsigned rpttimer:4; /*unacknowledged message service repeat timer*/
8     unsigned retry:4;   /*retry count*/
9     unsigned rsvd:4;    /*reserved*/
10    unsigned tx_timer:4; /*transmit timer index */
11    int subnet;         /*subnet*/
12 }
```

13 It should be noted here that many of the present invention's concepts of groups,  
14 domains, subnets, acknowledged messages, etc. are described in greater detail with  
15 reference to United States Patent Application Serial Number 07/621,737 filed December 3,  
16 1990 titled Network Communication Protocol (the '737 application) which is assigned to  
17 the assignee of the present invention and which is incorporated herein by reference.

18 Continuing with the description of assigning a network variable id to a connection,  
19 block 823, the first unassigned network id is assigned to the connection and the network  
20 variable id is written to the network variable table 902 for each node using the network.  
21 Thus, in the above example, the network variable id 00000000000000<sub>2</sub> may be assigned to  
22 the connection temp1\_controls of Table III; the network variable id 00000000000001<sub>2</sub>  
23 may be assigned to the connection temp2\_controls of Table III; and the network variable id  
24 00000000000010<sub>2</sub> may be assigned to the connection temp3\_controls of Table III. It  
25 should be noted that network variable ids need not be unique domain wide, but only need  
26 be unambiguous within the nodes involved.

27 Certain advantages gained through use of network variables have now been  
28 described such as the ability to automatically generate network addressing schemes from  
29 application level connections. In addition to allowing for such ease of use, network

1 variables lead to generally smaller and less complicated application programs over other  
2 forms of network communication, such as prior art messaging techniques. Tables V and  
3 VI better illustrate differences between and certain advantages of use of the present  
4 invention's techniques over, for example, prior messaging techniques. Table V is a  
5 program written using network variables of the present invention. Table VI is a  
6 functionally equivalent program written using prior art messaging techniques. It is useful  
7 to note the comparative program statistics at the end of each program listing in which it is  
8 shown that the message program requires 626 bytes of ROM; 177 bytes of EEPROM; and  
9 1314 bytes of RAM. By way of comparison, the network variables program requires only  
10 335 bytes of ROM while using 231 bytes of EEPROM and only 1126 bytes of RAM.

11 SELF-IDENTIFYING STANDARD NETWORK VARIABLE TYPES

12 It is desirable to provide for interoperability between nodes in a network. To  
13 provide for such interoperability, it is necessary to assure compatibility between network  
14 variables in the various nodes of a network. To facilitate such compatibility, as one feature  
15 of the present invention, a list of standard network variable types is provided by the  
16 assignee of the present invention. The currently preferred list of standard network variable  
17 types is provided as Table VII. By utilizing the list of standard network variable types,  
18 nodes in the network may be interrogated for information on the network variables  
19 employed by the node and the network may then be configured based on this information.  
20 This process is better illustrated with reference to Figure 10.

21 Initially, a node which must be configured is coupled to the network medium, block  
22 1001. After the node is coupled to the medium, an address of the node may be determined  
23 through any number of methods. At least one of such methods is described with reference

1 to the '737 application. After having determined an address for the node, messages may be  
2 communicated to the node over the medium. In the preferred embodiment, a network  
3 management node is coupled to the medium which is useful for configuring the network.  
4 The network management node may communicate a command to the new node requesting  
5 its information on the network variables employed by the node, block 1002, or may  
6 alternatively read such information from a file which has already been placed in the network  
7 management node's memory.

8 In the preferred embodiment, in order to allow for the information to be stored in  
9 the network management node's memory, such information is made available for  
10 importation into the network management node via a binder interface file (BIF). The BIF  
11 file is a byproduct of the compilation process for each node, and contains all the  
12 information necessary to install the node on the network. This information is also referred  
13 to as the exposed interface of the node.

14 The BIF file for a new node may be provided to the network management node  
15 prior to installation of the new node on the network in order to allow a complete network  
16 database to be constructed in advance of, and separate from, the physical installation of the  
17 new node on the network. For example, the BIF file may be supplied to the network  
18 management node on diskette, over phone lines, or on through other computer readable  
19 media.

20 Information equivalent to the information stored in the BIF file is also preferably  
21 stored in the memory of the node. In this case the preferred embodiment confines the  
22 application writer to use of a list of standard network variable types when developing an  
23 application program designed to run on the node. The list of standard network variable

1 types used by the system of the preferred embodiment is enumerated in Table VII. Use of  
2 the list of standard network variables minimizes the required space for storing the exposed  
3 interface in the node's memory. Storing the exposed interface in the node's memory  
4 offers the advantage of allowing the information to be retrieved without need for the  
5 network management node to include a floppy disk drive or other device for receiving  
6 externally communicated computer readable information. However, absent the option of  
7 providing the BIF file over such an external interface, the node must be physically  
8 connected on the same network with the network management node prior to construction of  
9 the network database. In the preferred embodiment, both options are available and the  
10 choice of how the exported interface is imported into the network management node is left  
11 up to the node designer.

12 The file layout for the BIF file of the preferred embodiment is given in Table IX.  
13 An example of a BIF file is given in Table X. This exemplary BIF file has been generated  
14 for the program given in Table V.

15 As was mentioned, in the preferred embodiment nodes may utilize the standard  
16 network variable types in declaration of network variables. The information describing its  
17 network variables is communicated (or exposed) by the node to the network management  
18 node, block 1003, using standard messaging features of the network. It will be understood  
19 that in alternative embodiments, information describing other, non-standard variable types  
20 may also be communicated in a manner similar to communicating the information on  
21 standard network variables.

22 The network management node receives the exposed network variable information,  
23 block 1004, and may then use information, including the network variable type, in

1 verifying valid connections and in the binding process. Only network variables of identical  
2 types may be bound together in a single connection—thus, use of standard network  
3 variable types facilitates interoperability of nodes in the network as well as facilitating  
4 identification of network variables when a command is issued to expose the network  
5 variables of a node.

6 As one extension to the concept of self-identifying standard network types as just  
7 described, it is possible to include in the information transmitted responsive to receiving the  
8 command to expose network variable's text strings and even graphical icons to the network  
9 management node. Such information would make the nodes largely self-documenting.

#### 10 EXTENSIONS TO THE "C" LANGUAGE

11 The present invention has implemented certain extensions and features to the "C"  
12 programming languages to support use of network variables—these extensions include (1)  
13 the already disclosed declarations of variables as network variables and the ability to declare  
14 such variables as standard network variable types; (2) declaration and use of I/O objects;  
15 and (3) scheduling clauses. Each of these extensions will be discussed in greater detail  
16 below. It should be noted that although the extensions have been preferably implemented  
17 in the "C" programming language, the idea and concepts of these extensions are not limited  
18 to use in this programming language and, in fact, these ideas and concepts may readily be  
19 extended to other programming languages.

#### 20 Network variable declarations

21 As has been discussed, the present invention provides for declaration of network  
22 variables in C programs. Importantly, the declaration of network variables allows for  
23 declaring certain information for use by the above-described binding process. This process

1 is better understood with reference to Figure 11. Initially, a network variable is declared in  
2 a computer program intended to run on a node of the network of the present invention,  
3 block 1101. The preferred format for the declaration may be found with reference to Table  
4 VIII, below. As can be seen with reference to Table VIII, the declaration format preferably  
5 includes a set of parameters called `bind_info`. These parameters allow the network variable  
6 to be declared with an initial specification of protocol services. When the program is  
7 compiled, this initial information is output as part of the BIF file. The format of the BIF  
8 file may be found with reference to Table IX. As one option in declaring network  
9 variables, these parameters may be declared as configurable or non-configurable, block  
10 1102. In this way, a programmer programming a node may make an initial determination  
11 as to the state the parameter should normally be set to. For example, the programmer may  
12 determine in a typical configuration, a particular network variable should use acknowledged  
13 message services. However, the programmer may also allow a network administrator  
14 flexibility in configuring and optimizing the network by declaring the acknowledged  
15 parameter as configurable. The program is then compiled and a compiled output is  
16 produced in the conventional manner. In addition to producing the conventional outputs of  
17 a compiler, e.g., object code, the compiler of the present invention produces the  
18 above-mentioned BIF file which includes information on the declared network variables  
19 such as the state of parameters and whether or not such parameters are configurable, block  
20 1103.

21 During configuration of the network of the present invention, the state of these  
22 configurable parameters may be modified by the network administrator, block 1104. In the  
23 above-discussed example, the network administrator may determine the network will be



1 optimally configured if the variable declared as acknowledged is actually configured as  
2 unacknowledged and repeated. It is worthwhile to again refer to Figure 8(a) which  
3 illustrates, in addition to other steps in the connection process, the step of setting  
4 parameters for the connection, block 806. The parameters which are settable in this step of  
5 the configuration process are those parameters declared as configurable in the network  
6 variable declarations. These parameters are displayed on a display screen during the  
7 configuration process and may be modified by changing the state of the parameters on the  
8 display screen. For example, one of three states may be set to tell the network the type of  
9 service to be used for a network variable—unacknowledged, unacknowledged and  
10 repeated, and acknowledged. The authentication feature may be set to an on state in which  
11 message authentication is used or to an off state in which message authentication is not  
12 used. Also, network variable may be set to a priority state or a non-priority state indicating  
13 whether messages associated with the variable are to be sent as priority messages or as  
14 normal messages.

15 *Declaration and use of Objects*

16 Each node of the present invention comprises its own scheduler, timers, and logical  
17 I/O devices. The "C" programming language employed by the present invention provides  
18 access to these devices through use of predefined objects; namely, an event scheduler  
19 which handles task scheduling for the node, timer objects which provide both millisecond  
20 and second timers, and I/O objects which provide for declaration of a number of logical I/O  
21 devices. Importantly, once declared a logical link is created between the object name and  
22 the physical device and references may be made to the object name to gain access to the  
23 physical device.

1 Declaration and use of objects will be discussed in greater detail by referring to  
2 declaration of I/O objects. Each node of the network of the present invention has a number  
3 of built-in electrical interface options for performing input and output. Prior to performing  
4 input or output, a program must declare an I/O object which interfaces with one of eleven  
5 I/O pins on the node; three serial pins 441 and eight parallel pins 445. The eleven pins are  
6 referred to with the reserved pin names: IO\_0, IO\_1, IO\_2, IO\_3, IO\_4, IO\_5, IO\_6,  
7 IO\_7, IO\_8, IO\_9, and IO\_10. The declaration syntax for an I/O object and use of the  
8 eleven pins in the present invention is discussed further with reference to Table XI.

9 It is worthwhile to turn to Figure 12 to discuss this concept in somewhat greater  
10 detail. Initially, a program statement is coded to declare an I/O device giving a pin  
11 designation, a device type and a device name; when the program is compiled the declaration  
12 statement causes declaration of the I/O device, block 1201. Other parameters and the  
13 format of the declaration for an I/O device in the preferred embodiment may be found with  
14 reference to Table XI. Responsive to declaring the I/O device, the pins are configured to  
15 perform the function specified by the device type, block 1202. The device types of the  
16 preferred embodiment may be found with reference to Table XI.

17 This process is further illustrated with reference to the exemplary network variable  
18 program of Table V and the associated assembly language code resulting from a compile of  
19 the program given in Table XIV. As can be seen with reference to the program source code  
20 in Table V, two I/O devices are declared, IO\_0 as a bit output named MotorCtrl and IO\_5  
21 as a pulsecount input named pulseamps.

22 The specified device name is logically associated with the specified device to  
23 perform the designated I/O, block 1204. In this way, a reference may be simply made to

1 the device name to accomplish the designated I/O with necessity of continued reference to  
2 specific pins and without need for special coding to implement the desired device types. As  
3 can be seen with reference to Table XII, built-in functions are provided to allow  
4 communication with the I/O devices. One of the built-in functions may be used to perform  
5 the built-in function referring to the desired device name to specify a hardware device,  
6 block 1204. The desired I/O is then performed in accordance with the device type specified  
7 in the device declaration, block 1205.

### 8 Scheduling

9 Scheduling on a node in the present invention is event driven. When a given  
10 condition becomes true, a body of code termed a task associated with that condition is  
11 executed. In the preferred embodiment, scheduling is accomplished through "when"  
12 statements. The syntax of a when statement of the preferred embodiment is given in Table  
13 XIII. An example of a when statement is given below:

```
14     when (timer_expires (led_timer))    /* This line is the when clause    */  
15     {  
16         io_out (led, OFF);              /* This is the task - turn the led off */  
17     }
```

18 In the above example, when the application timer led\_timer expires, the body of  
19 code following the when statement is executed (and the LED is turned off). When  
20 statements provide for execution of a task (the bracketed code) when the condition specified  
21 (e.g., the led\_timer expires) evaluates to true. It is known in the art to provide structures in  
22 programming languages which allow for conditional execution of a task when a statement  
23 evaluates to true. However, in known systems which include a scheduling statement (a

1 when statement or the equivalent), the event which is evaluated is a predefined event. As is  
2 noted in Table XIII, the present invention provides for use of predetermined events in  
3 scheduling statements. However, as one important aspect of the present invention, events  
4 may also be any valid C expression. For example, the following statement may be coded  
5 in a system of the present invention:

```
6     when (x == 3)                /* This line is the when clause */  
7     {  
8         io_out (led, OFF);      /* This is the task - turn the led off */  
9     }
```

10 In this case, whenever the event  $x=3$  occurs, the LED is turned off. Of course,  
11 significantly more complicated C programming statements may be envisioned to define an  
12 event. As will be understood by one of ordinary skill in the art, allowing evaluation of any  
13 valid language expression to define an event offers significant flexibility over known  
14 systems. The present invention further allows for use of multiple when statements to be  
15 associated with a single task. For example:

43

```
1      when (powerup)          /* This line is one when clause */
2      when (reset)           /* This line is another when clause */
3      when (io_changes(io_switch)) /* This line is another when clause */
4      when (x = 3)           /* This line is another when clause */
5      {
6          io_out (led, OFF); /* This is the task - turn the led off */
7      }
```

8 In this case, when any of the above events evaluates to true, the task is executed—  
9 e.g., the LED is turned off.

10 Importantly, as one aspect of the present invention, I/O objects may be referred to  
11 in an event clause allowing improved ease of programming of the system of the present  
12 invention. For example, two methods may be used to determine if an input value is new:  
13 (1) the io\_update\_occurs event may be used, referring to the desired device in a when  
14 statement or the io\_in function may be used. The below two programs accomplish the  
15 same goal.

16

17

PROGRAM 1

18

```
IO_5 input pulsecount dev;
```

19

```
when (io_update_occurs (dev))
```

20

```
{
```

21

```
    /* perform the desired function */
```

22

```
}
```

23

```
1           PROGRAM 2
2 stimer t;
3 IO_5 input pulsecount dev;
4 when (timer_expires(t))
5     { io_in (dev);
6       if (input_is_new)
7         {
8           /* perform the desired function */
9         }
10    }
```

11 The particular method chosen will depend on the individual case; however, the  
12 above is exemplary of the flexibility and ease of use of the system of the present invention.

13 Further, as an additional feature of the present invention and as is described with  
14 reference to Table VIII, the present invention provides for two levels of when clauses,  
15 priority when clauses and normal when clauses. Using this feature, it is possible to handle  
16 events which must be dealt with on a priority basis.

#### 17 PERFORMANCE OPTIMIZATIONS PERFORMED

#### 18 BY THE BINDER OF THE PREFERRED EMBODIMENT

19 As was discussed above, when more than two nodes are used in a connection, the  
20 nodes may be recognized as a group and a group address may be assigned to the group of  
21 nodes.

22 The preferred embodiment also carries out other performance optimization routines  
23 to achieve minimal network traffic with resulting optimized response time. For example,

1 the binder determines optimal protocol service class and addressing allocation at the time of  
2 binding variables in order. Illustrative of this, with reference to Figure 3(b), three separate  
3 connections are shown, 321-323. Although this represents a typical optimal configuration,  
4 these three connections could be combined by the binder into a single group resulting in  
5 nodes sometimes receiving messages about network variable updates which are not used by  
6 those nodes. In such a configuration, although there are additional messages received by  
7 the nodes, no effect is seen by the application running on the node because the network  
8 variable messages include a 14-bit network variable identification. Therefore, nodes which  
9 have no need for a variable sent to them simply discard and, in the case of acknowledged  
10 service, acknowledge the message.

11 An advantage of grouping many nodes in a single group in the system of the  
12 preferred embodiment is that such grouping simplifies tasks for the binder process and  
13 further uses only one group address (the preferred embodiment is limited to 255 group  
14 addresses per domain).

15 Further, the binder of the present invention dynamically selects an optimal protocol  
16 class of service at the time of binding. This is done by first computing the number of  
17 messages it would take to complete a transaction on the first using acknowledged service  
18 (including the original message and the acknowledgements). (Note that this number is the  
19 group size which is known by the network variable binder process at the beginning of the  
20 connection process). Second, this number is compared with the repeat count for repeating  
21 message. If the repeat count is less than the group size, and none of the programs require  
22 acknowledged services (each program allows the config option for its network variables),

1 then the binder dynamically converts the service from acknowledged to unacknowledged  
2 repeat. This reduces network traffic, thus improving response time.

3

4

---

5 Thus, an improved communication network having capability for communication of  
6 information between nodes in the network is described.

7



TABLE I

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44

```

/*****
/**
/** Temperature sensor control program writes an output network variable
/** temp_high responsive to changes in temperature sensed by a thermostat
/**
/*****

/** I/O Declarations */
IO_1 input bit temp_in;

/** Network variables declaration */
network output boolean temp_high;

/** working variables declarations and initializations */
int on_threshold = 72;
int off_threshold = 68;

/*****
/**
/** Event driven code; update temp_high responsive to changes in temperature
/** input to the program by temp_in
/**
/*****

when (powerup)
when (reset)
{
  io_change_init (temp_in);
}

when (io_changes(temp_in))
{
  if (temp_in > on_threshold)
    temp_high = true;
  if (temp_in < off_threshold)
    temp_high = false;
}

```

TABLE II

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29

```

/*****/
/**                                                    **/
/** Fan control program reads an input network variable fan_on to control **/
/** turning a fan on and off using output bit fan_active. **/
/**                                                    **/
/*****/

/** I/O Declarations **/

IO_1 output bit fan_active;

/** Network variables declaration **/

network input boolean fan_on;

/*****/
/**                                                    **/
/** Event driven code; updates fan_active each time a change in state occurs **/
/** for the network variable fan_on **/
/**                                                    **/
/*****/

when (nv_update_occurs(fan_on))
{
    io_out(fan_active, fan_on);
}
    
```

TABLE III

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

```
#####  
#  
# This connection associates the temperature sensor control output variables #  
# (temp_high) to a fan control input variable (fan_on). Specifically, temperature #  
# sensor 1 is connected to fan 1; temperature sensor 2 is connected to fan 1 and #  
# and fan 2; and temperature sensor 3 is connected to fan 2. #  
# #  
#####  
@N (temp1_controls) #  
temp_sensor_1.temp_high /** writer **/  
fan_1.fan_on /** reader **/  
  
@N (temp2_controls) #  
temp_sensor_2.temp_high /** writer **/  
fan_1.fan_on /** reader **/  
fan_2.fan_on /** reader **/  
  
@N (temp3_controls) #  
temp_sensor_3.temp_high /** writer **/  
fan_2.fan_on /** reader **/  
  
#
```

TABLE IV

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35

```
#####  
#  
# This connection associates the temperature sensor control output variables #  
# (temp_high) to a control cell; the control cell is connected to fan control #  
# input variable (fan_on). Specifically, temperature sensor 1 is connected to #  
# control cell 1; temperature sensor 2 is connected to control cell 1 and control #  
# cell 2; temperature sensor 3 is connected to control cell 2; control cell 1 is #  
# connected to fan 1; and control cell 2 is connected to fan 2 #  
#  
#####  
  
@N (connection_1)  
temp_sensor_1.temp_high          /** writer **/  
cell_1.temp_high                 /** reader **/  
  
@N (connection_2)  
temp_sensor_2.temp_high          /** writer **/  
cell_1.temp_high                 /** reader **/  
cell_2.temp_high                 /** reader **/  
  
@N (connection_3)  
temp_sensor_3.temp_high          /** writer **/  
cell_2.temp_high                 /** reader **/  
  
@N (connection_4)  
cell_1.fan_on                    /** writer **/  
fan_1.fan_on                     /** reader **/  
  
@N (connection_5)  
cell_2.fan_on                    /** writer **/  
fan_2.fan_on                     /** reader **/  
  
#####
```

TABLE V

NETWORK VARIABLE PROGRAM EXAMPLE

```

1
2
3
4
5 #pragma receive_trans_count 8
6
7 /* This example has all the setpoint configuration local to this node. */
8 /* In this way, generic temperature and pressure sensors can be used */
9 /* which have no knowledge of the setpoints. They only report new temp */
10 /* values when the old one changes by a threshold value. Additionally, */
11 /* the temperature value can be reported to many nodes which can each */
12 /* use the temperature for their specific function -- even applying */
13 /* different set points to the temperature value. In the old study, */
14 /* actual temperature values were not sent on the network. Messages */
15 /* were sent for above high setpoint, at setpoint & below low setpoint. */
16 /* Since no temperature values were sent, the messages sent could only */
17 /* be used by this node -- defeating the value of a networked approach */
18 /* This division of function in the old study was done to save EEPROM */
19 /* in this node since storing the setpoints takes EEPROM. */
20
21 #define true 1
22 #define false 0
23 #define on true
24 #define off false
25
26 typedef signed int fahrenheit;
27 typedef signed int PoundsPerSqIn;
28
29
30 struct tempSetpoints
31 {
32     fahrenheit LowSet,
33     HighSet;
34 };
35
36 struct pressureSetpoints
37 {
38     PoundsPerSqIn LowSet,
39     HighSet;
40 };
41
42 /* EEPROM nodal configuration parameters: Minimum time the motor must */
43 /* remain on, minimum time the motor must remain off. Temperature & */
44 /* pressure setpoints. Location and device type, too!! */

```

```

1
2 config network input
3     signed long int      MinOffTime,
4                           MinOnTime;
5
6 config network input
7     struct tempSetpoints OutletWater,
8                           CndnsrHead,
9                           CoolAir;
10
11 config network input
12     struct pressureSetpoints CmprsrInltGas;
13
14 #pragma set_id_string "HVACComp"
15
16 /* Network variable declarations for temperature & pressure inputs */
17 /* used to decide when the motor should be turned on and off */
18
19 network input fahrenheit      OutletWaterTemp,
20                               CndnsrHeadTemp,
21                               CoolAirTemp;
22
23 network input PoundsPerSqIn  CmprsrGasPrsr;
24
25 network input boolean        BuildingCooling;
26
27 /* Network variable declarations used to report status to the HVAC */
28 /* system controller. Reported conditions are: node offline/online, */
29 /* motor on/off, and motor overloaded/O.K. These conditions are only */
30 /* reported when they change. */
31
32 network output boolean       MotorOn,
33                               MotorOverload,
34                               AmOnline;
35
36 /* Definitions of the Neuron® I/O pins. The previous study used an */
37 /* onchip AtoD to measure the current that the motor used. This version */
38 /* uses a $.50 external AtoD to convert current to a number of pulses */
39 /* over a 1 second interval. These pulses are accumulated via the on */
40 /* chip timer/counter block to determine the current the motor uses */
41
42 IO_0 output bit              MotorCtrl;
43 IO_5 input pulsecount       PulseAmps;
44
45 /* Timer declarations take no EEPROM space -- they are in RAM */
46

```

```

1  stimer      MinOffTimer,
2              MinOnTimer,
3              MotorMeasurementTimer;
4
5  /* number of pulses that equal the maximum amount of current the motor */
6  /* can draw. The cheap AtoD gives 0 to 255 pulses per second depending */
7  /* on the analog current value. */
8
9  const int    CompressorMotorMaxDraw=180,
10             MeasurementInterval=10;
11
12  int          strikes; /* motor overdraw counter */
13
14  /* now for some real code! initialization for reset, powerup and online */
15  /* events. Online means the node received a network management message */
16  /* to go online. */
17
18  void motor(boolean on_off_flag)
19  {
20      MotorOn = on_off_flag;
21      io out(MotorCtrl, on_off_flag);
22
23      if (on_off_flag == on)
24          MinOnTimer = MinOnTime;
25      else
26          MinOffTimer = MinOffTime;
27  }
28
29  void control_action()
30  {
31      if ( AmOnline                                &&
32          BuildingCooling                          &&
33          MinOffTimer                               == 0   &&
34          OutletWaterTemp                          >   OutletWater.HighSet &&
35          CndnsrHeadTemp                           <   CndnsrHead.LowSet  &&
36          CmprssrGasPrssr                           <   CmprssrInltGas.LowSet &&
37          CoolAirTemp                               >   CoolAir.HighCet
38      )
39      {
40          motor(on);
41      }
42  else
43      {
44          if ( BuildingCooling                      &&
45              MinOnTimer                           = 0   &&
46              OutletWaterTemp < OutletWater.LowSet &&

```

```

1      CndnsrHeadTemp > CndnsrHead.HighSet      &&
2      CmprssrGasPrssr > CmprssrInltGas.HighSet  &&
3      CoolAirTemp < CoolAir.LowSet
4      )
5      {
6          motor(off);
7      {
8      }
9  }
10
11 when (reset)
12 {
13     MotorOn = false;
14     MotorOverload = false;
15     AmOnline = true;
16     motor(off);
17
18     /* initialize all input variables so that other nodes */
19     /* don't have to all update this one before this one */
20     /* begins operation. */
21
22     OutletWaterTemp = OutletWater.LowSet;
23     CndnsrHeadTemp = CndnsrHead.LowSet;
24     CoolAirTemp = CoolAir.LowSet;
25     CmprssrGasPrssr = CmprssrInltGas.LowSet;
26
27     strikes = 0;
28
29     poll(BuildingCooling); /* ask the controller if AC is on */
30
31 }
32
33
34 when (online)
35 {
36     AmOnline = true;
37     motor(off);
38
39     /* if the motor was overloaded & and we just came back online */
40     /* perhaps someone repaired it */
41
42     MotorOverload = false;
43 }
44
45 when(offline)
46 {

```



```

1      AmOnline = false;
2      motor(off);
3      }
4
5      when ( nv update occurs )
6      {
7          control action();
8      }
9
10     when ( timer expires(MotorMeasurementTimer) )
11     {
12         MotorMeasurementTimer = MeasurementInterval;
13
14         if ( io_in(PulseAmps) > CompressorMotorMaxDraw)
15         {
16             if (++strikes >= 3)          /* motor is really overloaded */
17             {
18                 motor(off);
19                 MotorOverload = true;
20             }
21             else
22                 strikes = 0
23         }
24
25     Link Memory Usage Statistics:
26     ROM Usage: User Code & Constant Data    335 bytes
27
28     EEPROM Usage: (not necessarily in order of physical layout)
29     System Data & Parameters                72 bytes
30     Domain & Address Tables                 105 bytes
31     Network Variable Config Tables         42 bytes
32     User EEPROM Variables                   12 bytes
33     User Code & Constant Data              0 bytes
34     Total EEPROM Usage                      231 bytes
35
36     RAM Usage: (not necessarily in order of physical layout)
37     System Data & Parameters                549 bytes
38     Transaction Control Blocks             132 bytes
39     User Timers & I/O Change Events        12 bytes
40     Network & Application Buffers         424 bytes
41     User RAM Variables                      9 bytes
42     Total RAM Usage                        1126 bytes
43
44     End of Link Statistics
45

```

1

## TABLE VI

2

MESSAGING PROGRAM EXAMPLE

3

4

5

#pragma receive\_trans\_count 8

6

7

/\* This example has all the setpoint configuration local to this node. \*/

8

/\* In this way, generic temperature and pressure sensors can be used \*/

9

/\* which have no knowledge of the setpoints. They only report new temp \*/

10

/\* values when the old one changes by a threshold value. Additionally, \*/

11

/\* the temperature value can be reported to many nodes which can each \*/

12

/\* use the temperature for their specific function -- even applying \*/

13

/\* different set points to the temperature value. In the old study, \*/

14

/\* actual temperature values were not sent on the network. Messages \*/

15

/\* were sent for above high setpoint, at setpoint &amp; below low setpoint. \*/

16

/\* Since no temperature values were sent, the messages sent could only \*/

17

/\* be used by this node -- defeating the value of a networked approach \*/

18

/\* This division of function in the old study was done to save EEPROM \*/

19

/\* in this node since storing the setpoints takes EEPROM. \*/

20

21

#define true 1

22

#define false 0

23

#define on true

24

#define off false

25

26

/\* Add In some message codes \*/

27

28

#define CondensrTemp 0 /\* net in condensor temp \*/

29

#define CoolTemp 1 /\* net in air temp \*/

30

#define GasPress 2 /\* net in gas pressure \*/

31

#define BldCool 3 /\* net in building cooling stat \*/

32

#define MotOn 4 /\* net out cmprsr mot on \*/

33

#define MotOvld 5 /\* net out cmprsr mot overload \*/

34

#define NdOnline 6 /\* net out online \*/

35

#define Poll BldCool 7 /\* poll building status \*/

36

#define TimeMinOff\_c 8 /\* Config Msg code for time off \*/

37

#define TimeMinOn\_c 9 /\* Config Msg code for time on \*/

38

#define OutletH2O 10 /\* Net in H2O temperature \*/

39

#define CndnsrHd\_c 11 /\* cndsr head temp config \*/

40

#define ColdAir\_c 12 /\* Cold air temp config \*/

41

#define CompGasPress\_c 13 /\* gass pressure config \*/

42

#define OutletH2O\_c 14 /\* Config Msg code for water tmp \*/

43

```

1 typedef signed int fahrenheit;
2 typedef signed int PoundsPerSqIn;
3
4 struct tempSetpoints
5 {
6     fahrenheit LowSet,
7     HighSet;
8 };
9
10 struct pressureSetpoints
11 {
12     PoundsPerSqIn LowSet,
13     HighSet;
14 };
15
16 /* EEPROM nodal configuration parameters: Minimum time the motor must */
17 /* remain on, minimum time the motor must remain off. Temperature & */
18 /* pressure setpoints. Location and device type, too!! */
19
20 signed long int    MinOffTime,
21                  MinOnTime;
22
23 struct tempSetpoints    OutletWater,
24                        CndnsrHead,
25                        CoolAir;
26
27 struct pressureSetpoints    CmprsrInltGas;
28
29 #pragma set_id_string "HVAComp"
30
31 /* Network variable declarations for temperature & pressure inputs */
32 /* used to decide when the motor should be turned on and off */
33
34 fahrenheit    OutletWaterTemp,
35              CndnsrHeadTemp,
36              CoolAirTemp;
37
38 PoundsPerSqIn    CmprsrGasPrsr;
39
40 boolean    BuildingCooling;
41
42 /* Network variable declarations used to report status to the HVAC */
43 /* system controller. Reported conditions are: node offline/online, */
44 /* motor on/off, and motor overloaded/O.K. These conditions are only */
45 /* reported when they change.*/
46

```

```

1  boolean      MotorOn,
2              MotorOverload,
3              AmOnline;
4
5  /* Definitions of the Neuron® I/O pins. The previous study used an */
6  /* onchip AtoD to measure the current that the motor used. This version */
7  /* uses a $.50 external AtoD to convert current to an number of pulses */
8  /* over a 1 second interval. These pulses are accumulated via the on */
9  /* chip timer/counter block to determine the current the motor uses */
10
11  IO_0 output bit MotorCtrl;
12  IO_5 input pulsecount PulseAmps;
13
14  /* Timer declarations */
15
16  stimer      MinOffTimer,
17              MinOnTimer,
18              MotorMeasurementTimer;
19
20  /* number of pulses that equal the maximum amount of current the motor */
21  /* can draw. The cheap AtoD gives 0 to 255 pulses per second depending */
22  /* on the analog current value. */
23
24  const int    CompressorMotorMaxDraw=180,
25              MeasurementInterval=10;
26
27  int strikes; /* motor overdraw counter*/
28
29  /* Define all the message tags */
30
31  msg  tag    air_temp_in;
32  msg  tag    gas_press_in;
33  msg  tag    bldstate_in;
34  msg  tag    motIsOn_out;
35  msg  tag    motIsOvrld_out;
36  msg  tag    Im_onln_out;
37  msg  tag    getBldState;
38  msg  tag    config_msg;
39  msg  tag    water_temp_in;
40  msg  tag    cndsr_temp_in;
41
42  /* now for some real code! initialization for reset, powerup and online */
43  /* events. Online means the node received a network management message */
44  /* to go online. */
45
46
47

```

```

1
2 void motor(boolean on_off_flag)
3 {
4     MotorOn          = on_off_flag;
5     io_out(MotorCtrl, on_off_flag);
6     msg_out.tag      = motIsOn_out;
7     msg_out.code     = MotOn;
8     msg_out.data[0]  = MotorOn;
9     msg_send();
10
11     if (on_off_flag == on)
12         MinOnTimer = MinOnTime;
13     else
14         MinOffTimer = MinOffTime;
15 }
16
17 void control_action()
18 {
19     if ( AmOnline           &&
20         BuildingCooling    &&
21         MinOffTimer == 0    &&
22         OutletWaterTemp > OutletWater.HighSet &&
23         CndnsrHeadTemp < CndnsrHead.LowSet  &&
24         CmprssrGasPrssr < CmprssrInltGas.LowSet &&
25         CoolAirTemp > CoolAir.HighSet
26     )
27     {
28         motor(on);
29     }
30
31     else
32     {
33
34         if ( BuildingCooling           &&
35             MinOnTimer == 0            &&
36             OutletWaterTemp < OutletWater.LowSet &&
37             CndnsrHeadTemp > CndnsrHead.HighSet &&
38             CmprssrGasPrssr > CmprssrInltGas.HighSet &&
39             CoolAirTemp < CoolAir.LowSet
40         )
41         {
42             motor(off);
43         }
44     }
45
46

```

```

1
2 when (reset)
3 {
4     MotorOn           =           false;
5     MotorOverload    =           false;
6     AmOnline          =           true;
7
8     msg_out.tag       =           motIsOn_out;
9     msg_out.code      =           MotOn;
10    msg_out.data[0]   =           MotorOn;
11    msg_send();
12
13    msg_out.tag       =           motIsOvrlld_out;
14    msg_out.code      =           MotOvld;
15    msg_out.data[0]   =           MotorOverload;
16    msg_send();
17
18    msg_out.tag       =           Im_onln_out;
19    msg_out.code      =           NdOnline;
20
21    msg_out.data[0]   =           AmOnline;
22    msg_send();
23
24    motor(off);
25
26    /* initialize all input variables so that other nodes */
27    /* don't have to all update this one before this one*/
28    /* begins operation.*/
29
30    OutletWaterTemp   =           OutletWater.LowSet;
31    CndnsrHeadTemp    =           CndnsrHead.LowSet;
32    CoolAirTemp       =           CoolAir.LowSet;
33    CmprssrGasPrssr   =           Cmprssr~nLLGa~.~owSet;
34
35    strikes            =           0;
36
37    msg_out.tag       =           getBldState;
38    msg_out.code      =           Poll BldCool;
39    msg_out.service    =           REQUEST;
40    msg_send();
41 }
42
43
44
45
46

```

```
1  when(online)
2      {
3          AmOnline           =   true;
4          msg_out.tag        =   Im_onln_out;
5          msg_out.code       =   NdOnline;
6          msg_out.data[0]    =   AmOnline;
7          msg_send();
8          motor(off);
9
10         /* if the motor was overloaded & and we just came back online */
11         /* perhaps someone repaired it*/
12
13         MotorOverload      =   false;
14         msg_out.tag        =   motIsOvrlid_out;
15         msg_out.code       =   MotOvld;
16         msg_out.data[0]    =   MotorOverload;
17         msg_send();
18     }
19
20  when (offline)
21      {
22          AmOnline           =   false;
23          msg_out.tag        =   Im_onln_out;
24          msg_out.code       =   NdOnline;
25          msg_out.data[0]    =   AmOnline;
26          motor(off);
27      }
28
29  when ( msg_arrives(CondensrTemp) )
30      {
31          CndnsrHeadTemp = (msg_in.data[0]<<8) + msg_in.data[1];
32          control_action();
33      }
34
35  when(msg_arrives(CoolTemp))
36      {
37          CoolAirTemp =(msg_in.data[0]<<8) + msg_in.data[1];
38          control_action();
39      }
40
41  when ( msg_arrives(GasPress))
42      {
43          CmprsrGasPrsr = (msg_in.data[0]<<8) + msg_in.data[1];
44          control_action();
45      }
46
```

```
1  when ( msg_arrives(BldCool))
2      {
3          BuildingCooling    =  msg_in.data[0];
4          control_action();
5      }
6
7  when ( msg_arrives(OutletH2O))
8      {
9          OutletWaterTemp =  (msg_in.data[0]<<8) + msg_in.data[1];
10         control_action();
11     }
12
13  when ( msg_arrives(TimeMinOff_c))
14      {
15          MinOffTime    =  (msg_in.data[0]<<8) + msg_in.data[1];
16     }
17
18  when ( msg_arrives(TimeMinOn_c))
19      {
20          MinOnTime    =  (msg_in.data[0]<<8) + msg_in.data[1];
21     }
22
23  when ( msg_arrives(CndnsrHd_c))
24      {
25          CndnsrHead.LowSet    =  (msg_in.data[0]<<8) + msg_in.data[1];
26          CndnsrHead.HighSet   =  (msg_in.data[2]<<8) + msg_in.data[3];
27     }
28
29  when ( msg_arrives(ColdAir_c))
30      {
31          CoolAir.LowSet    =  (msg_in.data[0]<<8) + msg_in.data[1];
32          CoolAir.HighSet   =  (msg_in.data[2]<<8) + msg_in.data[3];
33     }
34
35  when ( msg_arrives(CompGasPress_c))
36      {
37          CmprsrInltGas.LowSet =  (msg_in.data[0]<<8) + msg_in.data[1];
38          CmprsrInltGas.HighSet =  (msg_in.data[2]<<8) + msg_in.data[3];
39     }
40
41  when ( msg_arrives(OutletH2O_c))
42      {
43          OutletWater.LowSet =  (msg_in.data[0]<<8) + msg_in.data[1];
44          OutletWater.HighSet =  (msg_in.data[2]<<8) + msg_in.data[3];
45     }
46
```



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40

```

when ( timer_expires(MotorMeasurementTimer) )
{
    MotorMeasurementTimer = MeasurementInterval;

    if ( io_in(PulseAmps) > CompressorMotorMaxDraw)
    {
        if (++strikes >= 3) /* motor is really overloaded */
        {
            motor(off);
            MotorOverload = true;
            msg_out.tag = motIsOvrd out;
            msg_out.code = MotOvld;
            msg_out.data[0] = MotorOverload;
            msg_send();
        }
        else
            strikes = 0;
    }
}

```

Link Memory Usage Statistics:  
ROM Usage: User Code & Constant Data      626 bytes

EEPROM Usage: (not necessarily in order of physical layout)

System Data & Parameters	72 bytes
Domain & Address Tables	105 bytes
Network Variable Config Tables	0 bytes
User EEPROM Variables	0 bytes
User Code & Constant Data	0 bytes
Total EEPROM Usage	177 bytes

RAM Usage: (not necessarily in order of physical layout)

System Data & Parameters	549 bytes
Transaction Control Blocks	132 bytes
User Timers & I/O Change Events	12 bytes
Network & Application Buffers	600 bytes
User RAM Variables	21 bytes
Total RAM Usage	1314 bytes

End of Link Statistics

41

TABLE VII  
STANDARD NETWORK VARIABLE TYPES

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46

#	Name	Quantity	Units	Range	Bits	Resolution
1	SNVT_amp	current	amps	-3,276 - 3276	16	0.1 ampere
2	SNVT_amp_mil	current	milliAmps	-3,276 - 3276	16	0.1 milliampere
3	SNVT_angle	phase/rotation	radians	0 - 65	16	0.001 radian
4	SNVT_angle_vel	angular velocity	radians/sec	3,276 - 3276	16	0.1 radians/sec
5	SNVT_char_ascii	character	character	0 - 255	8	1 character
6	SNVT_count	count,event	counts	0 - 65,535	16	1 count
7	SNVT_count_inc	incremental counts	counts	-32,768 - +32,767	16	1 count
8	SNVT_date_cal	date	YYYY,MM,DD	1-3000,0-12,0-31,	32	1 day
9	SNVT_date_day	day of Week	Enum list	M,Tu,W,Th,F,Sa,Su	8	N/A
10	SNVT_date_time	time of day	HH:MM:SS	00:00:00 to 23:59:59	24	1 second
11	SNVT_elec_kwh	energy, elec	Kilowatt-Hours	0 - 65,535	16	1 KWH
12	SNVT_elec_whr	energy, elec	watt-hours	0 - 6,553	16	0.1 WHR
13	SNVT_flow_mil	flow	milliliters/sec	0 - 65,535	16	1ml/s
14	SNVT_length	length	meters	0 - 6,553	16	0.1m
15	SNVT_length_kilo	length	kilometers	0 - 6,553	16	0.1km
16	SNVT_length_micr	length	microns	0 - 6,553	16	0.1km
17	SNVT_length_mil	length	millimeters	0 - 6,553	16	0.1mm
18	SNVT_lev_contin	level, contin	percent	0 - 100%	8	.5%
19	SNVT_lev_disc	level, discrete	Enumerated list		8	N/A
20	SNVT_mass	mass	grams	0 - 6,553	16	0.1g
21	SNVT_mass_kilo	mass	kilograms	0 - 6,553	16	0.1kg
22	SNVT_mass_mega	mass	metric tons	0 - 6,553	16	0.1 tone
23	SNVT_mass_mill	mass	milligrams	0 - 6,553	16	0.1mg
24	SNVT_power	power	watts	0 - 6,553	16	0.1 watt
25	SNVT_power_kilo	power	watts	0 - 6,553	16	0.1 kwatt
26	SNVT_ppm	concentration	ppm	0-65,535	16	1ppm
27	SNVT_press	pressure	pascals	-32,768 - 32,767	16	1 pascal
28	SNVT_press_psi	pressure	lbs/sq-in	-3,276 - 3,276	16	0.1 psi
29	SNVT_res	resistance	Ohms	0 - 6,553	16	0.1 Ohm
30	SNVT_res_kilo	resistance	kiloOhms	0 - 6,553	16	0.1 kilo-Ohm
31	SNVT_sound_db	sound Level	dBspl	-327 - 327	16	0.01 dB
32	SNVT_speed	speed	meters/second	0 - 655	16	0.01m/s
33	SNVT_speed_kmh	speed	km/hour	0 - 655	16	0.01 km/h
34	SNVT_state_supr	sensor state	Enumerated list		8	N/A
35	SNVT_str_asc	char string	ASCII characters(s)	30 characters	248	N/A
36	SNVT_str_int	char string	Int'l char set (s)	14 characters	248	N/A

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16

#	Name	Quantity	Units	Range	Bits	Resolution
37	SNVT_telecom	phone state	Enumerated list		8	N/A
38	SNVT_temp	temperature	Celsius	-3,276 - +3,276	16	0.1 degree
39	SNVT_time_passed	elapsed time	HH:MM:SS:LL	0 - 65,536	48	0.001 sec
40	SNVT_vol	volume	liters	0 - 6,553	16	0.1 liter
41	SNVT_vol_kilo	volume	kiloliters	0 - 6,553	16	0.1 kiloliter
42	SNVT_vol_mil	volume	milliliters	0 - 6,553	16	0.1 milliliter
43	SNVT_volt	voltage	volts	-3,276 - 3,276	16	0.1 volt
44	SNVT_volt_dbmv	voltage	dB microvolts	-327 - 327	16	0.01 db uv dc
45	SNVT_volt_kilo	voltage	kilo volts	-3,276 - 3,276	16	0.1 kilovolt
46	SNVT_volt_mil	voltage	millivolts	-3,276 - 3,276	16	0.1 millivolt

## TABLE VIII

NETWORK VARIABLE DECLARATION1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22

The preferred syntax for declaration of a network variable is as follows:

network input | output [netvar modifier] [class] type [bind\_info (fields)] identifier;

where:

netvar modifier are the following optional modifiers which can be included in the declaration of a network variable:

*sync / synchronized* — specifies that all values assigned to this network variable must be propagated, and in their original order. However, if a synchronous network variable is updated multiple times within a single critical section, only the last value is sent out. If this keyword is omitted from the declaration, the scheduler does not guarantee that all assigned values will be propagated. For example, if the variable is being modified more rapidly than its update events can be processed, the scheduler may discard some intermediate data values. However, the most recent data value for a network variable will never be discarded.

*polled* — is used only for output network variables and specifies that the value of the output network variable is to be sent only in response to a poll request from a node which reads the network

1 variable. When this keyword is omitted, the value is propagated  
2 over the network every time the variable is assigned a value.  
3  
4 **class** Certain classes of storage may be specified for network variables.  
5 Specifically, the following keywords may be entered in the network  
6 variable declaration statement:  
7 *const* — specifies that the network variable may not be changed by  
8 the application program;  
9 *eprom* — allows the application program to indicate the value of  
10 the network variable is to be preserved across power outages. In the  
11 preferred embodiment, variables declared with this storage class are  
12 stored in the eeprom 401. EEPROM variables have a limited  
13 capability to accept changes before the EEPROM can no longer be  
14 guaranteed to operate properly. Therefore, initializers for the  
15 eeprom class take effect when the program is loaded and not each  
16 time the program is started. Reloading a program has the effect of  
17 reinitializing all eeprom variables.  
18 *config* — specifies a const network variable in EEPROM that can  
19 be changed only by a network management node node. This class  
20 of network variable is typically used for application configuration by  
21 a network manager.  
22

1	type	Network variable typing serves two purposes: (1) typing ensures
2		proper use of the variable in the compilation, and (2) typing ensures
3		proper connection of network variables at bind time. Network
4		variables may be declared as any one of the following types:
5		[signed] long integer
6		unsigned long integer
7		signed character
8		[unsigned] character
9		[signed] [short] integer
10		unsigned [short] integer
11		enumerated lists
12		structures and unions of the above types
13		standard network variable types (see Table VII)
14		
15	bind_info (fields)	The following optional fields may be included in the declaration of a
16		network variable; the compiler builds the BIF file utilizing
17		information declared in these fields and the information in the BIF
18		file is used for binding the network variable inputs and outputs. The
19		fields are each optional and may be specified in any order.
20		<i>offline</i> — is used to signal to the bind process that a node should be
21		taken offline before an update can be made to the network variable.
22		This option is commonly used with the config class network
23		variable.

1                    *bind* | *bind(var\_name)* — specifies whether the network variable is  
2                    bound to network variables on other nodes (the usual case) or to a  
3                    network variable on the same node. The default is *bind* which  
4                    indicates that the network variable can be bound to network  
5                    variables on other nodes. The other form, *bind (var\_name)* allows  
6                    binding an output to an input on the same node. The *var\_name* is  
7                    the name of another network variable on the same node. It should be  
8                    noted that this option has been omitted from the currently preferred  
9                    embodiment of the present invention.

10                   *unackd* | *unackd\_rpt* | *ack [(config | nonconfig)]* — tells the  
11                   protocol layer of the network management software of the present  
12                   invention the type of service to use for the variable. An  
13                   unacknowledged (*unackd*) network variable uses minimal network  
14                   resources to propagate its values to other nodes. As a result,  
15                   propagation failures are more likely to occur, and such failures are  
16                   not detected by the node. This class is typically used for variables  
17                   which are updated on a frequent, periodic basis, where the loss of  
18                   an update is not critical, or in cases where the probability of a  
19                   collision or transmission error is extremely low. The *unackd\_rpt*  
20                   class of service is used when a variable is sent to a large group of  
21                   other nodes; with this class the message is sent multiple times to  
22                   gain a greater probability of delivery. Acknowledged (*ackd*) service  
23                   provides for receiver node acknowledged delivery with retries. The

1 keyword *config*, indicates the service type can be changed at the  
2 time connections are specified for the network variable. The  
3 keyword *nonconfig* indicates the service type cannot be changed at  
4 configuration time.

5 *authenticated* | *nonauthenticated* [(*config* | *nonconfig*)] — specifies  
6 whether the network variable requires use of an authentication to  
7 verify the identity of the sender node. The *config* | *nonconfig*  
8 keywords specify whether the authentication designation is  
9 configurable. The default in the system of the preferred embodiment  
10 is *nonauth (config)*.

11 *priority* | *nonpriority* [(*config* | *nonconfig*)] — specifies whether  
12 the network variable receives priority or not. The keywords *config*  
13 | *nonconfig* specify whether priority is configurable. The default is  
14 *nonpriority (config)*.

15 *rate\_est (const\_expression)* — specifies the estimated average  
16 message rate, in tenths of messages per second, that an associated  
17 network variable is expected to transmit or receive. This value  
18 assists the network administrator in configuring the network.

19 *max\_rate\_est (const\_expression)* — specifies the estimated  
20 maximum message rate, in tenths of messages per second, that the  
21 associated network variable is expected to transmit or receive. This  
22 value assists the network administrator in configuring the network.  
23



## TABLE IX

BIF FILE FORMAT

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

The Binder Interface File (BIF) format comprises a number of records—one record per network variable and one record per message tag plus some overhead records. The format is designed to be concise with few lines of physical lines per record. The format of the file generally allows for the following record types: (1) File Header comprising timestamp and other general information (one record); (2) Global information comprising general information of indicating general information about the node and the application program running on the node; and (3) Network variable and message tag records for each network variable and message tag comprising information about the network variable or message tag.

Importantly, network variables and message tags may require differing amounts and formats of information. Therefore, as one aspect of the present invention, a record structure has been developed to allow efficient storage of the differing required information and efficient retrieval of the records. In addition, in order to conserve storage, the present invention discloses an encoding scheme to encode numeric information present in the records.

In general, string fields contain an asterisk if they are not applicable. Integer fields contain a zero. The first record in the file is a header which comprises three lines of commentary and copyright notice text and a timestamp. Following this header is one blank line followed by global information used by the binder process.

1           Global Information

2           The first global value line is a Program ID comprising eight 2-digit hexadecimal values,  
3 separated by colons. The second global value line comprises several numeric fields separated by  
4 spaces. The fields are defined in order as follows:

- 5           • Either a 1 or a 2 which specifies the number of domains.
- 6           • The number of address table 901 slots in the range of decimal 1-15.
- 7           • Either a 0 or a 1. Indicates whether the node application program handles incoming  
8 messages.
- 9           • The number of network variables defined by the application program in the range of 0 to 62.
- 10          • The number of message tags defined by the application program in the range 0 to 15.
- 11          • The number of network input buffers (encoded, see below).
- 12          • The number of network output buffers (encoded, see below).
- 13          • The number of priority network output buffers (encoded, see below).
- 14          • The number of priority application output buffers (encoded, see below).
- 15          • The number of application output buffers (encoded, see below).
- 16          • The number of application input buffers (encoded, see below).
- 17          • The size of a network input buffer (encoded, see below).
- 18          • The size of a network output buffer (encoded, see below).
- 19          • The size of an application input buffer (encoded, see below).
- 20          • The size of an application output buffer (encoded, see below).

21          The third line is used for node-specific parameters and has not been fully defined in the  
22 currently preferred embodiment. The fourth and following lines are optional and may include a node  
23 documentation string which may be transmitted to the network management node for documenting,

1 for example, the general function of the node. If not supplied, these nodes comprise a single  
 2 asterisk. If supplied, these lines each begin with a double-quote character which is not included as  
 3 part of the documentation string. Multiple lines are concatenated without any intervening characters.  
 4 There is no end double quote. The global values section ends with a blank line.

5 As noted above, buffer sizes and count fields are encoded. The encoded values allow  
 6 selected values, given below, to be stored in a nibble, thus reducing the size of the database. In the  
 7 preferred embodiment, buffer sizes must be one of 20, 21, 22, 24, 26, 30, 34, 42, 50, 66, 82, 114,  
 8 146, 210 or 255 (i.e., 15 allowed buffer size values where the buffer sizes are given in bytes);  
 9 non-priority buffer counts must be one of 1, 2, 3, 5, 7, 11,  
 10 15, 23, 31, 47, or 63 (i.e., 11 allowed buffer size values). Priority buffer counts must be one of 0,  
 11 1, 2, 3, 5, 7, 11, 15, 23, 31, 47, or 63 (i.e., 12 allowed buffer size values).

12 In order to represent these values in a single nibble (4 bits), the following formulas are used  
 13 to transform the nibble value (n) to the above values:

14 for buffer sizes:  $2^{n/2} + (n \& 1) * 2^{n/2-1} + 18$  (except where  $n = 0$ ; size = 255); and

15 for count (priority and non-priority):  $2^{n/2} + (n \& 1) * 2^{n/2-1} - 1$

16 where n is the nibble value and the & symbol indicates a logical AND function between the  
 17 four bit n value and 0001 (e.g., for  $n = 3$ , the result of  $n \& 1$  is  $0011 \& 0001 = 0001$  or 1, for  $n=2$ ,  
 18 the result is 010; in general for any even number n, the value of this function will be 0 and for any  
 19 odd number n, the value will be 1). Also, in the above equations, integer arithmetic is used;  
 20 therefore, where fractional values result in the computation of a value (e.g.,  $n/2$  where  $n = 1$ ), the  
 21 fractional values are rounded down to the next lowest integer (e.g., for  $n=1$ ,  $n/2 = 1/2$ , is rounded  
 22 down to 0). Use of the above formula, as opposed to, for example a table lookup routine, leads to  
 23 decreased requirements for static memory.

1           Network Variable and Message Tag Records

2           Zero or more records are produced which correspond to the network variables and message  
3 tag definitions in the program. Message tag records begin with the word "TAG"; Network variable  
4 messages begin with "VAR". Following the "TAG" or "VAR" identifier is a string of at maximum  
5 16 characters which is the name of the tag or variable. Next,  
6 there is a decimal number (0-61 for network variables; 0-14 for message tags) which translates the  
7 name into an internal program index for the object code. Finally, there are two rate estimates, each a  
8 decimal number from 0-255, which are the rate\_est and max\_rate\_est, respectively, in units of tenths  
9 of a message per second.

10           The second line of each record corresponds to the bind\_info fields and other numeric fields in  
11 the order and as defined below:

12	<u>Field</u>	<u>Values</u>
13	offline specifier	0 or 1
14	bindable specifier	0 or 1
15	bind target index	0-61 (63 if no bind target is specified)
16	direction	0=input, 1=output
17	service type	0=acknowledged, 1=unackd_rpt, 2=unackd
18	service type configurable?	1=yes, 0=no
19	authenticated?	1=yes, 0=no
20	authenticated configurable?	1=yes, 0=no
21	priority	1=yes, 0=no
22	priority configurable?	1=yes, 0=no
23	polled	1=yes, 0=no

1           synchronized                           1=yes, 0=no

2           config                                1=yes, 0=no

3           The third and any subsequent lines optionally contain variable documentation in the same  
4 format as the node documentation described above. If no documentation is supplied, a single asterisk  
5 is used.

6           For network variables, the remaining lines following any documentation comprise the  
7 following information. Message tags do not require this information.

8           The first line following the documentation lines is a header in which a first field indicates  
9 whether the variable is a standard network variable type; if so, the remaining fields are ignored and  
10 there are no more lines in the record. The format of the line, in order of the fields, is as follows:

<u>Field</u>	<u>Values</u>
Standard network variable type number	1-255 (0 if not a standard type)
First typedef name used in the definition	maximum length 16 characters, * if none
Number of elements in the type	1 unless structured or union, 256 max

15           There is one additional line per element, (where the number of elements was given  
16 immediately above). The format of these lines is as follows, in order of the fields presented:

<u>Field</u>	<u>Values</u>
Basic Type	0=char, 1=integer, 2=long, 3=bitfield, 4=union
Bitfield Offset	0-7, 0 if not applicable
Bitfield / union size	1-7 for bitfield; 1-31 for union; 0 if not applicable
Signedness	0=unsigned, 1=signed
Array bound	1-31, 0 if not an array

1

## TABLE X

2

BIF FILE FOR PROGRAM OF TABLE V

3

4

File: node\_31\_right.bif generated by APC Revision 0.99

Copyright (c) 1990 Echelon Corporation

Run on Mon Feb 4 10:31:40 1991

8

9

48:56:41:43:6F:6D:70:00

10 2 15 0 14 0 3 3 3 3 3 3 11 9 4 2

11 \*

12 \*

13

14 VAR MinOffTime 000

15 0 1 6 3 0 0 1 0 1 0 1 0 0 1

16 \*

17 0 \* 1

18 2 0 0 1 0

19 VAR MinOnTime 100

20 0 1 6 3 0 0 1 0 1 0 1 0 0 1

21 \*

22 0 \* 1

23 2 0 0 1 0

24 VAR OutletWater 200

25 0 1 6 3 0 0 1 0 1 0 1 0 0 1

26 \*

27 0 \* 2

28 1 0 0 1 0

29 1 0 0 1 0

30 VAR CndnsrHead 300

31 0 1 6 3 0 0 1 0 1 0 1 0 0 1

32 \*

33 0 \* 2

34 1 0 0 1 0

35 1 0 0 1 0

36 VAR CoolAir 400

37 0 1 6 3 0 0 1 0 1 0 1 0 0 1

38 \*

39 0 \* 2

40 1 0 0 1 0

41 1 0 0 1 0

42 VAR CmprsrInltGas 500

43 0 1 6 3 0 0 1 0 1 0 1 0 0 1

```

1 *
2 0* 2
3 10010
4 10010
5 VAR OutletWaterTemp 600
6 01630010101000
7 *
8 0* 1
9 10010
10 VAR CndnsrHeadTemp 700
11 01630010101000
12 *
13 0* 1
14 10010
15 VAR CoolAirtemp 800
16 01630010101000
17 *
18 0* 1
19 10010
20 VAR CmprsrGasPrsr 900
21 01630010101000
22 *
23 0* 1
24 10010
25 VAR BuildingCooling 1000
26 01630010101100
27 *
28 0 boolean 1
29 10010
30 VAR MotorOn 11000
31 01631010101000
32 *
33 0 boolean 1
34 10010
35 VAR MotorOverload 1200
36 01631010101000
37 *
38 0 boolean 1
39 10010
40 VAR AmOnline 1300
41 01631010101000
42 *
43 0 boolean 1
44 10010
    
```

## TABLE XI

I/O DEVICE DECLARATION

1

2

3

4

Each I/O device is declared in the application program as an external "device name".

5

The syntax for such declaration is as follows:

6

`<pin> <type> [<assign>] <device-name> [=<initial-output-level>];`

7

where `<pin>` is one of the eleven reserved pin names: IO\_0, IO\_1, IO\_2, IO\_3, IO\_4,

8

IO\_5, IO\_6, IO\_7, IO\_8, IO\_9, and IO\_10;

9

10

`<type>` is one of the following types, may specify the indicated pins and is subject

11

to the indicated restrictions:

12

(1) *output bit* — Used to control the logical output state of a single pin, where 0 equals low and 1 equals high; may specify any pin IO\_0 to IO\_10 and is unrestricted.

14

(2) *input bit* — Used to read the logical output state of a single pin, where 0 equals low and 1 equals high; may specify any pin IO\_0 to IO\_10 and is unrestricted.

16

(3) *[output] bitshift [numbits (<expr>)] [clockedge ({ + / - })] [kbaud (<expr>)]* —

17

Used to shift a data word of up to 16 bits out of the node. Data is clocked out by an

18

internally generated clock. *numbits* specifies the number of bits to be shifted; *clockedge*

19

specifies whether the data is stable on the positive going or negative going edge; and *kbaud*

20

specifies the baud rate. Requires adjacent pin pairs; the pin specification specifies the low

21

numbered pin of the pair and may be IO\_0 through IO\_6 or IO\_8 or IO\_9.

22

(4) *[input] bitshift [numbits (<expr>)] [clockedge ({ + / - })] [kbaud (<expr>)]* —

23

Used to shift a data word of up to 16 bits into the node. Data is clocked in by an internally



1 generated clock. *numbits* specifies the number of bits to be shifted; *clockedge* specifies  
2 whether the data is read on the positive going or negative going edge; and *kbaud* specifies  
3 the baud rate. Requires adjacent pin pairs; the pin specification specifies the low numbered  
4 pin of the pair and may be IO\_0 through IO\_6 or IO\_8 or IO\_9.

5 (5) *[output] frequency [invert] [clock (<expr>)]* — This device type produces a  
6 repeating square wave output signal whose period is a function of an output value and the  
7 selected clock, *clock (<expr>)*, where *clock (<expr>)* specifies one of 8 clocks provided by  
8 the node. Must specify IO\_0 or IO\_1. The mux keyword (see below) must be specified for  
9 IO\_0 and the ded keyword (see below) must be specified for IO\_1.

10 (6) *[output] triac sync <pin> [invert] [clock (<expr>)]* — This device type is used  
11 to control the delay of an output pulse signal with respect to an input trigger signal, the  
12 *sync* input. Must specify IO\_0 or IO\_1. The mux keyword (see below) must be specified  
13 for IO\_0 and the ded keyword (see below) must be specified for IO\_1. If IO\_0 is  
14 specified, the sync pin must be IO\_4 through IO\_7; if IO\_1 is specified, the sync pin must  
15 be IO\_4.

16 (7) *[output] pulsewidth [invert] [clock (<expr>)]* — This device type is used to  
17 produce a repeating waveform which duty cycle is a function of a specified output value  
18 and whose period is a function of a specified clock period. Must specify IO\_0 or IO\_1.  
19 The mux keyword (see below) must be specified for IO\_0 and the ded keyword (see  
20 below) must be specified for IO\_1.

21 (8) *input pulsecount [invert]* — This device type counts the number of input edges  
22 at the input pin over a period of 0.839 seconds. Must specify IO\_4 through IO\_7.

1           (9) *output pulsecount [invert] [clock (<expr>)]* — This device type produces a  
2 sequence of pulses whose period is a function of the specified clock period. Must specify  
3 IO\_0 or IO\_1. The mux keyword (see below) must be specified for IO\_0 and the ded  
4 keyword (see below) must be specified for IO\_1.

5           (10) *[input] ontime [invert] [clock (<expr>)]* — This device type measures the high  
6 period of an input signal in units of the specified clock period. Must specify IO\_4 through  
7 IO\_7.

8           (11) *{output | input } serial [baud (<expr>)]* — This device type is used to transfer  
9 data using an asynchronous serial data format, as in RS-232 communications. Output  
10 serial must specify IO\_10; input serial must specify IO\_8.

11           (12) *parallel* — This device type is used to transfer eight bit data words between  
12 two nodes across an eleven pin parallel bus. This is a bidirectional interface. Requires all  
13 pins and must specify IO\_0.

14           (13) *neurowire select <pin> [kbaud (<expr>)]* — This device type is used to  
15 transfer data using a synchronous serial data format. Requires three adjacent pins and must  
16 specify IO\_8. The select pin must be one of IO\_0 through IO\_7.

17           (14) *[input] quadrature* — This device type is used to read a shaft or positional  
18 encoder input on two adjacent pins. Requires adjacent pin pairs; the pin specification  
19 specifies the low numbered pin of the pair and may be IO\_0 through IO\_6 or IO\_8 or  
20 IO\_9.

21           (15) *[input] period [invert] [clock (<expr>)]* — This device type measures the total  
22 period from negative going edge to negative going edge of an input signal in units of the  
23 specified clock period. Must specify IO\_4 through IO\_7.

1           (16) *{output} oneshot [invert] [clock (<expr>)]* — This device type produces a  
2 single output pulse whose duration is a function of a specified output value and the selected  
3 clock value. Must specify IO\_0 or IO\_1. The mux keyword (see below) must be specified  
4 for IO\_0 and the ded keyword (see below) must be specified for IO\_1.

5           (17) *{output / input} nibble* — This device type is used to read or control four  
6 adjacent pins simultaneously. Requires four adjacent pins; the pin specifies denotes the  
7 lowest number pin of the quartet and may be pin IO\_0 through IO\_4.

8           (18) *{output / input} byte* — This device type is used to read or control eight pins  
9 simultaneously. Requires eight adjacent pins; the pin specification denotes the lowest  
10 number pin and must be IO\_0.

11           (In general, pins may appear in a single device declaration only; however, a pin  
12 may appear in multiple declarations if the types belong to the set of {bit, nibble and byte});

13  
14           where <assign> is one of "mux" which indicates the device is assigned to a  
15 multiplexed timer counter circuit or "ded" which indicates the device is assigned to a  
16 dedicated timer counter circuit; and

17  
18           where <initial-output-state> is a constant expression used to set the output pin of  
19 the channel to an initial state at initialization time (e.g., when the application program is  
20 reset).

21

1

## TABLE XII

2

ACCESS TO I/O DEVICES VIA BUILT IN FUNCTIONS

3

4

To access one of the I/O devices (after declaring it as shown above), the application programmer merely calls one of the built-in functions defined below. These built-in functions appear syntactically to be nothing more than procedure calls. However, these procedure calls are not be defined as external functions to be linked in. Instead, these procedure names are "known" to the compiler, and the compiler enforces type checking on the parameters of the procedures.

10

11 The built-in function syntax is as follows:

12

13 <return-value> io\_in ( <device> [<args>] )

14 <return-value> io\_out ( <device>, <output-value> [<args>] )

15 where the <device> name corresponds to an I/O device declaration and <args> are as  
16 follows, depending on the type of device:

17 bitshift [ , <numbits> ]

18 serial (output only) , <count>

19 serial (input only) , <input-value>, <count>

20 neurowire (output only) , <count>

21 neurowire (input only) , <input-value>, <count>

22 parallel (output only) , <count>

23 parallel (input only) , <input-value>, <count>

1

2

All other devices do not permit extra arguments in the calls to io\_in or io\_out.

3

4

Some of the above arguments may also appear in the device declaration. If the attribute is specified in the declaration and the attribute is supplied as an argument, the argument overrides the declared value for that call only. These attributes may be specified in both places, either place or not at all. If left unspecified, the default is used (see defaults below).

5

6

7

8

9

10

11

The data type of the <return-value> for the function io\_in, and the data type of the <output-value> for io\_out is given by the following table. The data values will be implicitly converted as necessary. A warning is output by the compiler if an io\_in that returns a 16-bit quantity is assigned to a smaller value.

12

13

14

15

16

17

18

19

20

21

bit	short	bit 0 used; others are 0
bitshift	long	shifted value
frequency	long	period in nanoseconds
pulsewidth	short	pulsewidth in nanoseconds
pulsecount	long	count in .84 seconds
ontime, period	long	period in nanoseconds
quadrature	short	signed count
oneshot	short	count
nibble	short	bit 0-3 used; others are 0
byte	short	all bits used

1 For period, pulsecount and ontime input devices, the built-in variable "input\_is\_new"  
 2 is set to TRUE whenever the io\_in call returns an updated value. The frequency with which  
 3 updates occur depends on the device declaration

4 For parallel, serial and neurowire, io\_out and io\_in require a pointer to the data buffer  
 5 as the <output-value> and the <input-value>, respectively. For parallel and serial, io\_in  
 6 returns a short integer which contains the count of the actual number of bytes received.

7 Ranges and defaults

8 The following ranges and defaults apply to the various IO attributes:

9 • The bitshift "numbits" may be specified in the bitshift declaration as any  
 10 number from 1 to 16 and, if not specified, defaults to 16. In the calls to io\_in  
 11 and io\_out, the shift value may be any number from 1 to 127. For io\_in, only  
 12 the last 16 bits shifted in will be returned. For io\_out, after 16 bits, zeroes are  
 13 shifted out.

14 • The bitshift output clock may be either '+' or '-'. It defaults to '+'. This  
 15 defines whether the data is shifted on the positive-going or negative-going  
 16 edge of the clock. This can only be specified in the declaration.

17 • The initial frequencies of the frequency output, triac output, pulsewidth  
 18 output and pulsecount output are 0.

19 • The clock value specifies a clock in the range 0...7 where 0 is the fastest  
 20 clock and 7 is the slowest. The defaults are as follows:

21	frequency output	0
22	triac output	7
23	pulsewidth output	0

1 pulsecount output 7  
2 oneshot output 7  
3 ontime input 2  
4 period input 0

5 The baud rate of serial may be 300, 1200 or 2400 baud with a default of  
6 2400.

7 • The baud rate for neurowire and bitshift may be 1,10 or 25 kbits/second  
8 and defaults to 25 kbits/second.

9

10 Example

11 An example follows—to read a switch attached to pin 1 and light an LED attached to  
12 pin 2 when the switch is closed, the following would be coded by the application  
13 programmer:

14

```
15 IO_1 input bit ch1 switch;  
16 IO_2 output bit led;  
17 if (io_in(switch))  
18 {  
19     io_out (led, TRUE);  
20 }
```

21

22

23

1        I/O Multiplexing

2        The timer counter circuit may be multiplexed among pins 4 to 7. To facilitate this, the  
3 following built-in function is provided:

4        `io_select (<device>);`

5        This function causes the specified device to become the new owner of the timer  
6 counter circuit. Any reinitialization of the timer counter circuit is handled by this function. It  
7 is under the application's control when the timer counter is connected to which pin. The  
8 multiplexed timer counter is initially assigned to the mux device which is declared first.

9        For example, the application may choose to select a new device after a when change  
10 clause has executed for the current connected device. Alternatively, the selection could be  
11 done based on a timer, e.g., select a new device every 100 milliseconds.

12



## TABLE XIII

WHEN STATEMENT SYNTAX

1

2

3

4

5

6

7

8

10

11

12

13

14

15

16

17

18

19

20

21

22

23

The syntax of a when statement in the system of the preferred embodiment is given below:

[priority] when (event) task

where:

priority is an option used to force evaluation of the following when clause each time the scheduler runs. This allows priority when clauses to be evaluated first. Within a program having multiple priority when clauses, priority when clauses are evaluated in the order coded in the program. If any priority when clause evaluates to true, the corresponding task is run and the scheduler starts over at the top of the priority when clauses. If no priority when clause evaluates to true then a non-priority when clause is evaluated and selected in a round robin fashion. The scheduler then starts over with the priority when clauses. This process may be best understood by example: Assume the following when clauses coded in the following order:

priority when (A)  
priority when (B)  
when (C)  
when (D).

Assume only C and D are true; first A is evaluated, then B is

1 evaluated and finally C is evaluated and the task associated with C is  
2 executed. A is then evaluated again, then B is evaluated and then, in  
3 round robin fashion, D is evaluated and executed.

4 **event** may be either a predefined event or, importantly, may be any valid C  
5 expression. Predefined events include, by way of example, input  
6 pin state changes (io changes, io update occurs); network variable  
7 changes (network variable update completes, network variable  
8 update fails, network variable update occurs, network variable  
9 update succeeds); timer expiration; message reception information  
10 (message arrives, message completes, message fails, message  
11 succeeds); and other status information (powerup, reset, online,  
12 offline).

13 **task** is a C compound statement consisting of a series of C declarations  
14 and statements enclosed in braces.

15

16 The following predefined events exist in the system of the preferred embodiment:

17 **flush\_completes** A flush function is available in the system of the preferred  
18 embodiment which causes the node to monitor the status of all  
19 incoming and outgoing messages. When the node has  
20 completed processing of all messages the flush\_complete event  
21 becomes true indicating all outgoing transactions have been  
22 completed, no more incoming messages are outstanding, and no  
23 network variable updates are occurring.

1	<b>io_changes</b>	This event indicates the status of one or more I/O pins associated
2		with a specified input device have changed state.
3	<b>io_update_occurs</b>	This event indicates that a timer/counter device associated with a
4		specified pin has been updated.
5	<b>msg_arrives</b>	This event indicates a message has arrived for processing.
6	<b>msg_completes</b>	This event indicates a message has completed (either
7		successfully or by failure).
8	<b>msg_fails</b>	This event indicates a message has failed.
9	<b>msg_succeeds</b>	This event indicates a message has completed successfully.
10	<b>nv_update_completes</b>	This event indicates a network variable update has completed
11		(either successfully or by failure).
12	<b>nv_update_fails</b>	This event indicates a network variable update has failed
13	<b>nv_update_occurs</b>	This event indicates a network variable update has occurred.
14	<b>nv_update_succeeds</b>	This event indicates a network variable update has completed
15		successfully.
16	<b>offline</b>	This event indicates the node has been taken offline.
17	<b>online</b>	This event indicates the node has been brought online.
18	<b>powerup</b>	This event indicates the node has been powered up.
19	<b>reset</b>	This event indicates the node has been reset.
20	<b>resp_arrives</b>	This event indicates a response has arrived to a message.
21	<b>timer_expires</b>	This event indicates the designated timer has expired.
22		

1           Predefined events may be used within other control expressions in addition to in the  
2 when statement; for example, in an if, while or for statement.

3           As noted above, a user-defined event may be any valid C expression and may  
4 include assignment statements to assign values to global variables and function calls.

5

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42

TABLE XIV

ASSEMBLY LANGUAGE LISTING FOR THE  
NETWORK VARIABLE PROGRAM OF TABLE V

```

; APC - Echelon (R) Neuron (R) Application-C Compiler
; Copyright (c) 1990, Echelon Corporation
; Assembly code from APC Revision 0.99
; Code generated from 'node _ 31 _ right.nc' on Mon Feb 4 10:31:40 1991

SUBHEAD Generated from the input file: node_ 31_ right.nc

RADIX HEX

SEG EECODE.
ORG 0F000
NBMTS EQU 000
NNVS EQU 00E
PROTECT EQU 00

NEURONID
RES 8
DIRECTORY
data.b PTR TEVT-3*NNVS,PROTECT+NNVS,0,0
PROGID
data.b 048,056,041,043,06F,06D,070,000
MODETABLE
data.b 053
data.b 0F0,037
data.b 024,09B
data.b 033,033,033
EENEARBYTES
data.b 000
data.b 000,000
data.b 001,000,000,047
LOCATION
RES 8
COMM
RES 10
MSG
RES 1
DOMAIN
RES 01E
    
```

```

1  ADDR
2  RES    04B
3  TNVCNFG
4  RES    3 *NNVS
5  CONFIGCHECKSUM
6  RES    1
7  PAGE
8  ; Boilerplate file for compiler-generated assembly output
9  ; Copyright (c) 1990, 1991 Echelon Corporation. All Rights Reserved.
10 ; Date last modified: 1/30/91
11 ; List of exported symbols
12     EXPORT APINIT,DOMAIN,EENEARBYTES,MODETABLE,TEVT
13 ; List of imported symbols - Library functions
14     IMPORT    application _restart,delay,error log
15     IMPORT    flush,flush _cancel,flush _wait,go _offline,go _unconfigured
16     IMPORT    node _reset,offline _confirm, _post events,powerup
17     IMPORT    random,retrieve _status,retrieve _xcvr _status
18     IMPORT    reverse,timers _off,watchdog _update
19 ; List of imported symbols - Compiler helper functions
20     IMPORT    _abs8,_abs16,_add16,_and16,_alloc
21     IMPORT    _bcd2bin,_bin2bcd,_dealloc
22     IMPORT    _div8,_div8s,_div16,_div16s
23     IMPORT    _drop_n,_drop_n _preserve_1,_drop_n _preserve_2
24     IMPORT    _drop_n _return_1,_drop_n _return_2
25     IMPORT    _eeprom_write,_eeprom _write_long
26     IMPORT    _eeprom_far_write,_eeprom_far _write_long
27     IMPORT    _equal 8,_equal 116
28     IMPORT    _gequ8,_gequ8s,_gequ16,_gequ16s
29     IMPORT    _less8,_less8s,_less16,_less16s
30     IMPORT    _log8,_log16,_lognot8,_lognot16
31     IMPORT    _l_shift8,_l_shift8s,_l_shift16,_l_shift16s
32     IMPORT    _l_shift8_3,_l_shift8_4,_l_shift8_5,_l_shift8_6,_l_shift8_7
33     IMPORT    _max8,_max 8s,_max16,_max16s
34     IMPORT    _memcpy,_memcpy_l,_memset,_memset_l
35     IMPORT    _min8,_min8s,_min16,_min16s
36     IMPORT    _minus_16_s
37     IMPORT    _mod8,_mod8s,_mod16,_mod16s,_mul8,_mul16
38     IMPORT    _not16,_or16
39     IMPORT    _pop,_push
40     IMPORT    _r_shift8,_r_shift8s,_r_shift16,_r_shift16s
41     IMPORT    _r_shift8_3,_r_shift8_4,_r_shift8_5,_r_shift8_6,_r_shift8_7
42     IMPORT    _register_call,_sign_extend16
43     IMPORT    _sub16,_xor16
44     PAGE
45 ; List of imported symbols - I/O functions
46     IMPORT    _bit_input,_bit_input_d,_bit_output

```

```

1  IMPORT  _bitshift_input,_bitshift_output
2  IMPORT  _byte_input,_byte_output
3  IMPORT  _nibble_input,_nibble_input_d,_nibble_output
4  IMPORT  _frequency_output,_leveldetect_input
5  IMPORT  _neurowire_input,_neurowire_output
6  IMPORT  _oneshot_output,_ontime_input
7  IMPORT  _parallel_input,_parallel_input_ready
8  IMPORT  _parallel_output,_parallel_output_ready,_parallel_output_request
9  IMPORT  _period_input
10 IMPORT  _pulsecount_input,_pulsecount_output
11 IMPORT  _pulsewidth_output
12 IMPORT  _quadrature_input
13 IMPORT  _serial_input,_serial_output
14 IMPORT  _totalize_input,_triac_output,_triggered_count_output
15 IMPORT  _init_timer_counter1,_init_timer_counter2
16 IMPORT  _init_baud,_io_set_clock
17 IMPORT  _io_input_value,_io_change_init,_select_input_fn
18 ; List of imported symbols - Messaging support
19 IMPORT  _bound_mt
20 IMPORT  _msg_alloc,_msg_alloc_priority,_msg_cancel,_msg_free
21 IMPORT  _msg_receive,_msg_send
22 IMPORT  _msg_addr_blockget,_msg_addr_get,_msg_auth_get,_msg_code_get
23 IMPORT  _msg_data_blockget,_msg_data_get,_msg_len_get,_msg_service_get
24 IMPORT  _msg_addr_blockset,_msg_addr_set,_msg_auth_set,_msg_code_set
25 IMPORT  _msg_data_blockset,_msg_data_set,_msg_domain_set,_msg_node_set
26 IMPORT  _msg_priority_set,_msg_service_set,_msg_tag_set
27 IMPORT  _resp_alloc,_resp_cancel,_resp_free,_resp_receive,_resp_send
28 IMPORT  _resp_code_set,_resp_data_blockset,_resp_data_set
29
30 ; List of imported symbols - Network Variable support
31 IMPORT  _bound_nv,_nv_poll,_nv_poll_all
32 IMPORT  _nv_update,_nv_update_int,_nv_update_long
33 IMPORT  _nv_update_int_offset,_nv_update_long_offset
34 ; List of imported symbols - Timer support
35 IMPORT  _timer_get,timer_off
36 IMPORT  _timer_mset,_timer_mset_repeat,_timer_sset,_timer_sset_repeat
37 ; List of imported symbols - Event support
38 IMPORT  _flush_completes
39 IMPORT  _io_changes,_io_changes_to,_io_changes_by,_io_update_occurs
40 IMPORT  _msg_arrives,_msg_code_arrives
41 IMPORT  _msg_completes,_msg_fails,_msg_succeeds
42 IMPORT  _nv_update_completes,_nv_update_fails,_nv_update_succeeds
43 IMPORT  _nv_update_occurs
44 IMPORT  _offline,_online,_resp_arrives
45 IMPORT  _timer_expires,_timer_expires_any,_wink
46 ; List of imported symbols - Misc builtin function support

```

```

1      IMPORT      _sleep
2      ; End boilerplate file
3
4      PAGE
5
6          SEG      ENEAR
7          ORG      CONSTRAINED
8      %MinOffTime
9          RES      02
10         EXPORT   %MinOffTime
11
12         SEG      EENEAR
13         ORG      CONSTRAINED
14     %MinOnTime
15         RES      02
16         EXPORT   %MinOnTime
17
18         SEG      EENEAR
19         ORG      CONSTRAINED
20     %OutletWater
21         RES      02
22         EXPORT   %OutletWater
23
24         SEG      EENEAR
25         ORG      CONSTRAINED
26     %CndnsrHead
27         RES      02
28         EXPORT   %CndnsrHead
29
30         SEG      EENEAR
31         ORG      CONSTRAINED
32     %CoolAir
33         RES      02
34         EXPORT   %CoolAir
35
36         SEG      EENEAR
37         ORG      CONSTRAINED
38     %CmprsrInltGas
39         RES      02
40         EXPORT   %CmprsrInltGas
41
42         SEG      RAMNEAR
43         ORG      CONSTRAINED
44     %OutletWaterTemp
45         RES      01
46         EXPORT   %OutletWaterTemp
    
```



1			
2	SEG	RAMNEAR	
3	ORG	CONSTRAINED	
4			
5	%CndnsrHeadTemp		
6	RES	01	
7	EXPORT	%CndnsrHeadTemp	
8			
9	SEG	RAMNEAR	
10	ORG	CONSTRAINED	
11	%CoolAirTemp		
12	RES	01	
13	EXPORT	%CoolAirTemp	
14			
15	SEG	RAMNEAR	
16	ORG	CONSTRAINED	
17	%CmprssrGasPrssr		
18	RES	01	
19	EXPORT	%CmprssrGasPrssr	
20			
21	SEG	RAMNEAR	
22	ORG	CONSTRAINED	
23			
24	RES	01	
25	EXPORT	%BuildingCooling	
26			
27	SEG	RAMNEAR	
28	ORG	CONSTRAINED	
29	%MotorOn		
30	RES	01	
31	EXPORT	%MotorOn	
32			
33	SEG	RAMNEAR	
34	ORG	CONSTRAINED	
35	%MotorOverload		
36	RES	01	
37	EXPORT	%MotorOverload	
38			
39	SEG	RAMNEAR	
40	ORG	CONSTRAINED	
41	%AmOnline		
42	RES	01	
43	EXPORT	%AmOnline	
44			
45	SEC	RAMNEAR	
46	ORG		

```

1  % strikes
2  RES      01
3  EXPORT  %strikes
4
5  SEG      CODE
6  ORG
7  EXPORT  %motor
8
9
10 %motor ; Function body
11 push    tos
12 push    #0B
13 call    _nv_update_int
14 push    tos
15 pushs   #00
16 call    _bit_output
17 pushs   01
18 push    next
19 call    _equal8
20 sbrnz   *+4
21 brf     %motor+01D
22 push    %MinOnTime
23 push    %MinOnTime+01
24 pushs   #01
25 call    _timer_sset
26 brf     %motor+026
27 push    %MinOffTime
28 push    %MinOffTime+01
29 pushs   #00
30 call    _timer_sset
31 dealloc #01
32
33 SEG      CODE
34 ORG
35 EXPORT  %control_action
36 %control_action ; Function body
37 push    [ 1 ] [ @NEAR ( %CoolAirTemp ) ]
38 push    %CoolAir+1
39 call    _less8s
40 push    %CmprsrInltGas
41 push    [ ] [ @NEAR ( CmprsrGasPsr ) ]
42 call    _less8s
43 push    %CndnsrHead
44 push    [ 1 ] [ @ NEAR ( %CndnsrHeadTemp ) ]
45 push    [ 1 ] [ @NEAR ( %OutletWaterTemp ) ]
46 push    %OutletWater+01

```

```

1      call      _less8s
2      pushs    #00
3      pushs    #00
4      pushs    #00
5      call      _timer_get
6      call      _equal16
7      push     [1][@NEAR(%BuildingCooling)]
8      push     [1][@NEAR(%AmOnline)]
9      and
10     and
11     and
12     and
13     and
14     and
15     sbrnz    *+4
16     brf      %control_action+038
17     pushs    #01
18     callf    %motor
19     brf      %control_action+06A
20     push     %CoolAir
21     push     [1][@NEAR(%CoolAirTemp)]
22     call      _less8s
23     push     [1][@NEAR(%CmprssrGasPrsr)]
24     push     %CmprssrInltGas+01
25     call      _less8
26     push     [1][@NEAR(%CndnsrHeadTemp)]
27     push     %CndnsrHead+01
28     call      _less8s
29     push     %OutletWater
30     push     [1][@NEAR(%OutletWaterTemp)]
31     call      _less8s
32     pushs    #00
33     pushs    #00
34     pushs    #01
35     call      _timer_get
36     call      _equal16
37     push     [1][@NEAR(%BuildingCooling)]
38     and
39     and
40     and
41     and
42     and
43     sbrnz    *+4
44     brf      %control_action+06A
45     pushs    #00
46     callf    %motor

```

```

1      ret
2
3      SEG      CODE
4      ORG
5      WHEN1
6      EXPORT  WHEN1
7      callf   %control_action
8      ret
9
10     SEG      CODE
11     ORG
12     WHEN2
13     EXPORT  WHEN2
14     push    #0A
15     call    _sign_extend16
16     pushs   #02
17     call    _timer_sset
18     push    #040
19     call    _pulsecount_input
20     push    #0B4
21     call    _sign_extend16
22     call    _less16
23     sbrnz   *+4
24     brf     WHEN2+02E
25     pushs   #03
26     push    [1][@NEAR(%strikes)]
27     inc
28     push    tog
29     pop     [1][@NEAR(%strikes)]
30     call    _gequ8s
31     sbrnz   *+4
32     brf     WHEN2+02B
33     pushs   #00
34     callf   %motor
35     pushs   #01
36     push    #0C
37     call    _nv_update_int
38     brf     WHEN2+031
39     pushs   #00
40     pop     [1][@NEAR(%strikes)]
41     ret
42
43     SEG      CODE
44     ORG
45     APINIT : Init & event code
46     push    #084

```

```

1      push      #072
2      call      _init_timer_counter1
3      ret
4      EXPORT    RESET
5      RESET ; When-unit body
6      pushs     #00
7      push      #0B
8      call      _nv_update_int
9      pushs     #00
10     push      #0C
11     call      _nv_update_int
12     pushs     #01
13     push      #0D
14     call      _nv_update_int
15     pushs     #00
16     callf     %motor
17     push      %OutletWater
18     pop       [1][@NEAR(%OutletWaterTemp)]
19     push      %CndnsrHead
20     pop       [1][@NEAR(%CndnsrHeadTemp) ]
21     push      %CoolAir
22     pop       [1][@NEAR(%CoolAirTemp) ]
23     push      %CmprsrInltGas
24     pop       [1][@NEAR ( %CmprsrGasPrsr) ]
25     pushs     #00
26     pop       [1][@NEAR(%strikes)]
27     push      #0A
28     call      _nv_poll
29     ret
30     EXPORT    OFFLINE
31     OFFLINE; When-unit body
32     pushs     #00
33     push      #0D
34     call      _nv_update_int
35     pushs     #00
36     callf     %motor
37     ret
38     EXPORT    ONLINE
39
40     ONLINE ; When-unit body
41     pushs     #01
42     push      #0D
43     call      _nv_update_int
44     pushs     #00
45     callf     %motor
46     pushs     #00

```

```

1      push      #0C
2      call      _nv_update_int
3      ret
4
5      SEG      CODE
6      ORG
7      TNVFIX ; NV Fixed table
8      data.b    022,PTR %MinOffTime
9      data.b    022,PTR %MinOnTime
10     data.b    022,PTR %OutletWater
11     data.b    022,PTR %CndnsrHead
12     data.b    022,PTR %CoolAir
13     data.b    022,PTR %CmprssrInltGas
14     data.b    021,PTR %OutletWaterTemp
15     data.b    021,PTR %CndnrHeadTemp
16     data.b    021,PTR %CoolAirTemp
17     data.b    021,PTR %CmprssrGasPrssr
18     data.b    021,PTR %BuildingCooling
19     data.b    021,PTR %MotorOn
20     data.b    021,PTR %MotorOverload
21     data.b    021,PTR %AmOnline
22
23     TEVT ; Event table
24     data.b    PTR APINIT-2
25     data.b    0,RESET-APINIT+1
26     data.b    OFFLINE-APINIT+1,ONLINE-APINIT+1
27     data.b    00,02
28     data.b    OFF,PTR WHEN1-1
29     data.b    0A,PTR WHEN2-1
30
31     ; Resource usage information
32     RESOURCE  NADDR      0F
33     RESOURCE  NDOM       2
34     RESOURCE  NRCVIX    08
35     RESOURCE  NTMR      03
36     RESOURCE  NNIB      02
37     RESOURCE  NNOB      02
38     RESOURCE  NAIB      02
39     RESOURCE  NAOB      02
40     RESOURCE  NNPOB     02
41     RESOURCE  NAPOB     02
42     RESOURCE  SNIB      042
43     RESOURCE  SNOB      02A
44     RESOURCE  SAIB      016
45     RESOURCE  SAOB      014
46     RESOURCE  NNVS      0E
    
```

## TABLE XV

GENERAL DEFINITIONS

1  
2  
3  
4 The following definitions are generally applicable to terms used in this specification:

5 Neuron, or node: A neuron or node is an intelligent, programmable element or  
6 elements providing remote control, sensing, and/or communications, that when  
7 interconnected with other like elements forms a communications, control and/or sensing  
8 network. Nodes are named with Neuron ids (see below). Nodes may be addressed as a  
9 part of a domain and subnet using a node number. The node number in the preferred  
10 embodiment is 7 bits. Multiple nodes may be addressed using a group id. The group id in  
11 the preferred embodiment is 8 bits.

12 Neuron id: Nodes in the present invention are assigned a unique identification  
13 number at the time of manufacture. The identification number is preferably 48-bits long.  
14 This 48-bit identification number does not change during the lifetime of node. As is  
15 appreciated, the assignment of a unique identification to each individual node allows for  
16 numerous advantages. This 48-bit identification number may be referred to as the node\_id.

17 Domain addresses: The term "domain" is used to describe a virtual network  
18 wherein all communication, as supported by the network of the present invention, must be  
19 within a single domain. Any required inter-domain communication must be facilitated by  
20 application level gateways. In the preferred embodiment, domains are identified with  
21 48-bit domain identifiers. However, in certain applications the size of the domain field may  
22 vary.

1           Subnet — In the preferred embodiment, a subnet is a subset of a domain  
2 containing from 0 to 127 nodes. In the preferred embodiment, subnets are identified with  
3 an 8-bit subnet identification number. A single domain may contain up to 255 subnets.

4           Group: A group is a set of nodes which work together to perform a common  
5 function. In the preferred embodiment, groups are identified with an 8-bit group  
6 identification number. A single domain may contain up to 255 groups. For example, a  
7 group may be created to include all nodes in a connection, such as connection\_2 142 in  
8 which case the group would include a node at temperature sensor\_2 116, a node at cell\_1  
9 101 and a node at cell\_2 121.

10           Addressing — The present invention provides for a hierarchical address structure  
11 and supports three basic addressing modes: (1) (Domain, Subnet, Node number); (2)  
12 (Domain, Subnet, Node\_id); and (3) (Domain, Group).  
13



CLAIMS

What is claimed is:

1. A method for declaring and configuring variables in a program for execution on a processor, said method comprising the steps of:
  - (a) declaring a state of a parameter for a variable, said parameter indicating a characteristic of said variable;
  - (b) declaring said parameter as configurable by a configuration process;
  - (b) compiling said program, said compiling producing a compiled program as an output; said compiling further producing an output file providing information on said parameter, said information including information on said parameter;
  - (d) configuring said program using a configuration routine said configuration process including varying the state of said parameter.
2. The method as recited by claim 1 wherein said state of said parameter is selected from a set of states consisting of unacknowledged, unacknowledged with repeat and acknowledged.
3. The method as recited by claim 1 wherein said state of said parameter is selected from a set of states consisting of authenticated and unauthenticated.
4. The method as recited by claim 1 wherein said state of said parameter is selected from a set of states consisting of priority and nonpriority.
5. A method of scheduling tasks in a computer system comprising the steps of:
  - (a) declaring a first event as a first logical programming expression;
  - (b) declaring at least one first step to be accomplished by a first task;
  - (c) evaluating said first logical programming expression;

- (d) determining said first logical programming expression evaluates to true; and
  - (e) executing said at least one first step responsive to determining said first logical programming expression evaluates to true.
6. The method as recited by claim 5 wherein said method further includes the steps of:
- (f) declaring a second event as a second logical programming expression, said second event declared as a priority event;
  - (g) declaring at least one second step to be accomplished by a second task;
  - (h) evaluating said second logical programming expression prior to evaluating said first logical programming expression;
  - (i) determining said second logical programming expression evaluates to true; and
  - (j) executing said at least one second step responsive to determining said second logical programming statement evaluates to true.
7. The method as recited by claim 5 wherein said declaration of said first event comprises the keyword "when" followed by said first logical programming expression.
8. A method of prioritizing and scheduling tasks in a computer system comprising the steps of:
- (a) declaring a first event as a priority event;
  - (b) declaring at least one first step to be accomplished by a first task;
  - (c) declaring a second event as a non-priority event;
  - (d) declaring at least one second step to be accomplished as a second task;
  - (e) evaluating said first event to determine if said first event has occurred;

- (f) evaluating said second event to determine if said second event has occurred, said evaluation of said second event being accomplished subsequent to said evaluation of said first event;
  - (g) determining said first event has occurred; and
  - (h) executing said at least one first step responsive to determining said first event has occurred.
9. The method as recited by claim 8 wherein said first event is declared as a logical programming expression.
10. In a method of scheduling tasks to be executed by a computer, said method including evaluating each of a plurality of events associated with each of said tasks in a round robin fashion and conditionally executing said associated ones of said tasks upon evaluation of one of said events as true, the improvement wherein:
- (a) certain of said events are designated as priority events and others of said events are evaluated as non-priority events; and
  - (b) said priority events are evaluated prior to evaluation of said non-priority events.
11. The method as recited by claim 9 wherein said at least one of said plurality of events is declared as a logical programming expression.
12. In a device for scheduling execution of tasks responsive to occurrence of events, said device comprising a processor for executing said tasks, wherein said events may comprise any of a plurality of predefined event types, an improvement wherein said events may further comprise a logical programming expression.

13. The improvement of claim 12 wherein said logical programming expression may comprise any valid C programming language logical statement.
  
14. A method for declaring and accessing I/O devices comprises the steps of:
  - (1) declaring an I/O device designating a hardware device, a device type and a device name;
  - (2) configuring said hardware device to perform functions of said device type responsive to declaring said I/O device;
  - (3) executing an operation, said operation designating said device name; and
  - (4) performing an I/O operation comprising functions of said device type on said hardware device responsive to execution of said operation.
  
15. The method of claim 14 wherein said device type is chosen from the set consisting of bit, bitshift, frequency, triac, pulsewidth, pulsecount, ontime, serial, parallel, microwire, quadrature, period, oneshot, nibble and byte.

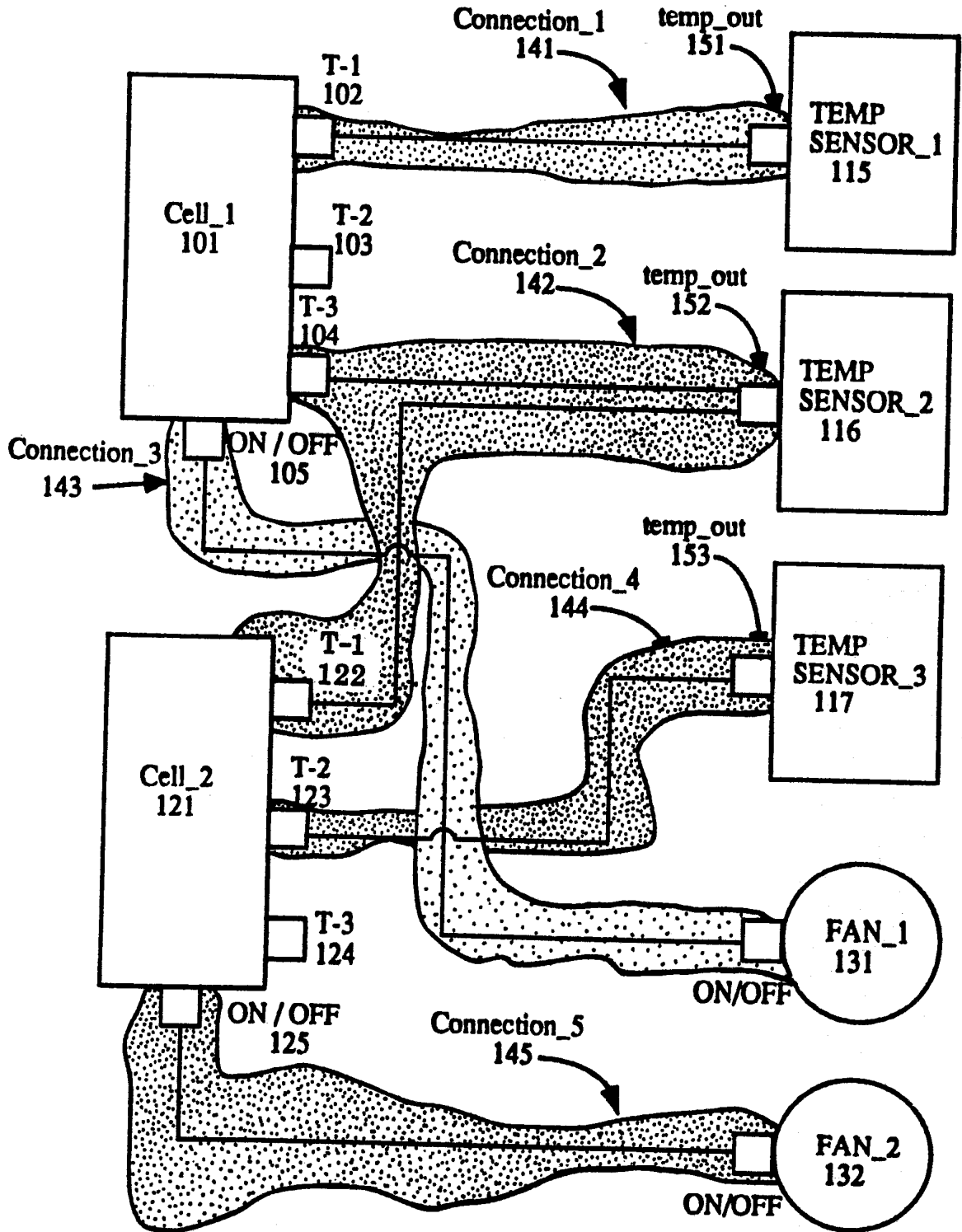


FIG. 1

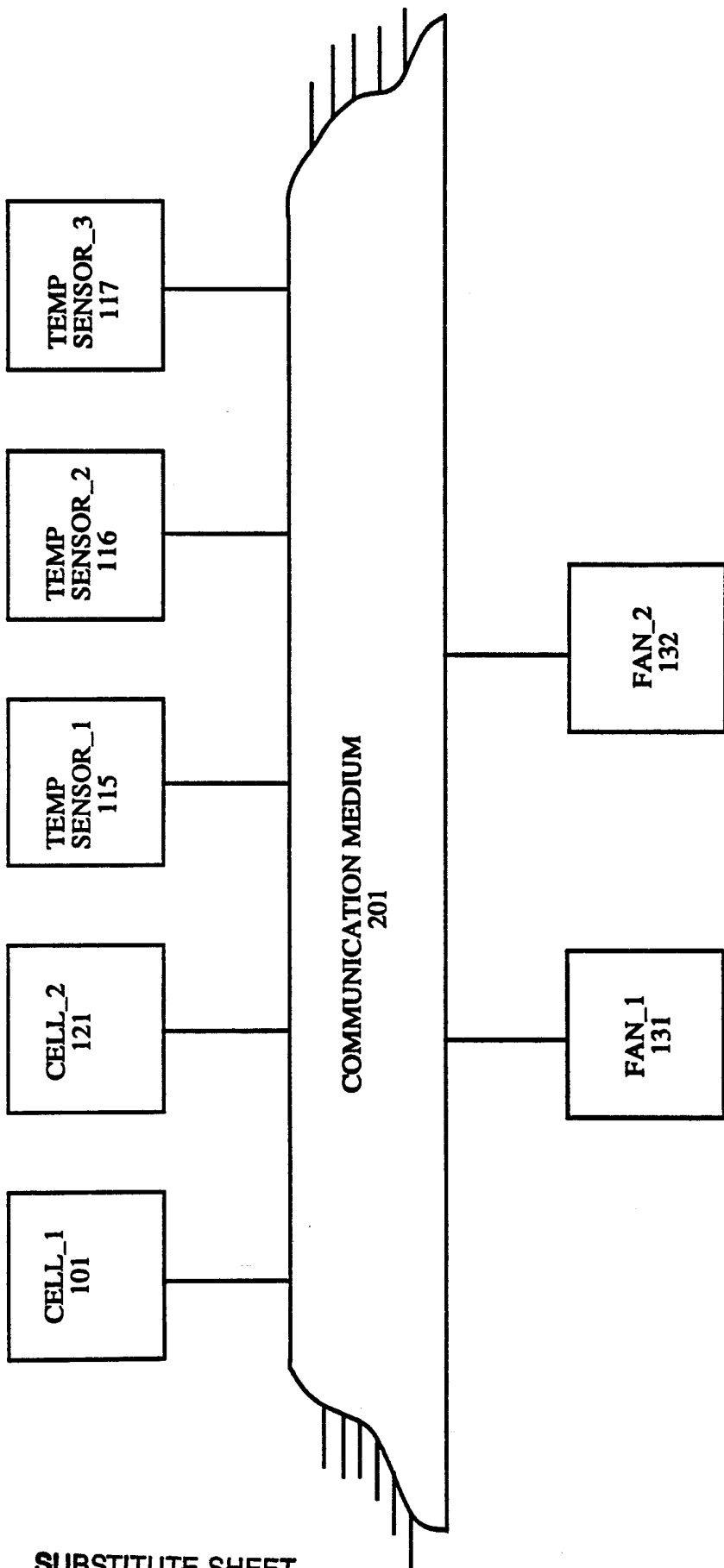


FIG. 2

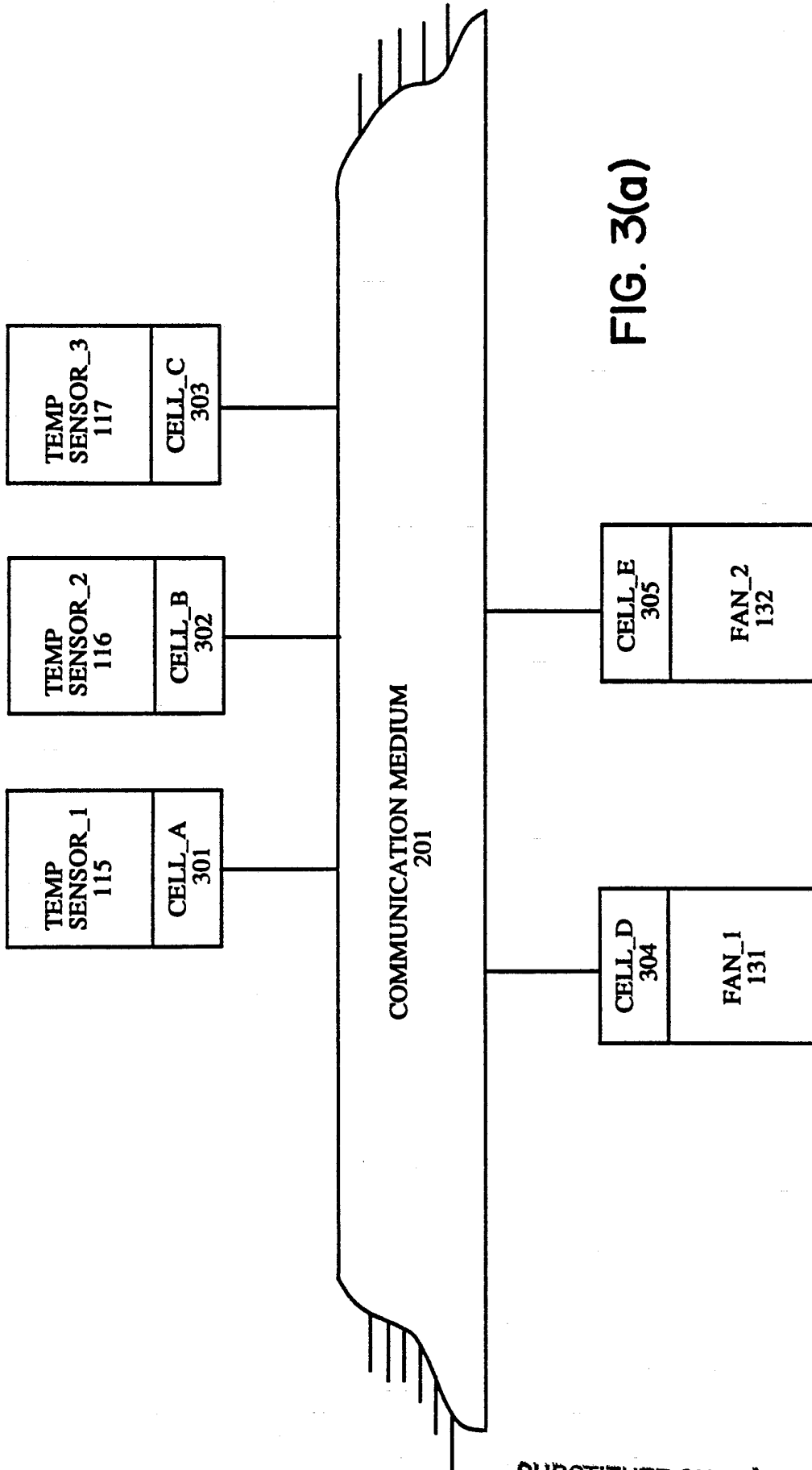


FIG. 3(a)

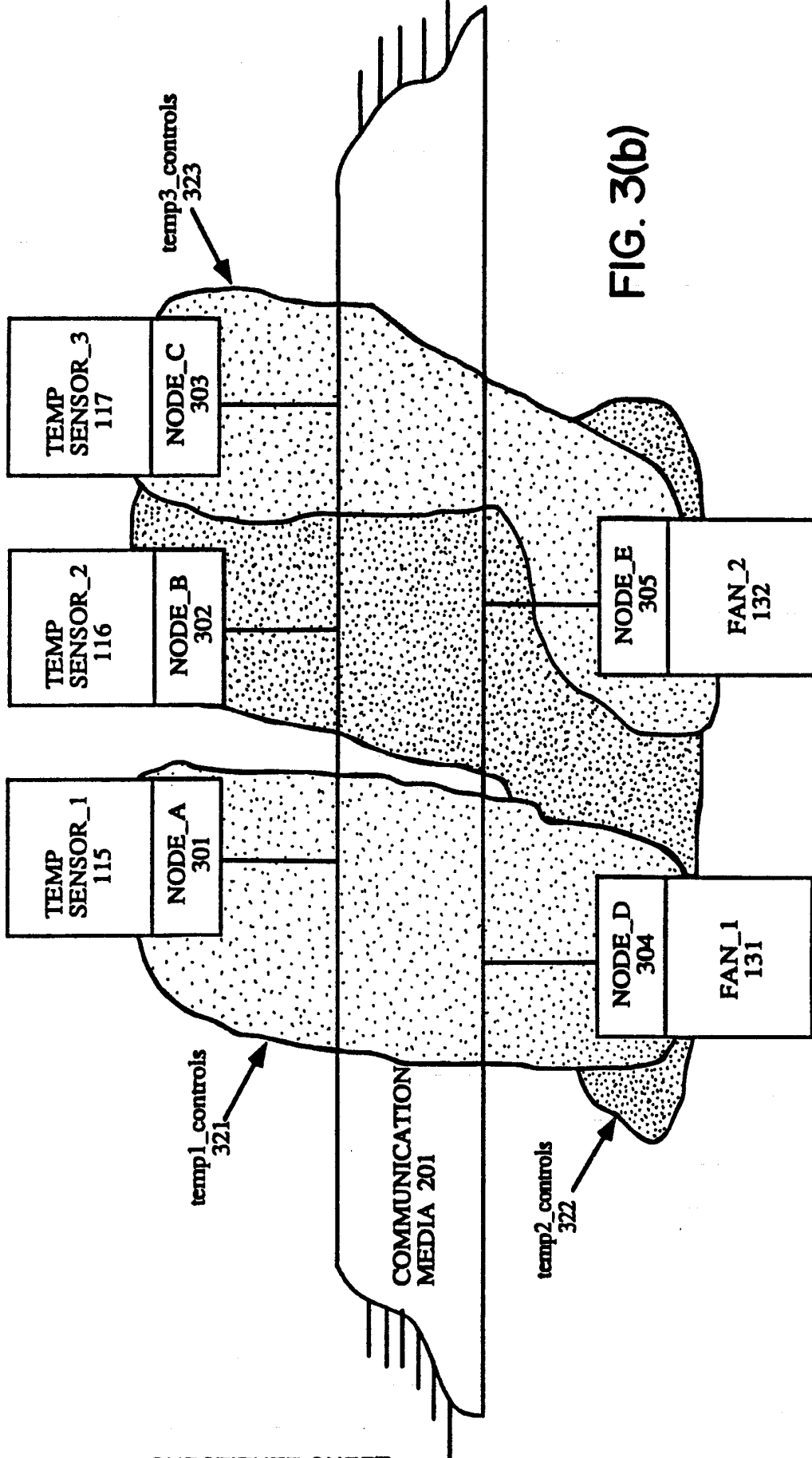


FIG. 3(b)



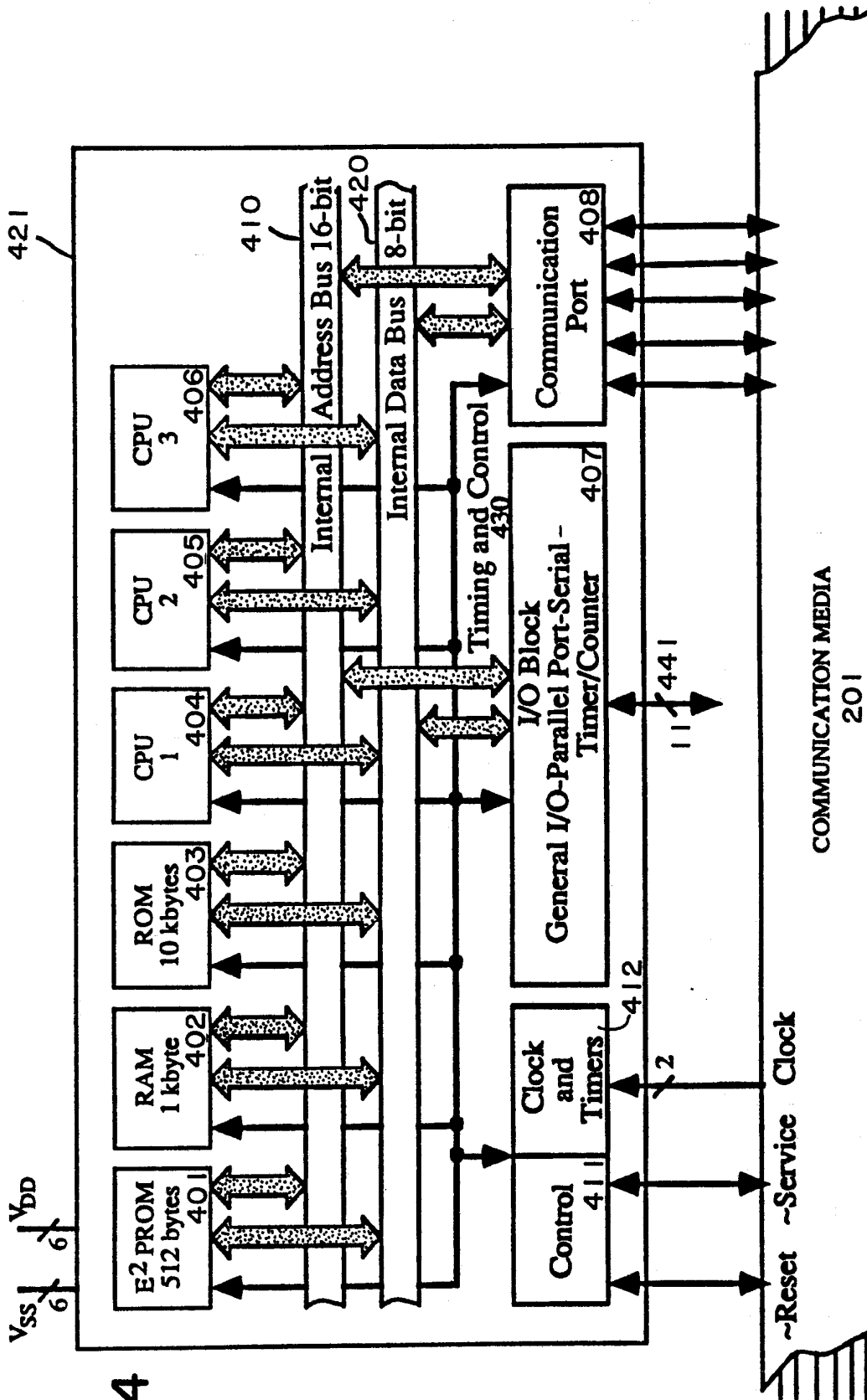


FIG. 4

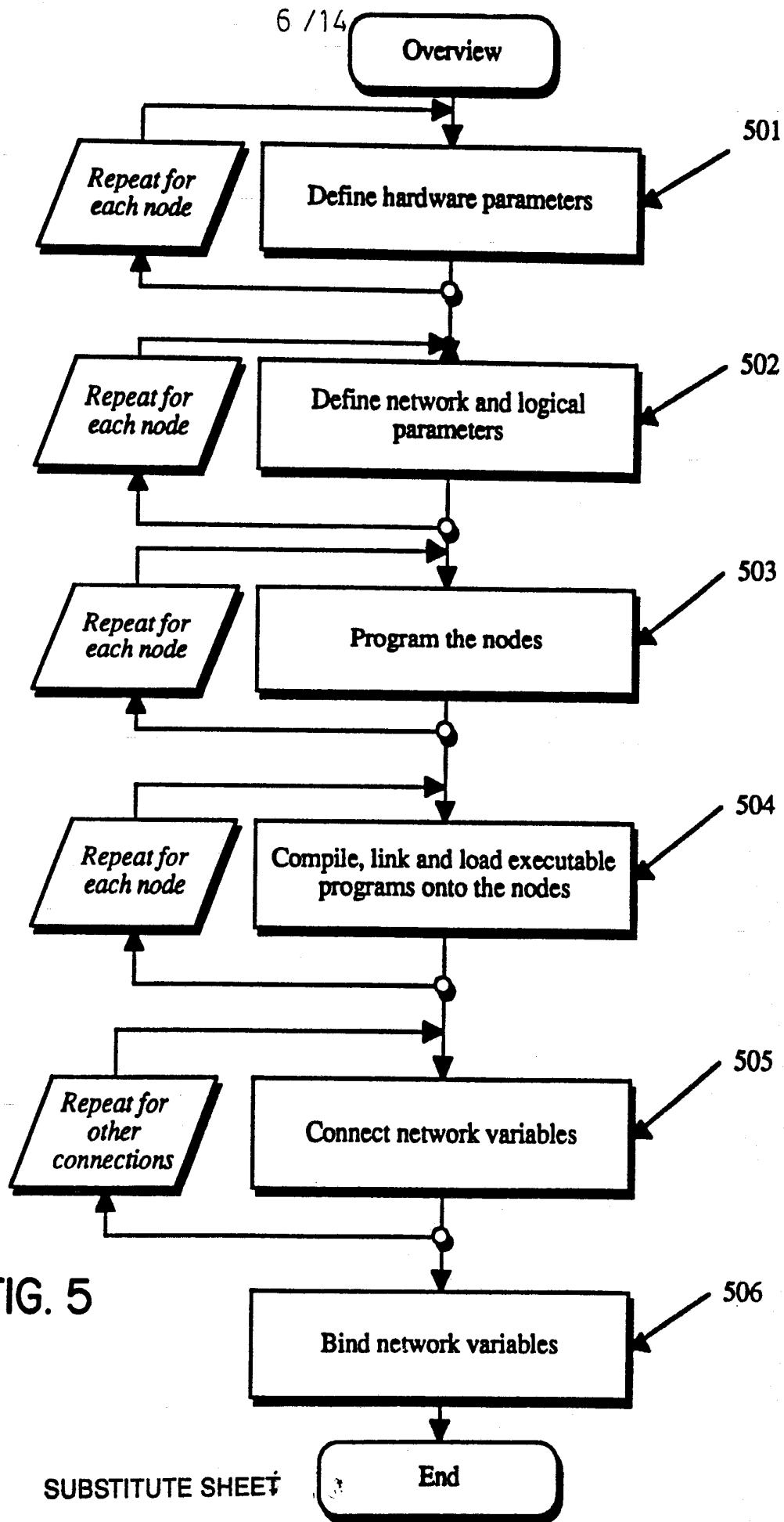


FIG. 5

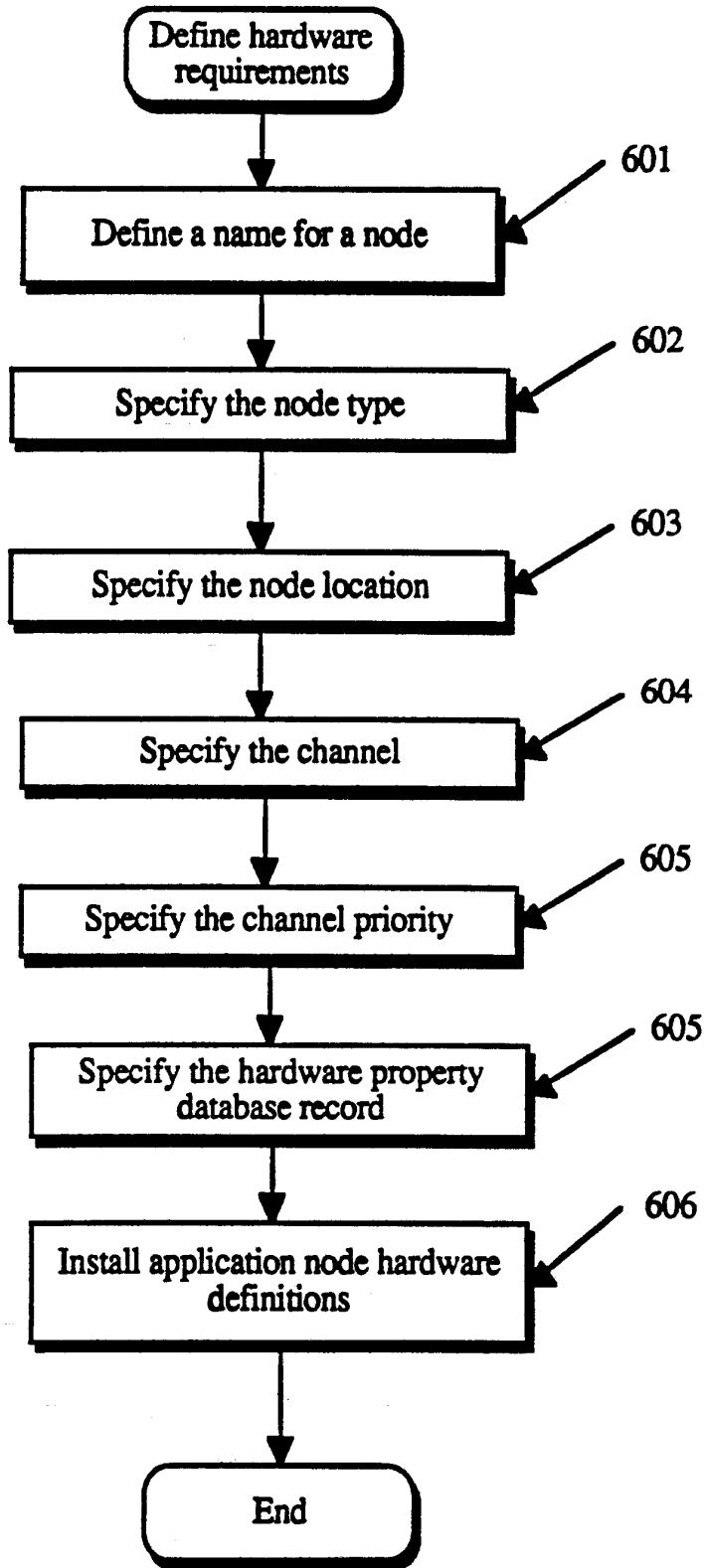


FIG. 6

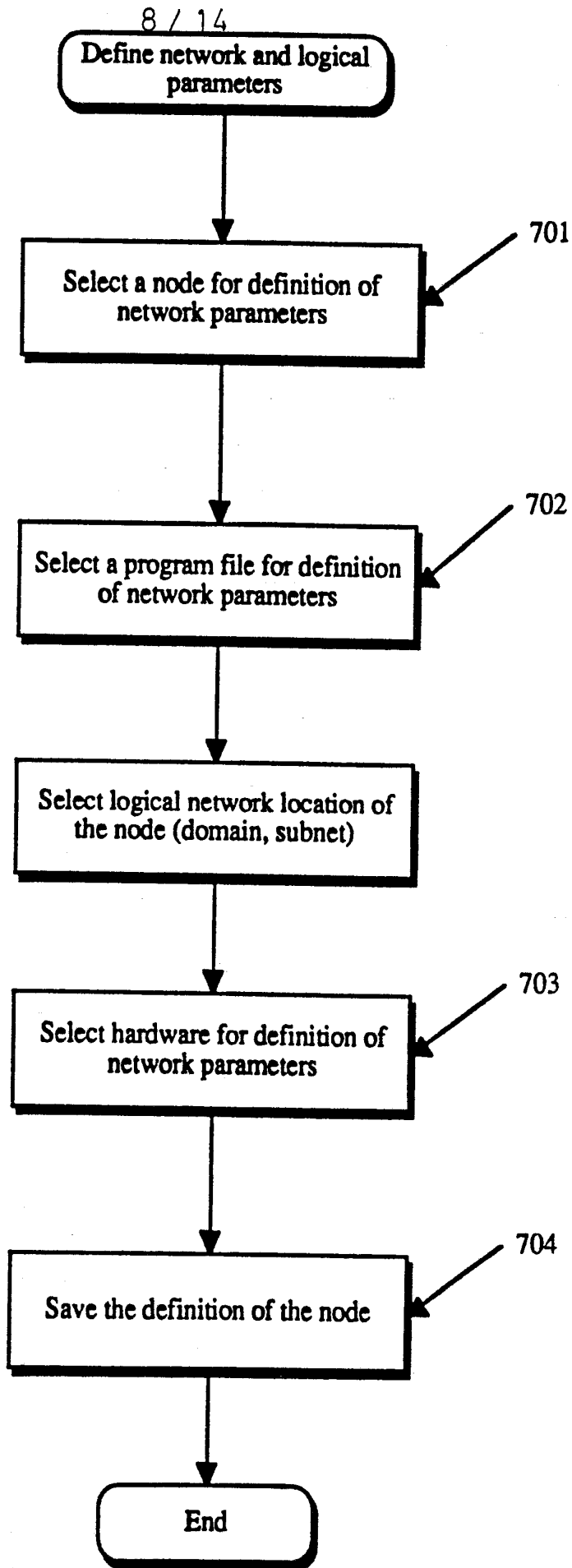


FIG. 7

9 / 14

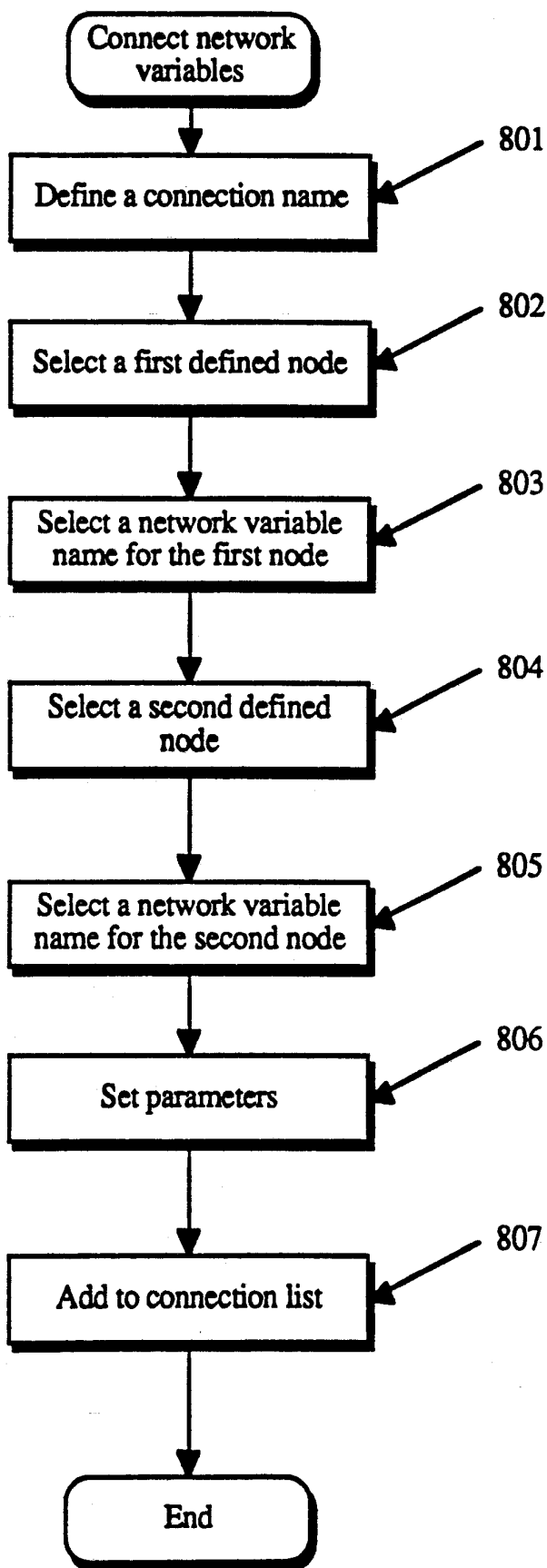


FIG. 8(a)

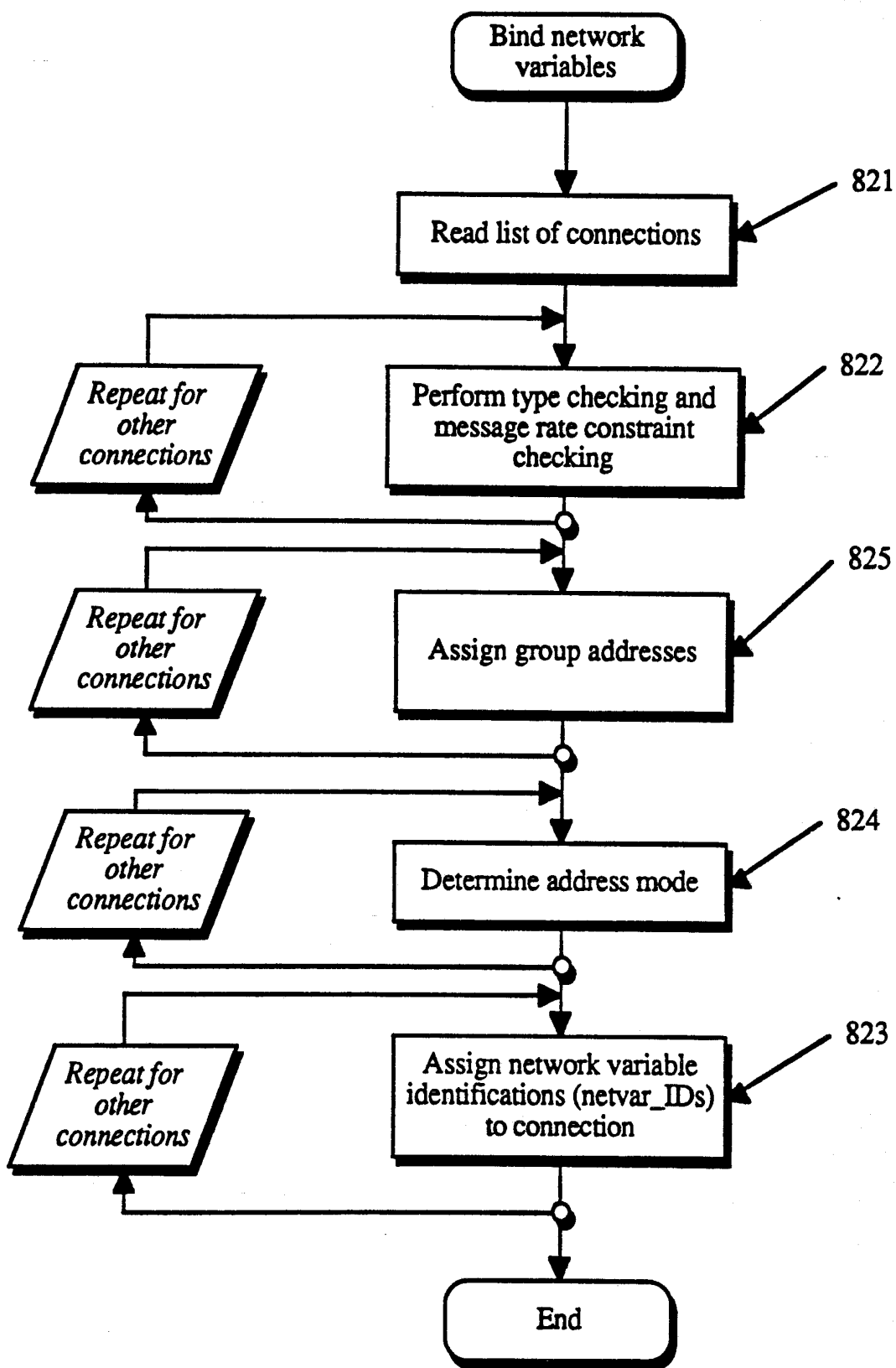


FIG. 8(b)

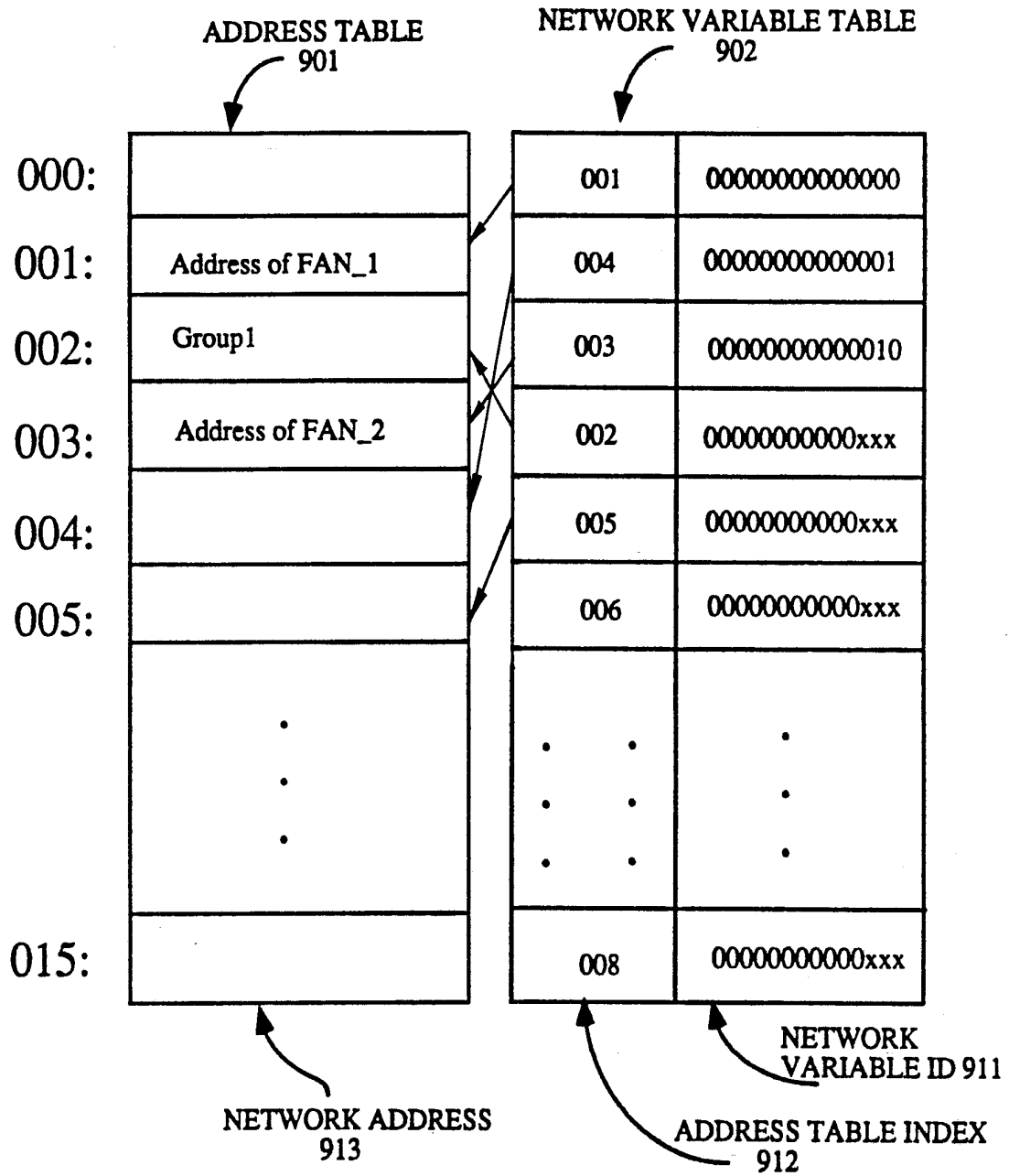


FIG. 9

12 / 14

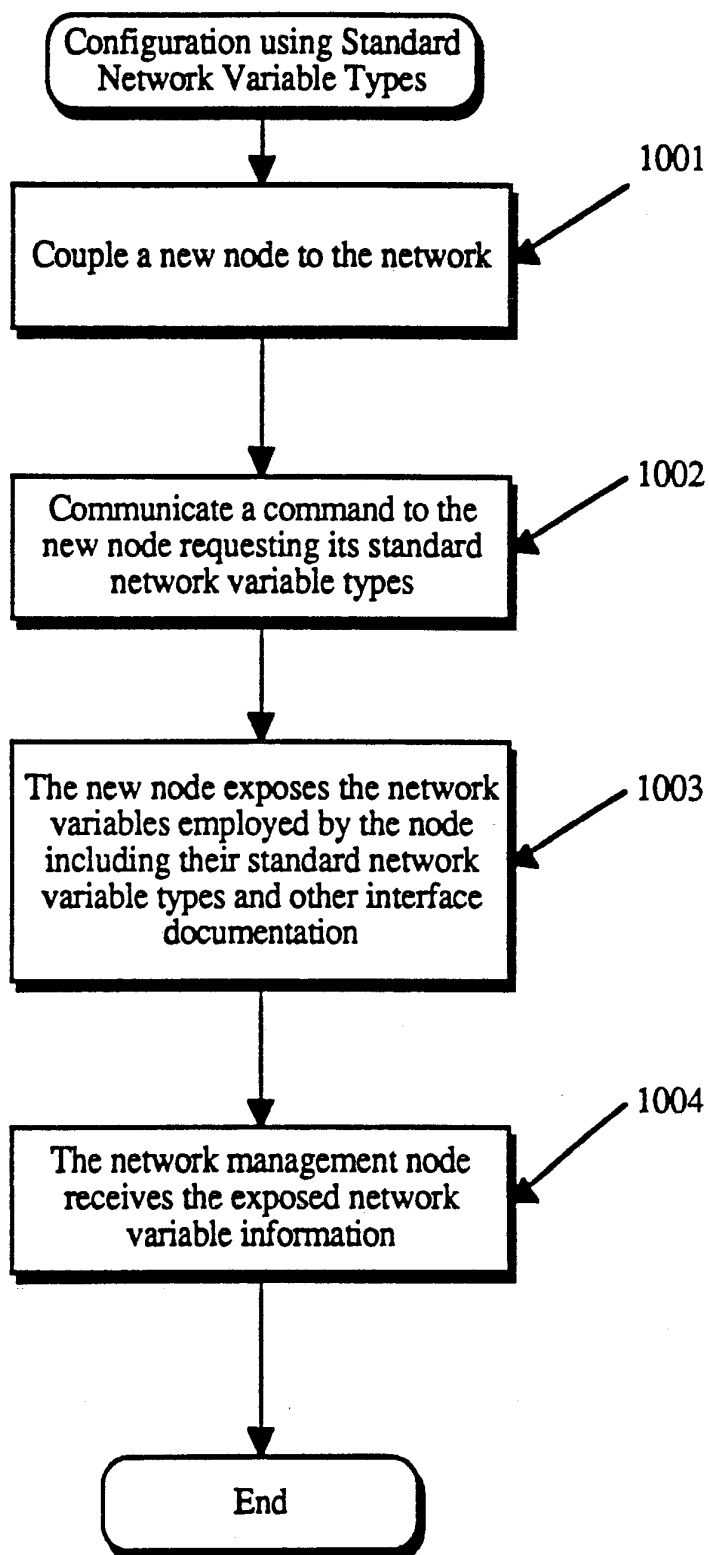


FIG. 10



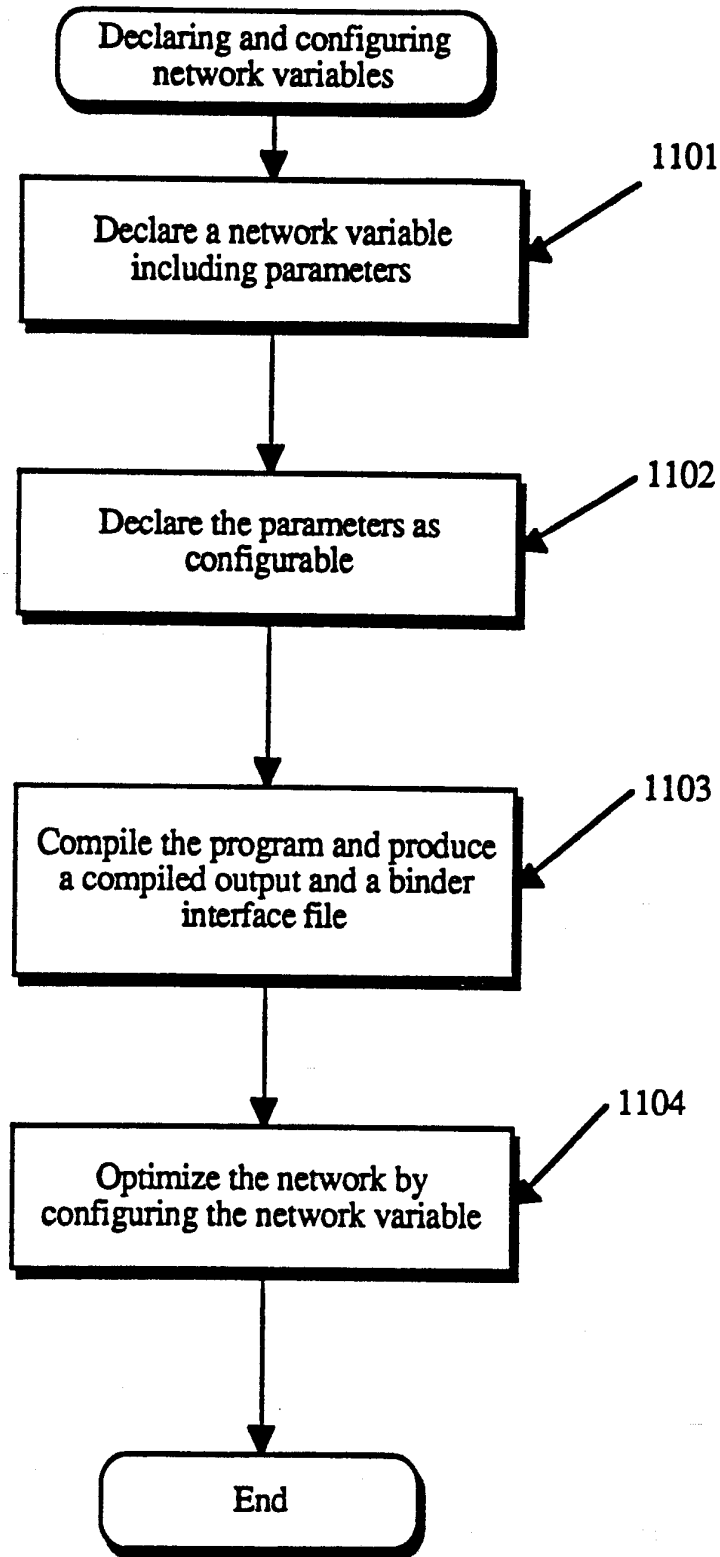


FIG. 11

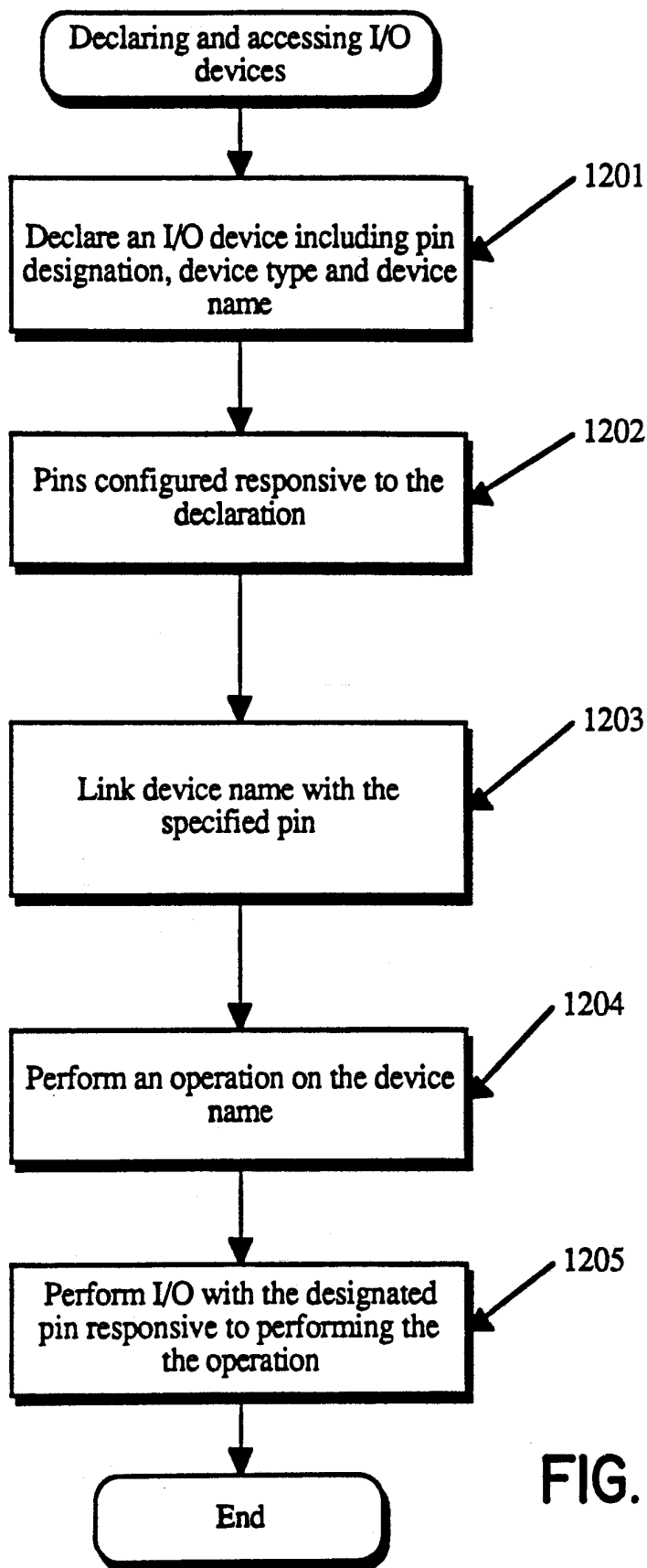
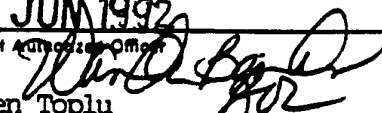


FIG. 12

# INTERNATIONAL SEARCH REPORT

International Application No. PCT/US92/02179

<b>I. CLASSIFICATION OF SUBJECT MATTER</b> <small>(Indicate the classification symbols used, indicate the class)</small>				
<small>According to International Patent Classification (IPC) or to both National Classification and IPC</small>				
IPC(5) G06F 15/40, 15/46, 15/56, G05B 19/00				
US Cl 395 600,700,725,200 364/468				
<b>II. FIELDS SEARCHED</b>				
<small>Minimum Documentation Searched *</small>				
<small>Classification System</small>	<small>Classification Symbols</small>			
US	395/600,700,725,200; 364/468			
<small>Documentation Searched other than Minimum Documentation to the extent that such Documents are included in the Fields Searched *</small>				
<b>III. DOCUMENTS CONSIDERED TO BE RELEVANT</b>				
<small>Category *</small>	<small>Citation of Document, with indication where appropriate, of the relevant passages **</small>	<small>Relevant to Claim No. :</small>		
Y,P	US, A 5093916(KART ET AL) 03 March 1992 See the entire document	1-15		
Y	US, A 4937760 (BEITEL ET AL) 26 June 1990 See the entire document	1-15		
Y	US, A 4843545 (KIKUCH()) 27 June 1989 See the entire document	1-15		
Y	AUTOMATIC PROGRAMMING OF COMMUNICATIONS SOFTWARE VIA NONPROCEDURAL DESCRIPTIONS, GINSPARG ET AL, IEEE TRANSACTION ON COMMUNICATIONS, VOL. COM-30, NO.6, June 1982, See the entire article	5-13		
Y	US, A 4727575 (HANSEN ET AL) 23 February 1988 See the entire document	1-15		
Y	US, A 4974151 (ADVANI ET AL) 27 November 1990 See the entire document	14-15		
<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; border: none; vertical-align: top;"> <p><small>* Special categories of cited documents: 10</small></p> <p><small>"A" document defining the general state of the art which is not considered to be of particular relevance</small></p> <p><small>"E" earlier document but published on or after the international filing date</small></p> <p><small>"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)</small></p> <p><small>"O" document referring to an oral disclosure, use, exhibition or other means</small></p> <p><small>"P" document published prior to the international filing date but later than the priority date claimed</small></p> </td> <td style="width: 50%; border: none; vertical-align: top;"> <p><small>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</small></p> <p><small>X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</small></p> <p><small>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.</small></p> <p><small>&amp;" document member of the same patent family</small></p> </td> </tr> </table>			<p><small>* Special categories of cited documents: 10</small></p> <p><small>"A" document defining the general state of the art which is not considered to be of particular relevance</small></p> <p><small>"E" earlier document but published on or after the international filing date</small></p> <p><small>"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)</small></p> <p><small>"O" document referring to an oral disclosure, use, exhibition or other means</small></p> <p><small>"P" document published prior to the international filing date but later than the priority date claimed</small></p>	<p><small>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</small></p> <p><small>X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</small></p> <p><small>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.</small></p> <p><small>&amp;" document member of the same patent family</small></p>
<p><small>* Special categories of cited documents: 10</small></p> <p><small>"A" document defining the general state of the art which is not considered to be of particular relevance</small></p> <p><small>"E" earlier document but published on or after the international filing date</small></p> <p><small>"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)</small></p> <p><small>"O" document referring to an oral disclosure, use, exhibition or other means</small></p> <p><small>"P" document published prior to the international filing date but later than the priority date claimed</small></p>	<p><small>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</small></p> <p><small>X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</small></p> <p><small>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.</small></p> <p><small>&amp;" document member of the same patent family</small></p>			
<b>IV. CERTIFICATION</b>				
<small>Date of the Actual Completion of the International Search</small>	<small>Date of Mailing of this International Search Report</small>			
22 May 1992	24 JUN 1992			
<small>International Searching Authority</small>	<small>Signature of Authorizing Officer</small>			
ISA/US	 Lucien Toplu			

III DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

Category \* Citation of Document, with indication, where appropriate, of the relevant passages Relevant to Claim No

Y US, A 4589063 (SHAH ET AL) 13 May 1986  
See the entire document

14-15

## FURTHER INFORMATION CONTINUED FROM THE SECOND SHEET

Y	US, A 4926375 (MERCER ET AL) 15 May 1990 See the entire document	1-15
Y	US, A 4720782 (KOVALCIN) 19 January 1988 See the entire document	1-15
Y	US, A 3582901 (COCHRANE) 01 June 1971 See the entire document	1-15
Y	US, A 4885684 (AUSTIN ET AL) 05 December 1989 See the entire document	1-15

 **OBSERVATIONS WHERE CERTAIN CLAIMS WERE FOUND UNSEARCHABLE**

This international search report has not been established in respect of certain claims under Article 17(2) (a) for the following reasons:

- 1  Claim numbers \_\_\_\_\_ because they relate to subject matter not required to be searched by this Authority, namely:
- 2  Claim numbers \_\_\_\_\_ because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out<sup>11</sup>, specifically:
- 3  Claim numbers \_\_\_\_\_ because they are dependent claims not drafted in accordance with the second and third sentences of PCT Rule 6.4(a).

 **OBSERVATIONS WHERE UNITY OF INVENTION IS LACKING**

This International Searching Authority found multiple inventions in this international application as follows:

- 1  As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims of the international application.
- 2  As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims of the international application for which fees were paid, specifically claims:
- 3  No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claim numbers:
- 4  As all searchable claims could be searched without effort justifying an additional fee, the International Searching Authority did not invite payment of any additional fee.

Remark on Protest

- The additional search fees were accompanied by applicant's protest.
- No protest accompanied the payment of additional search fees.