US 20070094429A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2007/0094429 A1**

(43) **Pub. Date: Apr. 26, 2007**

Wingard et al.

(54) **COMMUNICATIONS SYSTEM AND METHOD WITH MULTILEVEL CONNECTION IDENTIFICATION**

(76) Inventors: **Drew E. Wingard**, Palo Alto, CA (US); **Jay S. Tomlinson**, San Jose, CA (US)

Correspondence Address:
BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030 (US)

(21) Appl. No.: **11/546,131**

(22) Filed: **Oct. 10, 2006**

Related U.S. Application Data

(63) Continuation of application No. 10/787,942, filed on Feb. 25, 2004, now Pat. No. 7,120,712, which is a continuation of application No. 09/721,044, filed on Nov. 21, 2000, now Pat. No. 6,725,313, which is a continuation of application No. 09/191,291, filed on Nov. 13, 1998, now Pat. No. 6,182,183.

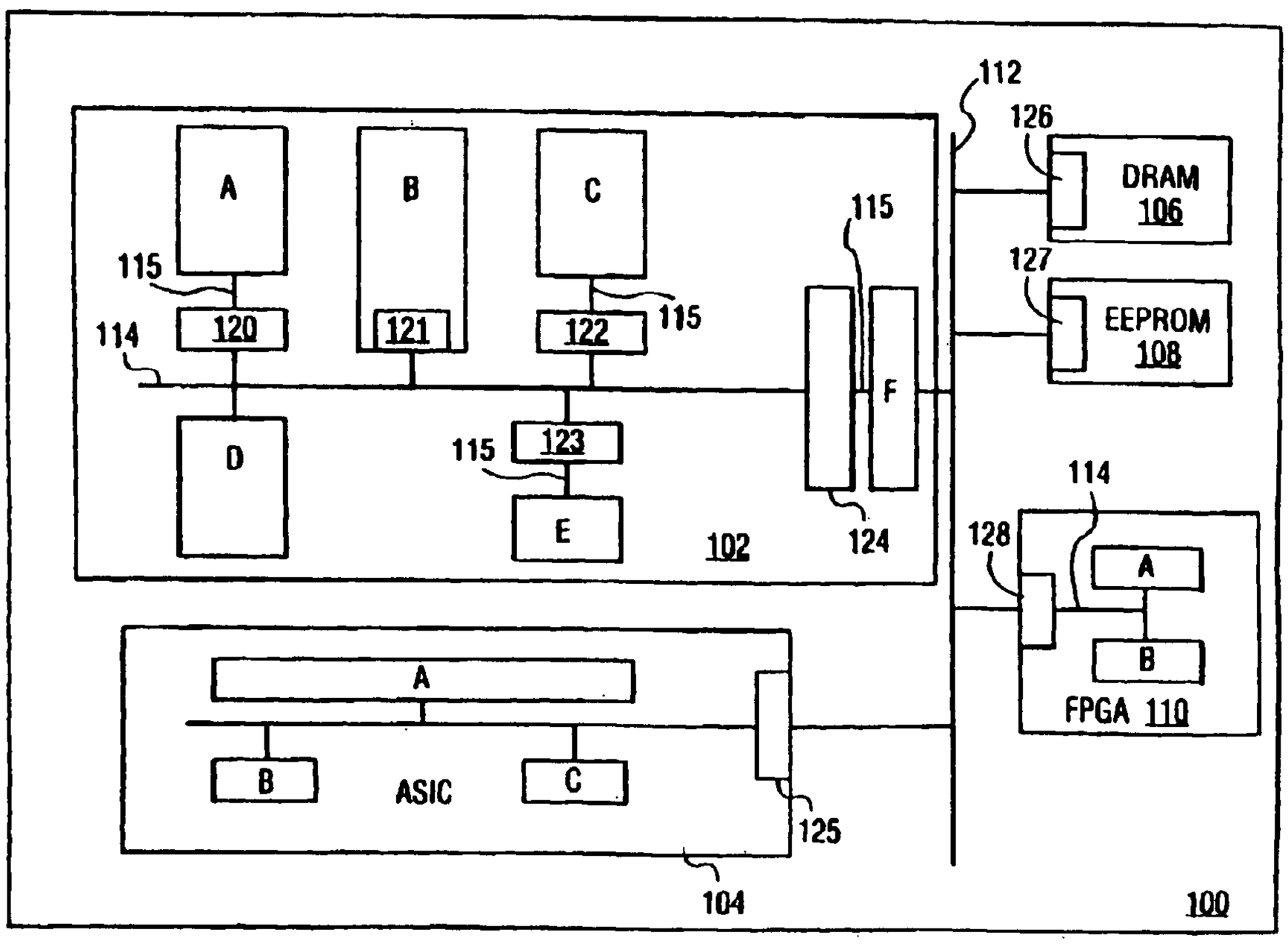
Publication Classification

(51) **Int. Cl.**
G06F 13/42 (2006.01)

(52) **U.S. Cl.** **710/106**

(57) **ABSTRACT**

A communication system. One embodiment includes at least two functional blocks, wherein an first functional block communicates with a second functional block by establishing a connection, wherein a connection is a logical state in which data may pass between the first functional block and the second functional block. One embodiment includes a bus coupled to each of the functional blocks and configured to carry a plurality of signals. The plurality of signals includes a connection identifier that indicates a particular connection that a data transfer is part of, and a thread identifier that indicates a transaction stream that the data transfer is part of.



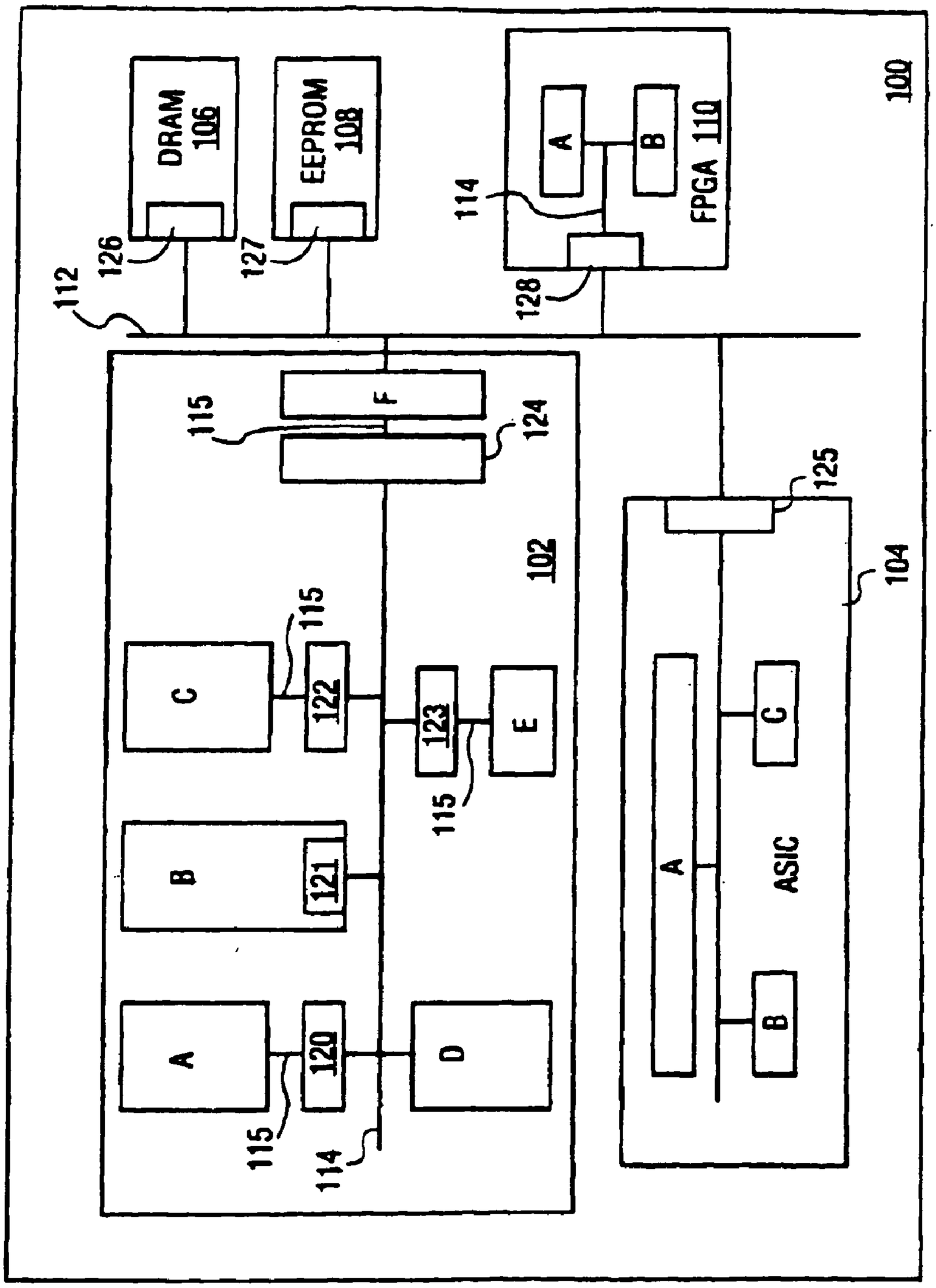

100
DRAM 106
EEPROM 108
FPGA 110
A
B
112
126
127
114
128
102
F
115
124
C
122
123
E
121
B
A
120
D
115
114
104
125
A
C
ASIC
B

FIG. 1

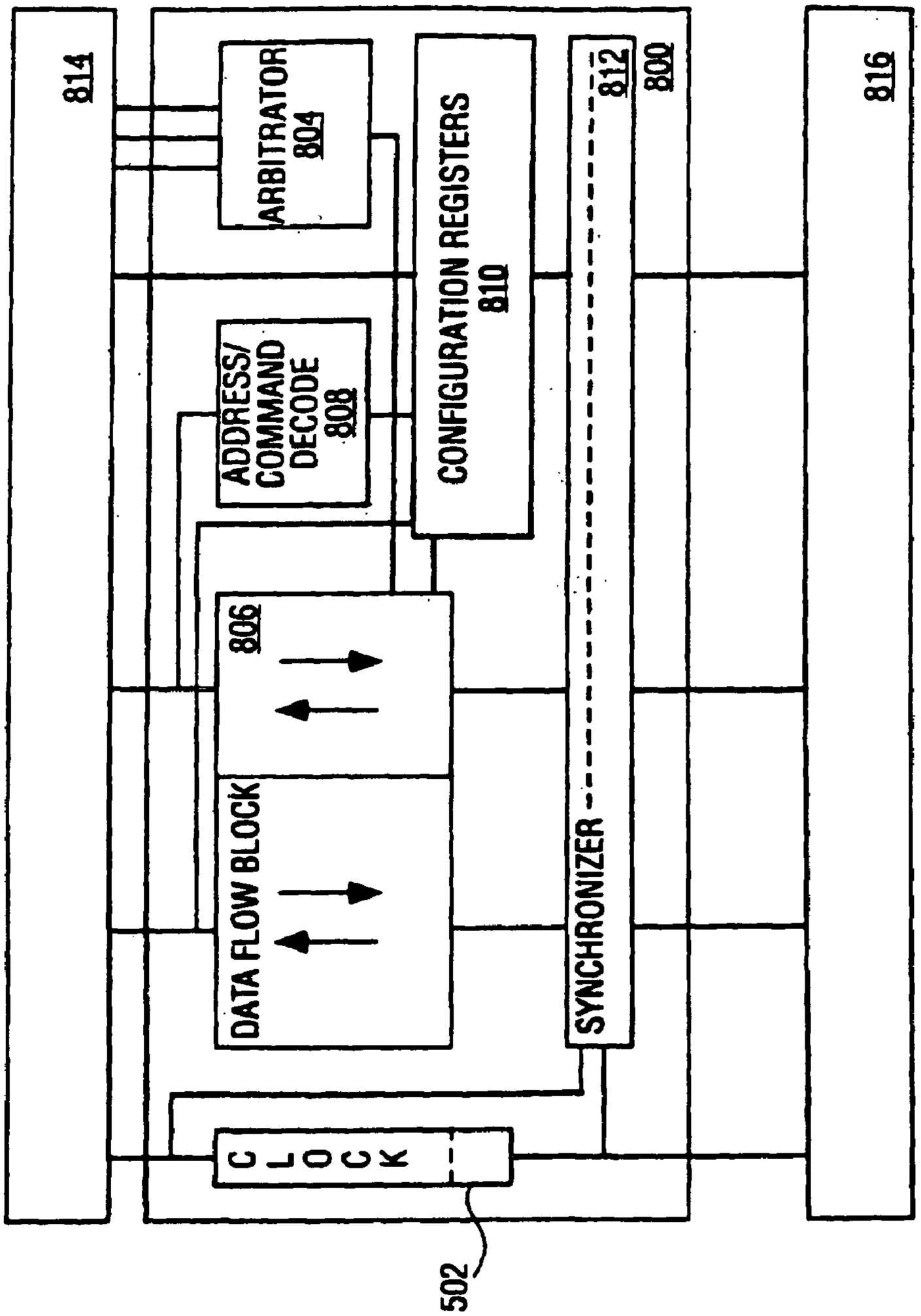

814
ARBITRATOR
804
ADDRESS/
COMMAND
DECODE
808
CONFIGURATION REGISTERS
810
812
800
816
806
DATA FLOW BLOCK
SYNCHRONIZER
CLOCK
502

FIG. 2

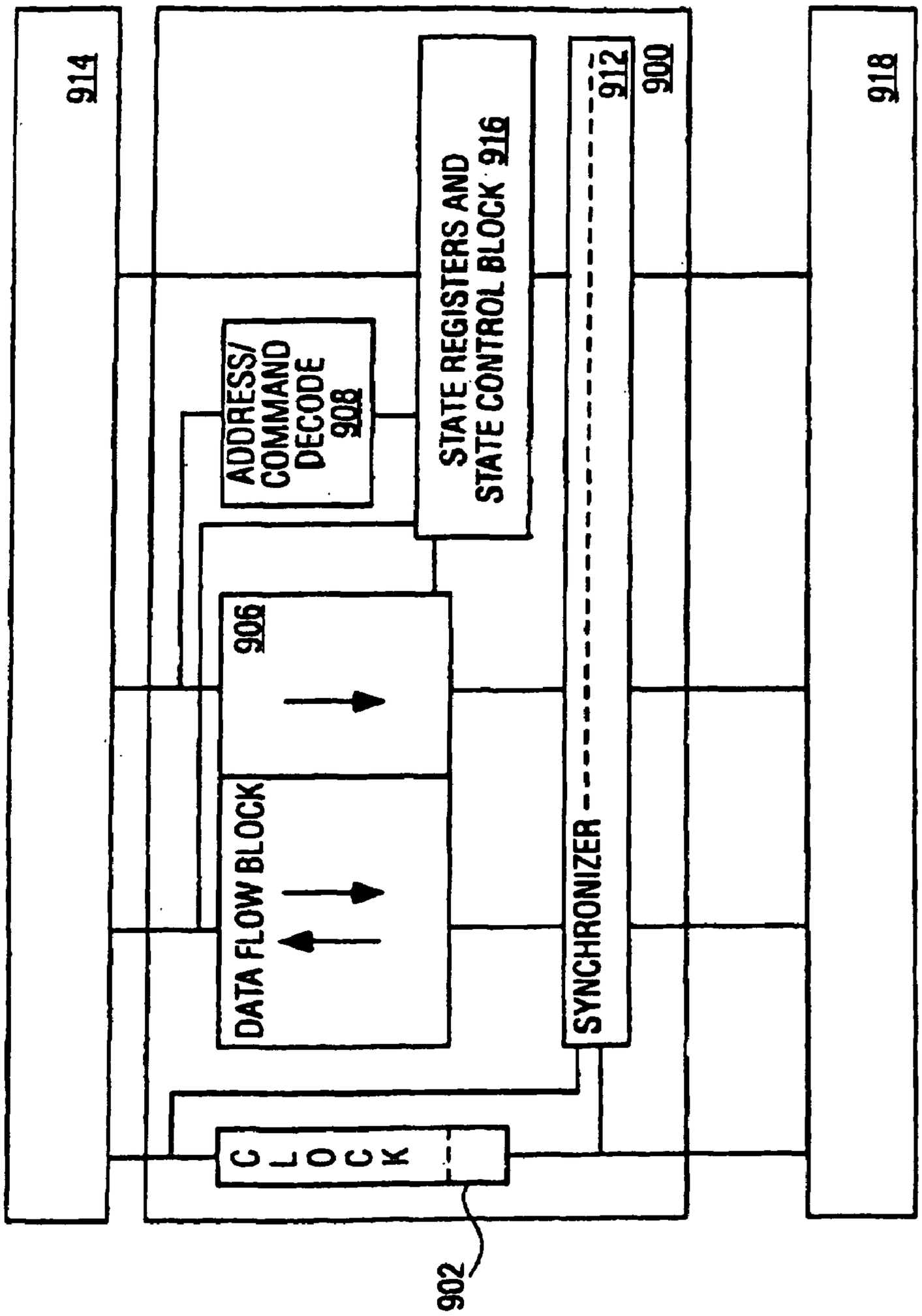

914
ADDRESS/ COMMAND DECODE 908
STATE REGISTERS AND STATE CONTROL BLOCK 916
912
900
918
906
DATA FLOW BLOCK
SYNCHRONIZER
CLOCK
902

FIG. 3

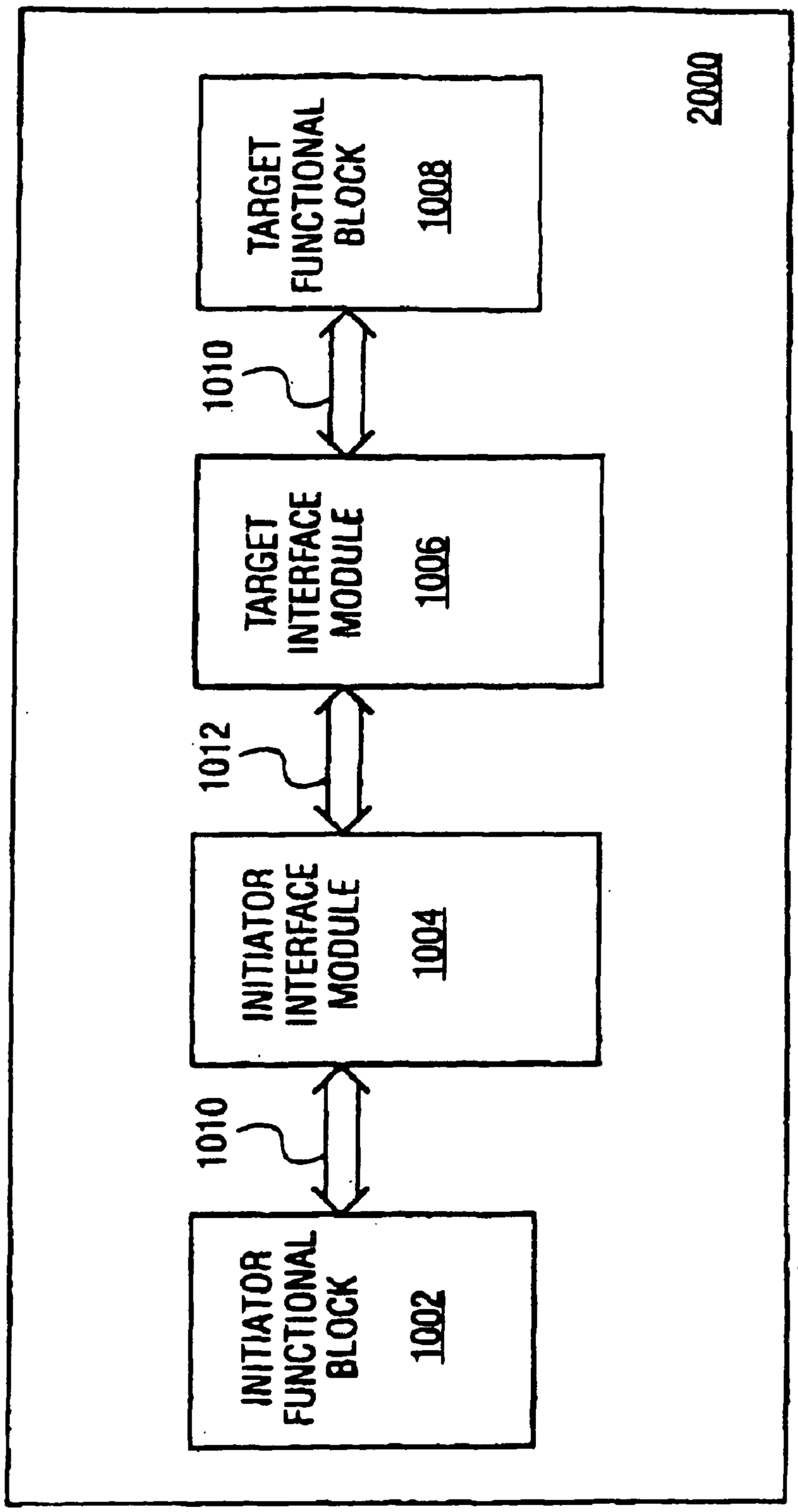

2000
TARGET FUNCTIONAL BLOCK
1008
1010
TARGET INTERFACE MODULE
1006
1012
INITIATOR INTERFACE MODULE
1004
1010
INITIATOR FUNCTIONAL BLOCK
1002

FIG. 4

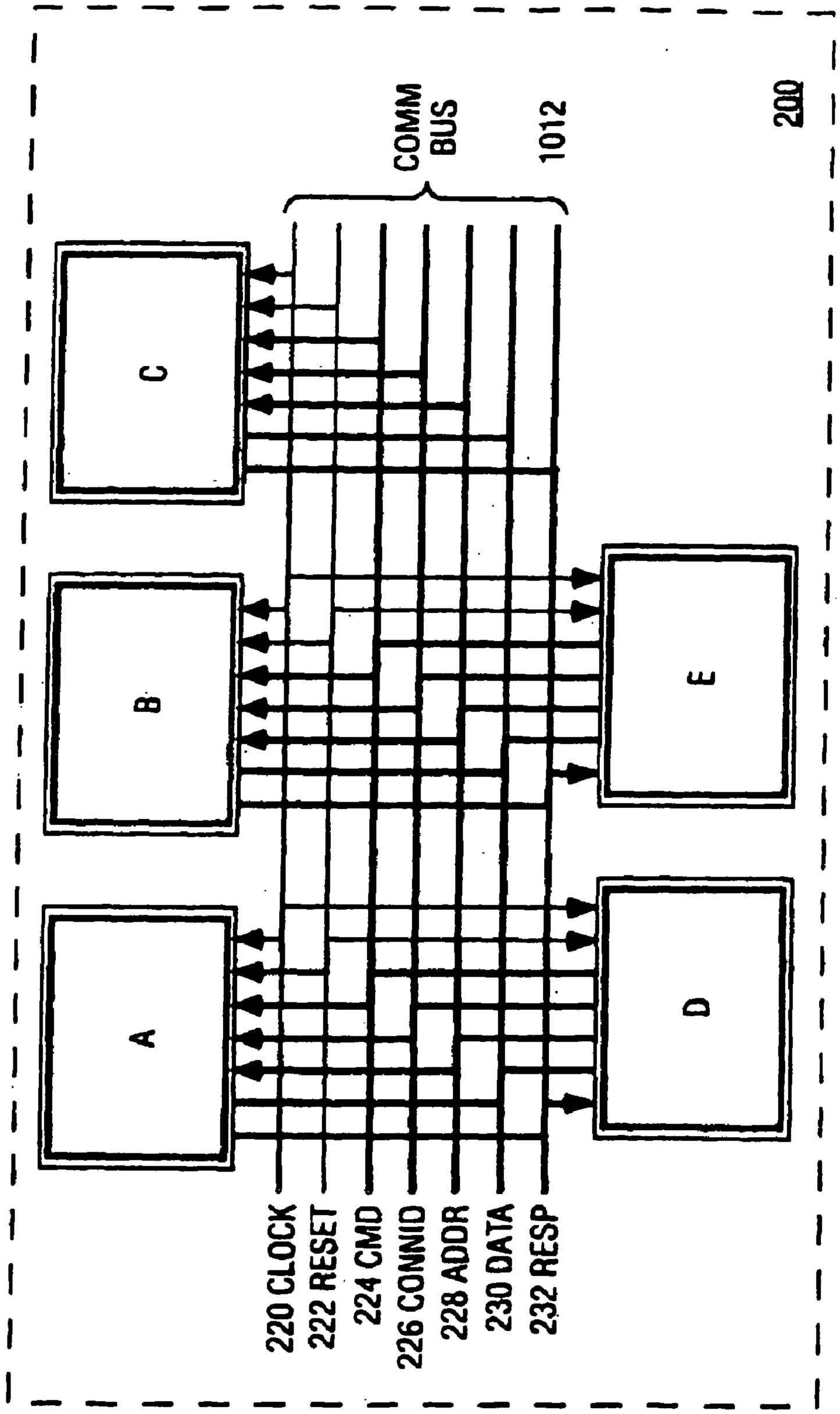

200
COMM BUS
1012
A
B
C
D
E
220 CLOCK
222 RESET
224 CMD
226 CONNID
228 ADDR
230 DATA
232 RESP

FIG. 5

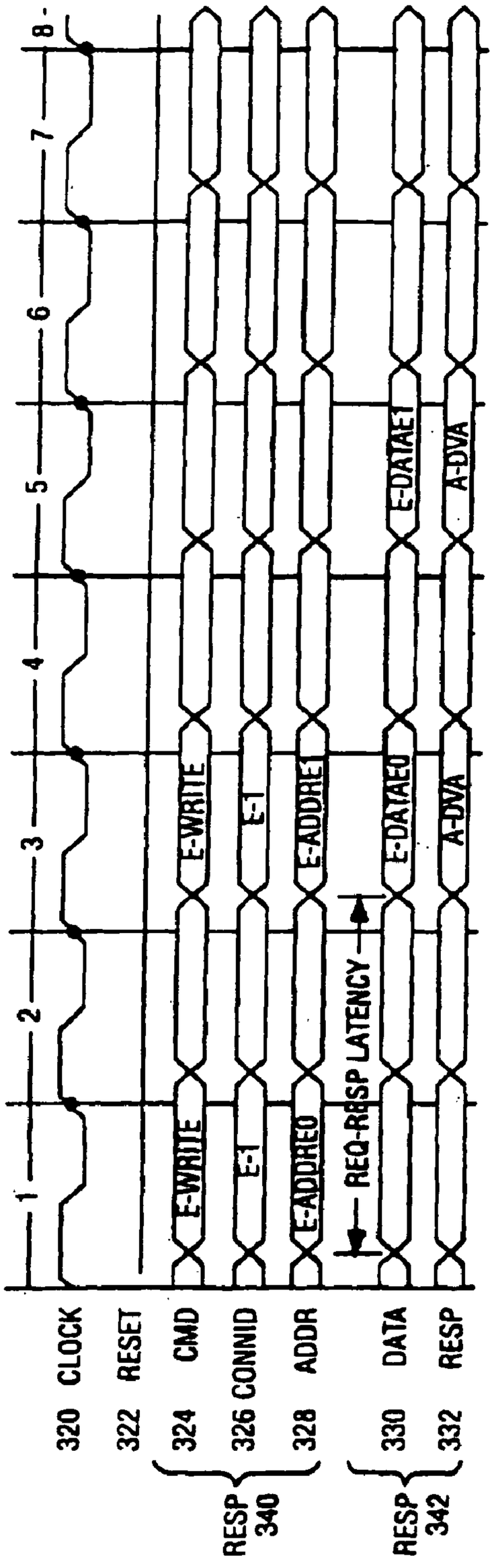

1
2
3
4
5
6
7
8
320 CLOCK
322 RESET
324 CMD
326 CONNID
328 ADDR
330 DATA
332 RESP
E-WRITE
E-1
E-ADDREQ
REQ-RESP LATENCY
E-DATAEQ
A-DVA
E-ADDRE1
E-DATAE1
RESP 340
RESP 342

FIG. 6

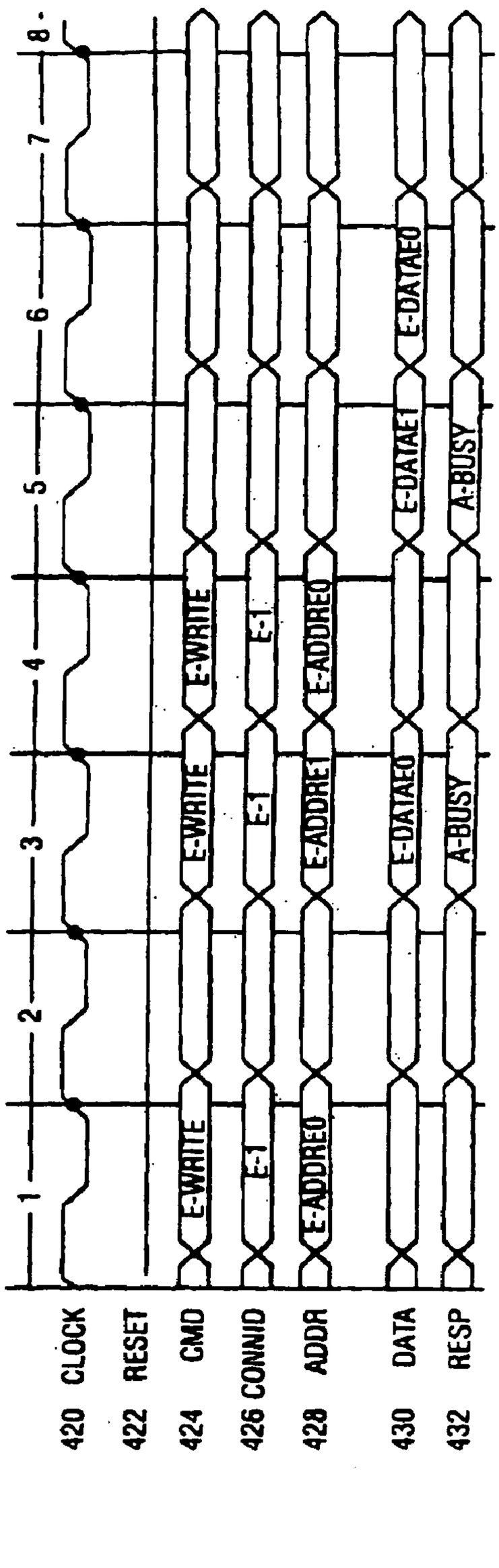

1
2
3
4
5
6
7
8
420 CLOCK
422 RESET
424 CMD
426 CONNID
428 ADDR
430 DATA
432 RESP
E-WRITE
E-1
E-ADDREQ
E-DATAEQ
A-BUSY
E-ADDRE1
E-DATAE1

FIG. 7

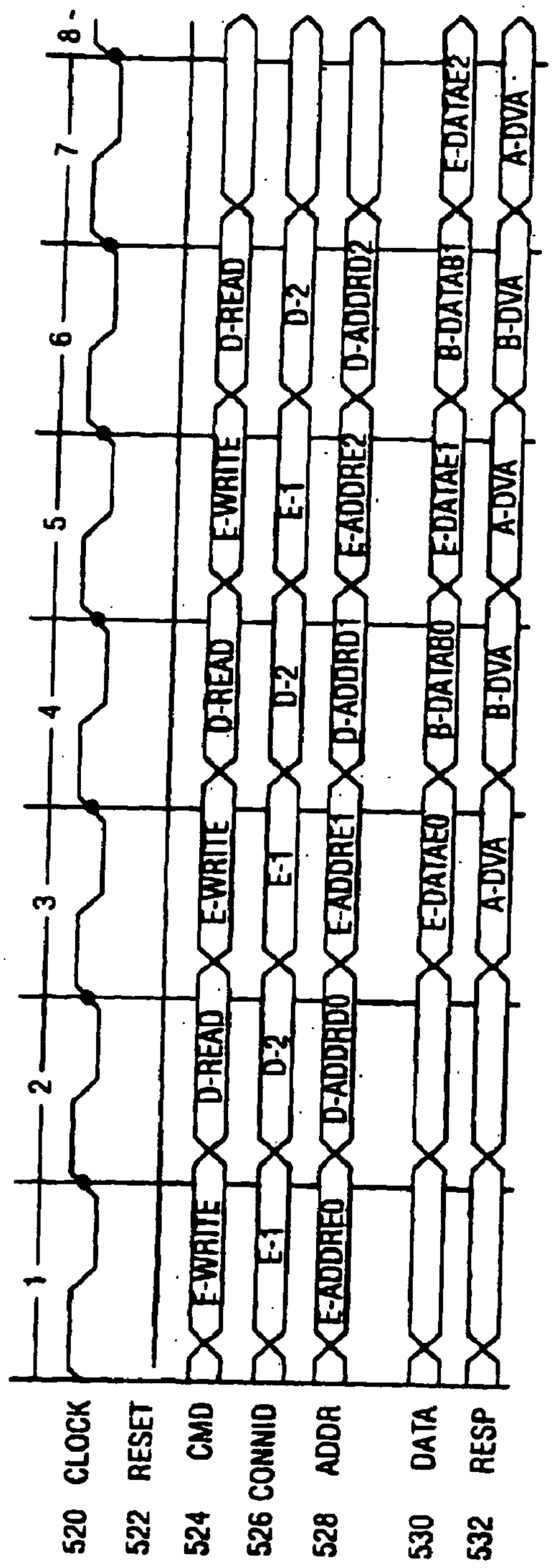

8
7
6
5
4
3
2
1
520 CLOCK
522 RESET
524 CMD
526 CONNID
528 ADDR
530 DATA
532 RESP
E-WRITE
D-READ
E-1
D-2
E-ADDRE0
D-ADDRD0
E-DATAE0
A-DVA
E-WRITE
D-READ
E-1
D-2
E-ADDRE1
D-ADDRD1
B-DATAB0
B-DVA
E-WRITE
D-READ
E-1
D-2
E-ADDRE2
D-ADDRD2
B-DATAB1
E-DATAE1
B-DVA
E-DATAE2
A-DVA

FIG. 8

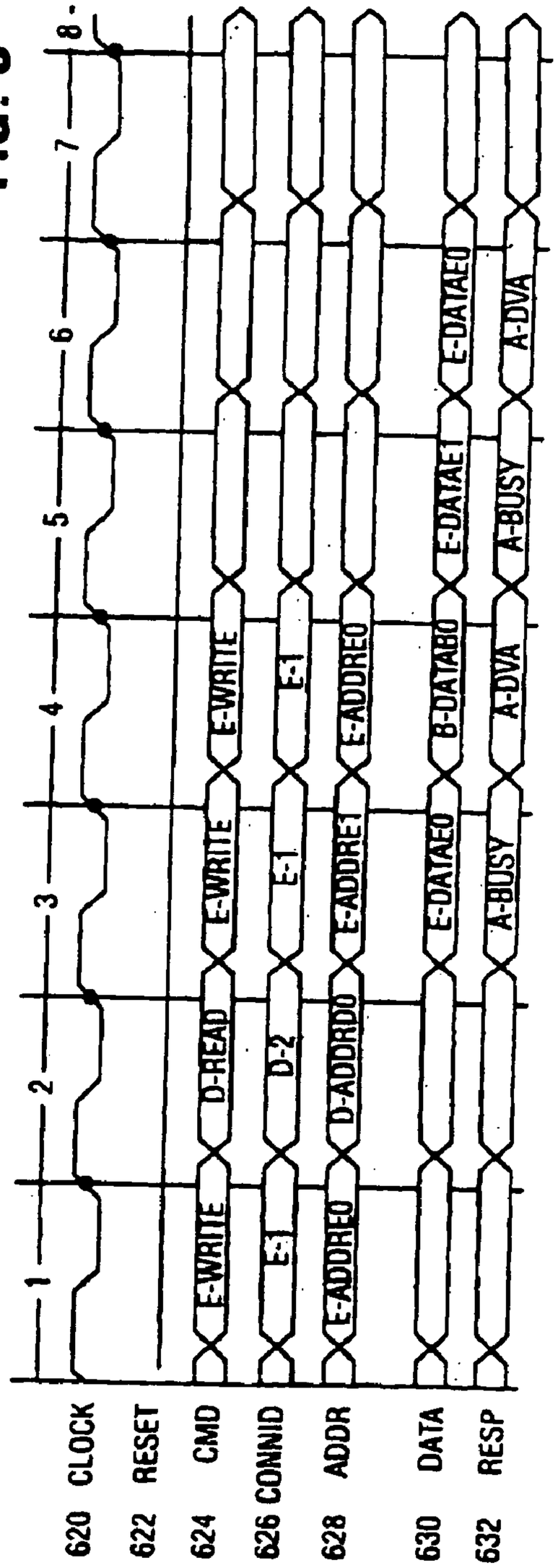

8
7
6
5
4
3
2
1
620 CLOCK
622 RESET
624 CMD
626 CONNID
628 ADDR
630 DATA
632 RESP
E-WRITE
D-READ
E-1
D-2
E-ADDRE0
D-ADDRD0
E-DATAE0
A-BUSY
E-WRITE
D-READ
E-1
D-2
E-ADDRE1
D-ADDRD1
B-DATAB0
A-DVA
E-DATAE1
E-DATAE0
A-BUSY
A-DVA

FIG. 9

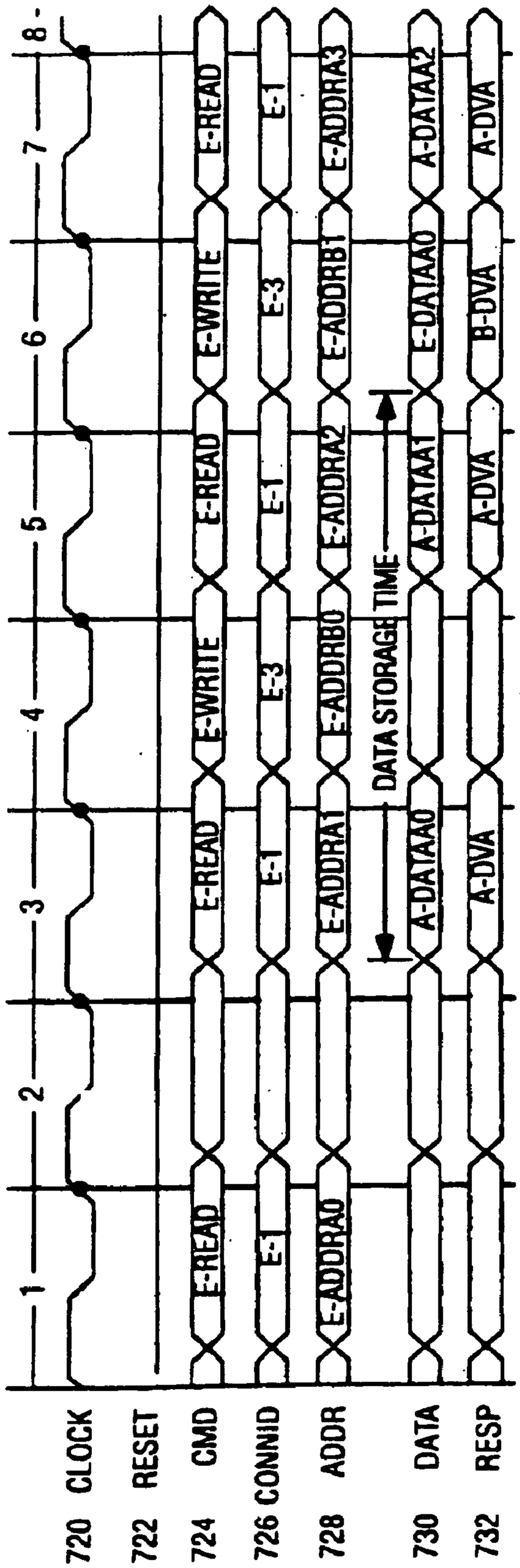

1
2
3
4
5
6
7
8
720 CLOCK
722 RESET
724 CMD
726 CONNID
728 ADDR
730 DATA
732 RESP
E-READ
E-WRITE
E-READ
E-READ
E-1
E-3
E-1
E-1
E-ADDRA0
E-ADDRB0
E-ADDRA1
E-ADDRA2
E-ADDRA3
A-DATAA0
A-DATAA1
A-DATAA2
A-DVA
B-DVA
DATA STORAGE TIME

FIG. 10

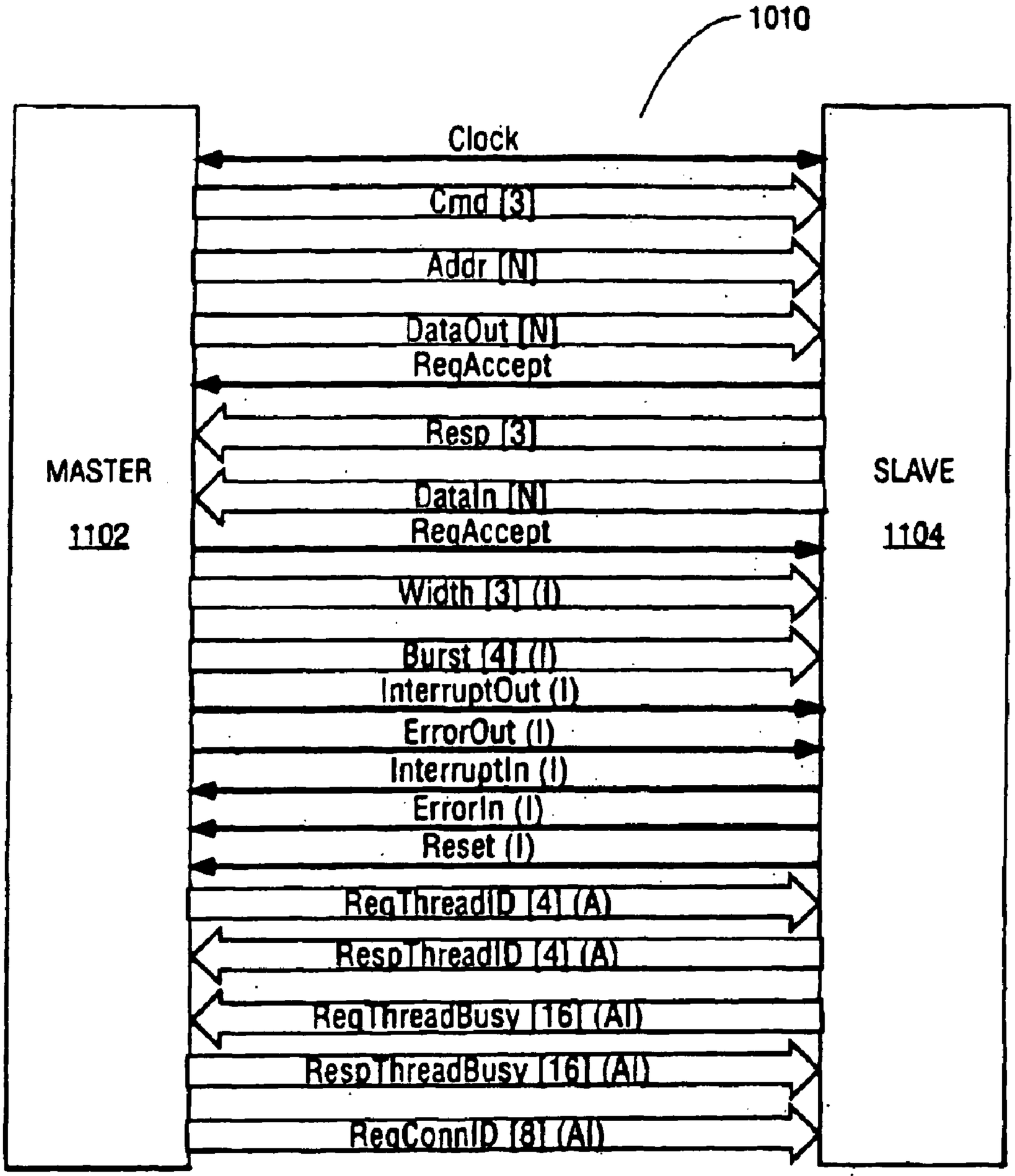
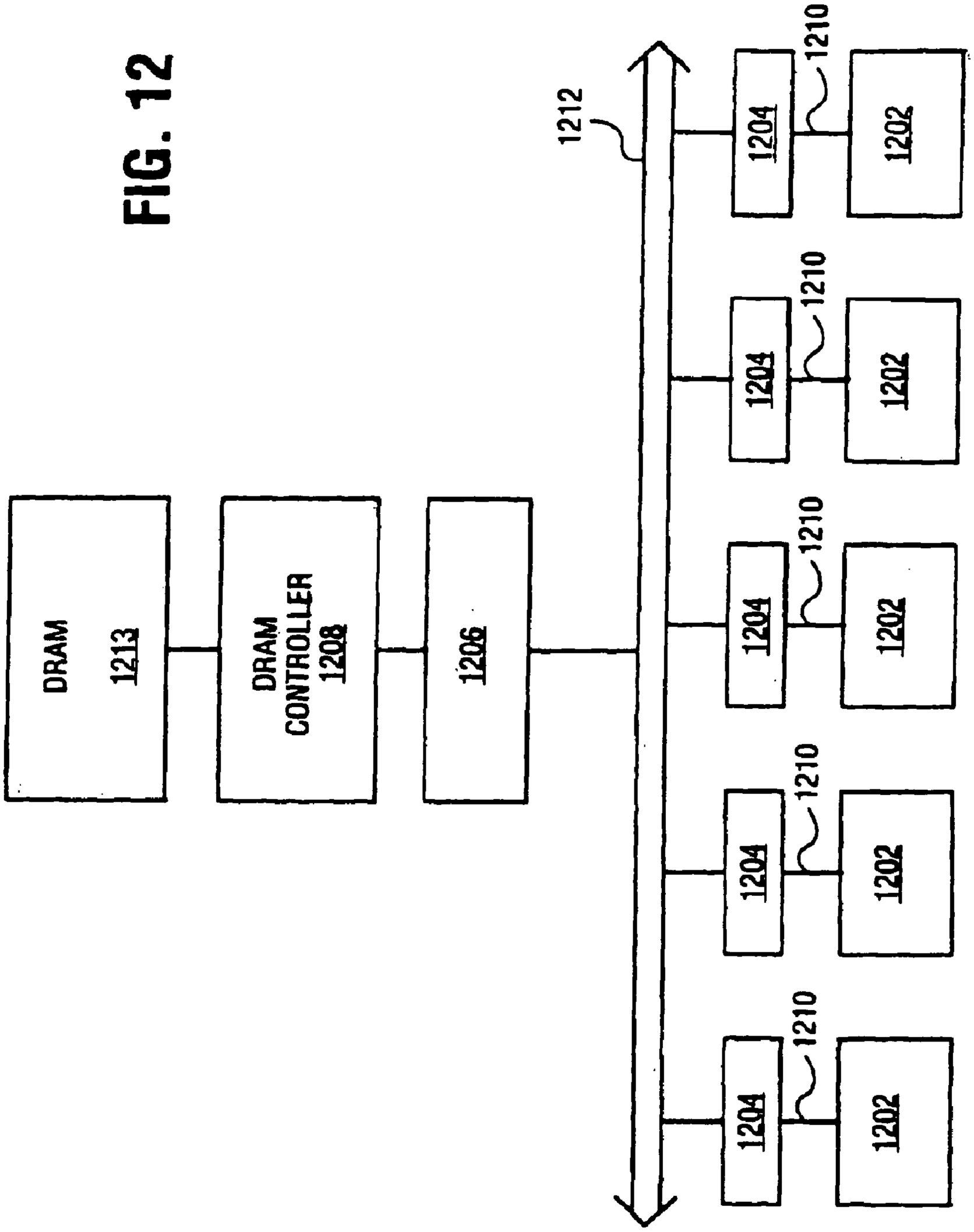

1010
Clock
Cmd [3]
Addr [N]
DataOut [N]
ReqAccept
Resp [3]
DataIn [N]
ReqAccept
Width [3] (I)
Burst [4] (I)
InterruptOut (I)
ErrorOut (I)
InterruptIn (I)
ErrorIn (I)
Reset (I)
ReqThreadID [4] (A)
RespThreadID [4] (A)
ReqThreadBusy [16] (AI)
RespThreadBusy [16] (AI)
ReqConnID [8] (AI)
MASTER
1102
SLAVE
1104

FIG. 11

FIG. 12



COMMUNICATIONS SYSTEM AND METHOD WITH MULTILEVEL CONNECTION IDENTIFICATION

RELATED APPLICATIONS

[0001] This application is a continuation application of and claims the benefit of and incorporates in by reference the following U.S. application Ser. No. 10/789,942 filed Feb. 25, 2004, which is which is a continuation of U.S. Pat. No. 6,725,313 filed Nov. 21, 2000, which is a continuation of U.S. Pat. No. 6,182,183 filed Nov. 13,1998.

FIELD OF THE INVENTION

[0002] The present invention relates to a communication system to couple computing sub-systems.

BACKGROUND OF THE INVENTION

[0003] Electronic computing and communications systems continue to include greater numbers of features and to increase in complexity. At the same time, electronic computing and communications systems decrease in physical size-and cost per function. Rapid advances in semiconductor technology such as four-layer deep-sub-micron complimentary metal-oxide semiconductor (CMOS) technology, have enabled true "system-on-a-chip" designs. These complex designs may incorporate, for example, one or more processor cores, a digital signal processing (DSP) core, several communications interfaces, and graphics support in application-specific logic. In some systems, one or several of these extremely complex chips must communicate with each other and with other system components. Significant new challenges arise in the integration, verification and testing of such systems because efficient communication must take place between sub-systems on a single complex chip as well as between chips on a system board. One benefit to having an efficient and flexible method for communication between sub-systems and chips is that system components can be reused in other systems with a minimum of redesign.

[0004] One challenge in the integration, verification and testing of modern electronic systems stems from the fact that modern electronic systems in many application areas have functionality, cost and form-factor requirements that mandate the sharing of resources, such as memory, among multiple functional blocks, where functional blocks can be any entity that interfaces to a communication system. In such systems, the functional blocks typically possess different performance characteristics and requirements, and the communications system and shared resources must simultaneously satisfy the total requirements. Key requirements of typical functional blocks are bandwidth and latency constraints that can vary over several orders of magnitude between functional blocks. In order to simultaneously satisfy constraints that vary so widely, communications systems must provide high degrees of predictability.

[0005] Traditional approaches to the design of communications systems for modern, complex computer systems have various strengths and weaknesses. An essential aspect of such approaches is the communications interface that various sub-systems present to one another. One approach is to define customized point-to-point interfaces between a sub-system and each peer with which it must communicate. This customized approach offers protocol simplicity, guaranteed performance, and isolation from dependencies on unrelated sub-systems. Customized interfaces, however, are by their nature inflexible. The addition of a new sub-system with a different interface requires design rework.

[0006] A second approach is to define a system using standardized interfaces. Many standardized interfaces are based on pre-established computer bus protocols. The use of computer buses allows flexibility in system design, since as many different functional blocks may be connected together as required by the system, as long as the bus has sufficient performance. It is also necessary to allocate access to the bus among various sub-systems. In the case of computer buses, resource allocation is typically referred to as arbitration.

[0007] One disadvantage of computer buses is that each sub-system or component connected to the bus is constrained to use the protocol of the bus. In some cases, this limits the performance of the sub-system. For example, a sub-system may be capable of handling multiple transaction streams simultaneously, but the bus protocol is not capable of fully supporting concurrent operations. In the case of a sub-system handling multiple transaction streams where each transaction stream has ordering constraints, it is necessary for the sub-system to identify each increment of data received or transmitted with a certain part of a certain data stream to distinguish between streams and to preserve order within a stream. This includes identifying a sub-system that is a source of a data transmission. Conventionally, such identification is limited to a non-configurable hardware identifier that is generated by a particular sub-system or component.

[0008] Current bus systems provide limited capability to preserve order in one transaction stream by supporting "split transactions" in which data from one transaction may be interleaved with data from another transaction in the same stream. In such a bus, data is tagged as belonging to one stream of data, so that it can be identified even if it arrives out of order. This requires the receiving sub-system to decode an arriving address to extract the identification information.

[0009] Current bus systems do not support true concurrency of operations for a sub-system that can process multiple streams of transactions over a single interconnect, such as a memory controller that handles access to a single dynamic random access memory (DRAM) for several clients of the DRAM. A DRAM controller may require information related to a source of an access request, a priority of an access request, ordering requirements, etc. Current communication systems do not provide for such information to be transmitted with data without placing an additional burden on the sub-system to adapt to the existing protocol.

[0010] In order for many sub-systems to operate in conventional systems using all of their capabilities, additional knowledge must be designed into the sub-systems to provide communication over existing communication systems. This makes sub-systems more expensive and less flexible in the event the sub-system is later required to communicate with new sub-systems or components. Existing communication approaches thus do not meet the requirements of today's large, complex electronics systems. Therefore, it is desirable for a communications system and mechanism to allow sub-systems of a large, complex electronics system to interoperate efficiently regardless of their varying performance characteristics and requirements.

SUMMARY OF THE INVENTION

[0011] One embodiment of the present invention includes a shared communications bus for providing flexible communication capability between electronic sub-systems. One embodiment includes a protocol that allows for identification of data transmissions at different levels of detail as required by a particular sub-system without additional knowledge being designed into the sub-system.

[0012] One embodiment of the invention includes several functional blocks, including at least one initiator functional block and one target functional block. Some initiator functional blocks may also function as target functional blocks. In one embodiment, the initiator functional block is coupled to an initiator interface module and the target functional block is coupled to a target interface module. The initiator functional block and the target functional block communicate to their respective interface modules and the interface modules communicate with each other. The initiator functional block communicates with the target functional block by establishing a connection, wherein a connection is a logical state in which data may pass between the initiator functional block and the target functional block.

[0013] One embodiment also includes a bus configured to carry multiple signals, wherein the signals include a connection identifier signal that indicates a particular connection that a data transfer between an initiator functional block and a target functional block is part of. The connection identifier includes information about the connection, such as which functional block is the source of a transmission, a priority of a transfer request, and transfer ordering information. One embodiment also includes a thread identifier, which provides a subset of the information provided by the connection identifier. In one embodiment, the thread identifier is an identifier of local scope that identifies transfers between an interface module and a connected functional block, where in some embodiments, an interface module connects a functional block to a shared communications bus.

[0014] The connection identifier is a an identifier of global scope that transfers information between interface modules or between functional blocks through their interface modules. Some functional blocks may require all the information provided by the connection identifier, while other functional blocks may require only the subset of information provided by the thread identifier.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] FIG. **1** is a block diagram of one embodiment of a complex electronics system according to the present invention.

[0016] FIG. **2** is an embodiment of an interface module.

[0017] FIG. **3** is an embodiment of an interface module.

[0018] FIG. **4** is an embodiment of a communications bus.

[0019] FIG. **5** is a timing diagram showing pipelined write transfers.

[0020] FIG. **6** is a timing diagram showing rejection of a first pipelined write transfer and a successful second write transfer

[0021] FIG. **7** is a timing diagram showing interleaving of pipelined read and write transfers.

[0022] FIG. **8** is a timing diagram showing interleaved connections to a single target.

[0023] FIG. **9** is a timing diagram showing interleaved connections from a single initiator.

[0024] FIG. **10** is a block diagram of one embodiment of part of a computer system.

[0025] FIG. **11** is one embodiment of a communications bus.

[0026] FIG. **12** is a block diagram of one embodiment of part of a computer system.

DETAILED DESCRIPTION OF THE INVENTION

[0027] The present invention is a communications system and method for allowing multiple functional blocks or sub-systems of a complex electronics system to communicate with each other through a shared communications resource, such as a shared communications bus. In one embodiment, a communications protocol allows multiple functional block on a single semiconductor device to communicate to each other. In another embodiment, the communications protocol may be used to allow multiple functional blocks on different semiconductor devices to communicate to each other through a shared off-chip communications resource, such as a bus.

[0028] In one embodiment, the present invention is a pipelined communications bus with separate command, address, and data wires. Alternative embodiments include a pipelined communications bus with multiplexed address, data, and control signals. The former embodiment offers higher performance and simpler control than the latter embodiment at the expense of extra wires. The former embodiment may be more appropriate for on-chip communications, where wires are relatively less expensive and performance requirements are usually higher. The latter embodiment offers higher per-wire transfer efficiency, because it shares the same wires among address and data transfers. The latter embodiment may be more appropriate for chip-to-chip communications between semiconductor devices, because package pins and board traces increase the per signal cost, while total required communications performance is usually lower.

[0029] FIG. **1** is a block diagram of a complex electronics system **100**. Shared communications bus **112** connects sub-systems **102**, **104**, **106**, **108**, and **110**. Sub-systems are typically functional blocks including a interface module for interfacing to a shared bus. Sub-systems may themselves include one or more functional blocks and may or may not include an integrated or physically separate interface module. In one embodiment, the sub-systems connected by communications bus **112** are separate integrated circuit chips. Sub-system **104** is an application specific integrated circuit (ASIC) which, as is known, is an integrated circuit designed to perform a particular function. Sub-system **106** is a dynamic random access memory (DRAM). Sub-system **108** is an erasable, programmable, read only memory (EPROM). Sub-system **110** is a field programmable gate array (FPGA). Sub-system **102** is a fully custom integrated circuit designed specifically to operate in system **100**. Other embodiments may contain additional sub-systems of the same types as shown, or other types not shown. Other

embodiments may also include fewer sub-systems than the sub-systems shown in system **100**. Integrated circuit **102** includes sub-systems **102**A, **102**B, **102**C, **102**D and **102**E. ASIC **104** includes functional blocks **101**A, **104**B and **104**C. FPGA **110** includes functional blocks **110**A and **110**B. A functional block may be a particular block of logic that performs a particular function. A functional block may also be a memory component on an integrated circuit.

[**0030**] System **100** is an example of a system that may consist of one or more integrated circuits or chips. A functional block may be a logic block on an integrated circuit such as, for example, functional block **102**E, or a functional block may also be an integrated circuit such as fully custom integrated circuit **102** that implements a single logic function.

[**0031**] Shared communications bus **112** provides a shared communications bus between sub-systems of system **100**. Shared communication bus **114** provides a shared communications bus between sub-systems or functional blocks on a single integrated circuit. Some of the functional blocks shown are connected to interface modules through which they send and receive signals to and from shared communications bus **112** or shared communications bus **114**. Interconnect **115** is a local point-to-point interconnect for connecting interface modules to functional blocks.

[**0032**] Interface modules **120-127** are connected to various functional blocks as shown. In this embodiment, interface modules **120**,**122**,**123** and **124** are physically separated from their connected functional block (A, B, C, E and F, respectively). Interface modules **121**, and **125-128** are essentially part of their respective functional blocks or sub-systems. Some functional blocks, such as **102**D, do not require a dedicated interface module. The arrangement of sub-systems, functional blocks and interface modules is flexible and is determined by the system designer.

[**0033**] In one embodiment there are four fundamental types of functional blocks. The four fundamental types are initiator, target, bridge, and snooping blocks. A typical target is a memory device, a typical initiator is a central processing unit (CPU). A typical bridge might connect shared communications buses **112** and **114**. Functional blocks all communicate with one another via shared communications bus **112** or shared communications bus **114** and the protocol of one embodiment. Initiator and target functional blocks may communicate a shared communications bus through interface modules. An initiator functional block may communicate with a shared communications bus through an initiator interface module and a target functional block may communicate with a shared communications bus through a target interface module.

[**0034**] An initiator interface module issues and receives read and write requests to and from functional blocks other than the one with which it is associated. In one embodiment, an initiator interface module is typically connected to a CPU, a digital signal processing (DSP) core, or a direct memory access (DMA) engine.

[**0035**] FIG. **2** is a block diagram of an embodiment of an initiator interface module **800**. Initiator interface module **800** includes clock generator **802**, data flow block **806**, arbitrator block **804**, address/command decode block **808**, configuration registers **810**, and synchronizer **812**. Initiator interface module **800** is connected to a shared communications bus **814** and to an initiator functional block **816**. In one embodiment, shared communications bus **814** is a shared communications bus that connects sub-systems, as bus **112** does in FIG. **1**.

[**0036**] Clock generator **802** is used to perform clock division when initiator functional block **816** runs synchronously with respect to shared communications bus **814** but at a different frequencies. When initiator functional block **816** runs asynchronously with respect to communications bus **814**, clock generator **802** is not used, but synchronizer **812** is used. Arbitrator block **804** performs arbitration for access to shared communications bus **814**. In one embodiment, a multi-level arbitration scheme is used wherein arbitrator module **804** includes logic circuits that manage pre-allocated bandwidth aspects of first level arbitration and also logic that manages second level arbitration. Data flow block **806** includes data flow first-in-first-out (FIFO) buffers between shared communications bus **814** and initiator functional block **816**, in addition to control logic associated with managing a transaction between shared communications bus **814** and initiator functional block **816**. The FIFO buffers stage both the address and data bits transferred between shared communications bus **814** and initiator functional block **816**. In one embodiment, shared communications bus **814** implements a memory mapped protocol. Specific details of an underlying computer bus protocol are not significant to the invention, provided that the underlying computer bus protocol supports some operation concurrency. A preferred embodiment of a bus protocol for use with the present invention is one that supports retry transactions or split transactions, because these protocols provide a mechanism to deliver operation concurrency by interrupting a multi-cycle transaction to allow transfers belonging to other unrelated transactions to take place. These protocols allow for higher transfer efficiencies because independent transactions may use the bus while an initiator waits for a long latency target to return data that has been previously requested by an initiator.

[**0037**] Address/command decode block **808** decodes an address on shared communications bus **814** to determine if a write is to be performed to registers associated with initiator functional block **816**. Address/command decode block **808** also decodes incoming commands. Configuration registers **810** store bits that determine the state of module **800**, including bandwidth allocation and client address base. One register **810** stores an identification (ID) which is a set of bits uniquely identifying initiator functional block **816**.

[**0038**] FIG. **3** is a block diagram of an embodiment of a target interface module **900**. Target interface module **900** is connected to shared communications bus **914** and to target functional block **918**. Target interface module **900** includes clock generator **902**, data flow block **906**, address/command decode block **908**, synchronizer **912**, and state registers in state control block **916**. Blocks of target interface module **900** that are named similarly to blocks of initiator module **800** function in substantially the same way as explained with respect to initiator block **800**. State registers and state control block **916** include registers that store, for example, client address base and an identifier for target functional block **918**.

[**0039**] In one embodiment, an initiator functional block such as initiator functional block **816** may also act as a target

functional block in that it has the capability to respond to signals from other functional blocks or sub-systems as well as to initiate actions by sending signals to other functional blocks or sub-system.

[0040] FIG. **4** is a block diagram of a part of a computer system **1000** according to one embodiment. FIG. **4** is useful in illustrating multilevel connection identification. System **1000** includes initiator functional block **1002**, which is connected to initiator interface module **1004** by interconnect **1010**. Initiator interface module **1004** is connected to target interface module **1006** by shared communications bus **1012**. Target interface module **1006** is connected to target functional block **1008** by an interconnect **1010**. Typically, shared communications bus **1012** is analogous to shared communications bus **112** of FIG. **1** or to shared communications bus **114** of FIG. **1**. Interconnects **1010** are typically analogous to interconnect **115** of FIG. **1** in that they connect functional blocks to interface modules and are point-to-point, rather than shared, interconnects. Interconnects **1010** are typically physically shorter than shared communications bus **1012** because of their local nature. As will be explained more fully below, system **1000** uses two different levels of connection identification depending upon the requirements of a particular functional block. "Global" connection identification information is sent on shared communications bus **1012**, while "local" connection information, or thread identification information, is sent in interconnects **1010**.

[0041] FIG. **5** is a block diagram of one embodiment of a shared communications bus **1012**. Shared communications bus **1012** is shown connected to entities A, B, C, D and E, which may be interface modules, functional blocks, or a combination of both. Shared communications bus **1012** is composed of a set of wires. Data wires **230** provide direct, high efficiency transport of data traffic between functional blocks on shared communications bus **1012**. In one embodiment, shared communications bus **1012** supports a bus protocol that is a framed, time division multiplexed, fully pipelined, fixed latency communication protocol using separate address, data and connection identification wires. The bus protocol supports fine grained interleaving of transfers to enable high operation concurrency, and uses retry transactions to efficiently implement read transactions from target devices with long or variable latency. Details of the arbitration method used to access shared communications bus **1012** are not required to understand the present invention. The delay from when an initiator functional block drives the command and address until the target functional block drives the response is known as the latency of shared communications bus **1012**. The bus protocol supports arbitration among many initiator functional blocks and target functional blocks for access to the bus. In the embodiment shown, arbitration for access to shared communications bus **1012** is performed by an initiator interface module, such as module **1004** of FIG. **4**. In other embodiments, arbitration is performed by functional blocks directly, or by a combination of interface modules and functional blocks. In one embodiment, a bus grant lasts for one pipelined bus cycle. The protocol does not forbid a single functional block from becoming a bus owner for consecutive bus cycles, but does require that the functional block successfully win arbitration on consecutive cycles to earn the right.

[0042] Shared communications bus **1012** includes separate address, data, and control wires. Other embodiments may include multiplexed address, data, and control signals that share a wire or wires. Such an embodiment would provide high per-wire transfer efficiency because wires are shared among address and data transfers. A non-multiplexed embodiment of shared communications bus **1012** may be more appropriate for communication between functional blocks on a single integrated circuit chip because wires are relatively inexpensive and performance requirements are usually higher on a single integrated circuit chip.

[0043] Clock line **220** is a global signal wire that provides a time reference signal to which all other shared communications bus **1012** signals are synchronized. Reset line **222** is a global signal wire that forces each connected functional block into a default state from which system configuration may begin. Command line **224** carries a multi-bit signal driven by an initiator bus owner. In various embodiments, the multi-bit command signal may convey various types of information. For example, a command signal may indicate a transfer type, information regarding duration of a connection, and expected initiator and target behavior during the connection. In one embodiment, the command signal includes one or more bits indicating the beginning and end of a connection. In one embodiment, for example, one bit may indicate the status of a connection. If the bit is zero, the current transfer is the final transfer in the connection. After the receipt of a zero connection status bit, the next receipt of a connection status bit that is a logic one indicates that the transfer is the first in a newly opened connection. Each subsequently received one connection status bit then indicates that the connection is still open.

[0044] Supported transfer types in this embodiment include, but are not limited to read and write transfers. Address lines **228** carry a multi-bit signal driven by an initiator bus owner to specify the address of the object to be read or written during the current transfer. Response lines **232** carry a multi-bit signal driven by a target to indicate the status of the current transfer. Supported responses include, but are not limited to the following responses. A NULL response indicates that the current transfer is to be aborted, presumably because the address does not select any target. A data valid and accepted (DVA) response indicates, in the case of a read, that the target is returning requested data on data lines **230**. In the case of a write, a DVA response indicates that the target is accepting the provided data from data lines **230**. A BUSY response indicates that the selected target has a resource conflict and cannot service the current request. In this case an initiator should reattempt the transfer again later. A RETRY response indicates that the selected target could not deliver the requested read data in time, but promises to do so at a later time. In this case an initiator must reattempt the transfer at a later time.

[0045] Connection identifier (CONNID) lines **226** carry a multi-bit signal driven by an initiator bus owner to indicate which connection the current transfer is part of. A connection is a logical state, established by an initiator, in which data may pass between the initiator and an associated target. The CONNID typically transmits information including the identity of the functional block initiating the transfer and ordering information regarding an order in which the transfer must be processed. In one embodiment, the information conveyed by the CONNID includes information regarding the priority of the transfer with respect to other transfers. In one embodiment the CONNID is a eight-bit code. An

initiator interface module sends a unique CONNID along with an initial address transfer of a connection. Later transfers associated with this connection (for example, data transfers) also provide the CONNID value so both sender and receiver (as well as any device monitoring transfers on shared communications bus **1012**) can unambiguously identify transfers associated with the connection. One advantage of using a CONNID is that transfers belonging to different transactions can be interleaved arbitrarily between multiple devices on a per cycle basis. In one embodiment, shared communications bus **1012** implements a fully pipelined protocol that requires strict control over transaction ordering in order to guarantee proper system operation. Without the use of a CONNID, ordering constraints within a particular transaction may be violated because transfers associated with a particular connection are not identified.

[**0046**] Because a first command may be rejected by a BUSY response while a later command is already in flight, it is essential to provide mechanisms that allow full control over which commands complete. If such control is not present, ambiguous system behavior can result. For instance, if a single initiator interface module issues a sequence of dependent read and write commands, a busy response to one of the commands could result in later commands returning the wrong data. One solution to such problems is to avoid overlapping dependent commands. This solution, however, increases the latency of every dependent command in order to ensure proper results. The present invention uses a CONNID signal, in part, to allow overlapping of dependent commands. Therefore, use of a CONNID improves system performance and efficiency. Another benefit of the CONNID of the present invention is that communication system predictability is enhanced because it allows a shared functional block to respond to requests based upon quality of service guarantees that may vary between connections. For example, data requested to operate a computer display cannot tolerate unpredictable delay because delay causes the display to flicker. Therefore, the CONNID may be used to prioritize data requests from a display controller so that requests from the display controller to a common resource are serviced before other requests. The present invention also allows for flexible reconfiguration of the CONNID to retune system performance.

[**0047**] FIG. **6** is a timing diagram of a pipelined write transaction consisting of two write transfers on shared communications bus **1012**. Reference may also be made to FIG. **5**. A single pipelined bus transfer, as shown in FIG. **6**, includes an arbitration cycle (not shown), followed by a command/address/CONNID (CMD **324**/ADDR **328**/CONNID **326**) cycle (referred to as a request, or REQ cycle), and completed by a DATA **330**/RESP **342** cycle (referred to as a response, or RESP cycle). In one embodiment, the number of cycles between a REQ cycle and a RESP cycle is chosen at system implementation time based upon the operating frequency and module latencies to optimize system performance. The REQ-RESP latency, in one embodiment, is two cycles and is labeled above the DATA **330** signal line on FIG. **6**. Therefore, a complete transfer time includes four shared communications bus **1012** cycles, arbitration, request, delay and response.

[**0048**] Two transfers are shown in FIG. **6**. On cycle **1**, initiator E drives REQ fields **340** to request a WRITE transfer to address ADDRE**0**. This process is referred to as issuing the transfer request. In one embodiment, a single target is selected to receive the write data by decoding an external address portion of ADDRE**0**. On cycle **3** (a REQ-RESP latency later), initiator E drives write data DATAE**0** on the DATA wires; simultaneously, the selected target A drives RESP wires **342** with the DVA code, indicating that A accepts the write data. By the end of cycle **3**, target A has acquired the write data, and initiator E detects that target A was able to accept the write data; and the transfer has thus completed successfully.

[**0049**] Meanwhile (i.e. still in cycle **3**), initiator E issues a pipelined WRITE transfer (address ADDRE**1**) to target A. The write data and target response for this transfer both occur on cycle **5**, where the transfer completes successfully. Proper operation of many systems and sub-systems rely on the proper ordering of related transfers. Thus, proper system operation may require that the cycle **3** WRITE complete after the cycle **1** WRITE transfer. In FIG. **6**, the CONNID field conveys crucial information about the origin of the transfer that can be used to enforce proper ordering. A preferred embodiment of ordering restrictions is that the initiator and target collaborate to ensure proper ordering, even during pipelined transfers. This is important, because transfer pipelining reduces the total latency of a set of transfers (perhaps a single transaction), thus improving system performance (by reducing latency and increasing usable bandwidth).

[**0050**] According to the algorithm of one embodiment:

[**0051**] 1. An initiator may issue a transfer Y: a) if transfer Y is the oldest, non-issued, non-retired transfer among the set of transfer requests it has with matching CONNID, or b) if all of the older non-retired transfers with matching CONNID are currently issued to the same target as transfer Y. If issued under this provision, transfer Y is considered pipelined with the older non-retired transfers.

[**0052**] 2. A target that responds to a transfer X in such a way that the initiator might not retire the transfer must respond BUSY to all later transfers with the same CONNID as transfer X that are pipelined with X.

[**0053**] Note that an older transfer Y that is issued after a newer transfer X with matching CONNID is not considered pipelined with X, even if Y Issues before X completes. This situation is illustrated in FIG. **7**. If target A has a resource conflict that temporarily prevents it from accepting DATAE**0** associated with the WRITE ADDRE**0** from cycle **1**, then A responds BUSY. Step **2** of the foregoing algorithm requires that A also reject (using BUSY) any other pipelined transfers from the same CONNID (in this case, CONNID **1**), since the initiator cannot possibly know about the resource conflict until after the REQ-RESP latency has passed. Thus, target A must BUSY the WRITE ADDRE**1** that is issued in cycle **3**, because it has the same CONNID and was issued before the initiator could interpret the BUSY response to the first write transfer, and is thus a pipelined transfer. Furthermore, the second attempt (issued in cycle **4**) of the WRITE ADDRE**0** transfer is allowed to complete because it is not a pipelined transfer, even though it overlaps the cycle **3** WRITE ADDRE**1** transfer.

[**0054**] Note that target A determines that the cycle **4** write is not pipelined with any earlier transfers because of when

it occurs and which CONNID it presents, and not because of either the CMD nor the ADDR values. Step **1** of the algorithm guarantees that an initiator will only issue a transfer that is the oldest non-issued, non-retired transfer within a given connection. Thus, once the first WRITE ADDRE**0** receives the BUSY response in cycle **3**, it is no longer issued, and so it becomes the only CONNID=1 transfer eligible for issue. It is therefore impossible for a properly operating initiator to issue a pipelined transfer in cycle **4**, given that an initial cycle **1** transfer received a BUSY response and the REQ-RESP latency is two cycles.

[**0055**] One embodiment of the initiator maintains a time-ordered queue consisting of the desired transfers within a given CONNID. Each transfer is marked as non-issued and non-retired as they are entered into the queue. It is further marked as pipelined if the immediately older entry in the queue is non-retired and addresses the same target; otherwise, the new transfer is marked non-pipelined. Each time a transfer issues it is marked as issued. When a transfer completes (i.e., when the RESP cycle is finished) the transfer is marked non-issued. If the transfer completes successfully, it is marked as retired and may be deleted from the queue. If the transfer does not complete successfully, it will typically be re-attempted, and thus can go back into arbitration for re-issue. If the transfer does not complete successfully, and it will not be re-attempted, then it should not be marked as retired until the next transfer, if it exists, is not marked as issued. This restriction prevents the initiator logic from issuing out of order. As the oldest non-Retired transfer issues, it is marked as issued. This allows the second-oldest non-retired transfer to arbitrate to issue until the older transfer completes (and is thus marked as non-issued), if it is marked as pipelined.

[**0056**] An embodiment of the target implementation maintains a time-ordered queue whose depth matches the REQ-RESP latency. The queue operates off of the bus clock, and the oldest entry in the queue is retired on each bus cycle; simultaneously, a new entry is added to the queue on each bus cycle. The CONNID from the current REQ phase is copied into the new queue entry. In addition, if the current REQ phase contains a valid transfer that selects the target (via the External Address), then "first" and "busy" fields in the new queue entry may be set; otherwise, the first and busy bits are cleared. The first bit will be set if the current transfer will receive a BUSY response (due to a resource conflict) and no earlier transfer in the queue has the same CONNID and has its first bit set. The first bit implies that the current transfer is the first of a set of potentially-pipelined transfers that will need to be BUSY'd to enforce ordering. The busy bit is set if either the target has a resource conflict or one of the earlier transfers in the queue has the same CONNID and has the first bit set. This logic enforces the REQ-RESP pipeline latency, ensuring that the target accepts no pipelined transfers until the initiator can react to the BUSY response to the transfer marked first.

[**0057**] Application of the algorithm to the initiators and targets in the communication system provides the ability to pipeline transfers (which increases per-connection bandwidth and reduces total transaction latency) while maintaining transaction ordering. The algorithm therefore facilitates high per-connection performance. The fundamental interleaved structure of the pipelined bus allows for high system performance, because multiple logical transactions may overlap one another, thus allowing sustained system bandwidth that exceeds the peak per-connection bandwidths. For instance, FIG. **8** demonstrates a system configuration in which initiator E needs to transfer data to target A on every other bus cycle, while initiator D requests data from target B on every other bus cycle. Since the communication system supports fine interleaving (per bus cycle), the transactions are composed of individual transfers that issue at the natural data rate of the functional blocks; this reduces buffering requirements in the functional blocks, and thus reduces system cost. The total system bandwidth in this example is twice the peak bandwidth of any of the functional blocks, and thus high system performance is realized.

[**0058**] The present invention adds additional system-level improvements in the area of efficiency and predictability. First, the connection identifier allows the target to be selective in which requests it must reject to preserve in-order operation. The system only need guarantee ordering among transfers with the same CONNID, so the target must reject (using BUSY) only pipelined transfers. This means that the target may accept transfers presented with other CONNID values even while rejecting a particular CONNID. This situation is presented in FIG. **9**, which adds an interleaved read transfer from initiator D to the pipelined write transfer of FIG. **7**. All four transfers in FIG. **9** select target A, and A has a resource conflict that prevents successful completion of the WRITE ADDRE**0** that issues in cycle **1**. While the rejection of the first write prevents A from accepting any other transfers from CONNID **1** until cycle **4**, A may accept the unrelated READ ADDRD**0** request of cycle **2** if A has sufficient resources. Thus, overall system efficiency is increased, since fewer bus cycles are wasted (as would be the case if target A could not distinguish between connections).

[**0059**] Second, in one embodiment the connection identifier allows the target to choose which requests it rejects. The target may associate meanings such as transfer priority to the CONNID values, and therefore decide which requests to act upon based upon a combination of the CONNID value and the internal state of the target. For instance, a target might have separate queues for storing transfer requests of different priorities. Referring to FIG. **9**, the target might have a queue for low priority requests (which present with an odd CONNID) and a queue for high priority requests (which present with an even CONNID). Thus, the CONNID **1** WRITE ADDRE**0** request of cycle **1** would be rejected if the low-priority queue were full, whereas the CONNID **2** READ ADDRD**0** transfer could be completed successfully based upon available high-priority queue resources. Such differences in transfer priorities are very common in highly-integrated electronic systems, and the ability for the target to deliver higher quality of service to higher priority transfer requests adds significantly to the overall predictability of the system.

[**0060**] As FIG. **9** implies, the algorithm described above allows a target to actively satisfy transfer requests from multiple CONNID values at the same time. Thus, there may be multiple logical transactions in flight to and/or from the same target, provided that they have separate CONNID values. Thus, the present invention supports multiple connections per target functional block.

[**0061**] Additionally, an initiator may require the ability to present multiple transactions to the communications system

at the same time. Such a capability is very useful for initiator such as direct memory access (DMA) devices, which transfer data between two targets. In such an application, the DMA initiator would present a read transaction using a first CONNID to a first target that is the source of the data, and furthermore present a write transaction using a second CONNID to a second target that is the data destination. At the transfer level, the read and write transfers could be interleaved. This reduces the amount of data storage in the DMA initiator, thus reducing system cost. Such an arrangement is shown in FIG. **10**, where initiator E interleaves pipelined read transfers from target A with pipelined write transfers to target B. Thus, the present invention supports multiple connections per initiator functional block.

[**0062**] The control structures required to support implementation of the present invention, as described above with respect to the algorithm, are simple and require much less area than the data buffering area associated with traditional protocols that do not provide efficient fine interleaving of transfers. Thus, the present invention minimizes communication system area and complexity, while delivering high performance and flexibility.

[**0063**] Finally, the CONNID values that are associated with particular initiator transactions should typically be chosen to provide useful information such as transfer priorities but also to minimize implementation cost. It is useful to choose the specific CONNID values at system design time, so the values can be guaranteed to be unique and can be ordered to simplify comparison and other operations. Furthermore, it is frequently useful to be able to change the CONNID values during operation of the communications system so as to alter the performance and predictability aspects of the system. Preferred implementations of the present invention enable flexible system configuration by storing the CONNID values in ROM or RAM resources of the functional blocks, so they may be readily re-configured at either system build time or system run time.

[**0064**] FIG. **11** shows an interconnect **1010**, which is a point-to-point interconnect as shown in FIG. **4**. Interconnect **1010** includes additional signals as compared to the protocol described with reference to FIG. **5**. As will be explained below, some of the additional signals are particularly useful as signals sent over point-to-point interconnects such as interconnects **1010**. The protocol of interconnect **1010** controls point-to-point transfers between a master entity **1102** and a slave entity **1104** over a dedicated (non-shared) interconnect. Referring to FIG. **4**, a master entity may be, for example, initiator functional block **1002** or target interface module **1006**. A slave entity may be, for example, initiator interface module **1004** or target functional block **1008**.

[**0065**] Signals shown in FIG. **11** are labeled with signal names. In addition, some signal names are followed by a notation or notations in parentheses or brackets. The notations are as follows:

[**0066**] (I) The signal is optional and is independently configurable

[**0067**] (A) The signal must be configured together with signals having similar notations

[**0068**] (AI) The signal is independently configurable if (A) interface modules exist [#] Maximum signal width

[**0069**] The clock signal is the clock of a connected functional block. The command (Cmd) signal indicates the type of transfer on the interconnect. Commands can be issued independent of data. The address (Addr) signal is typically an indication of a particular resource that an initiator functional block wishes to access. Request Accept (ReqAccept) is a handshake signal whereby slave **1104** allows master **1102** to release Cmd, Addr and DataOut from one transfer and reuse them for another transfer. If slave **1104** is busy and cannot participate in a requested transfer, master **1102** must continue to present Cmd, Addr and DataOut. DataOut is data sent from a master to a slave, typically in a write transfer. DataIn typically carries read data.

[**0070**] Response (Resp) and DataIn are signals sent from slave **1104** to master **1102**. Resp indicates that a transfer request that was received by slave **1104** has been serviced. Response accept (RespAccept) is a handshake signal used to indicate whether the master allows the slave to release Resp and DataIn.

[**0071**] Signals Clock, Cmd, Addr, DataOut, ReqAccept, Resp, DataIn, and RespAccept, in one embodiment, make up a basic set of interface module signals. For some functional blocks, the basic set may be adequate for communication purposes.

[**0072**] In other embodiments, some or all of the remaining signals of bus **1012** may be used. In one embodiment, Width is a three-bit signal that indicates a width of a transfer and is useful in a connection that includes transfers of variable width. Burst is a multibit signal that allow individual commands to be associated within a connection. Burst provides an indication of the nature of future transfers, such as how many there will be and any address patterns to be expected. Burst has a standard end marker. Some bits of the Burst field are reserved for user-defined fields, so that a connection may be ignorant of some specific protocol details within a connection.

[**0073**] Interrupt and error signals are an important part of most computer systems. Interrupt and error signals generated by initiator or target functional blocks are shown, but the description of their functionality is dependent upon the nature of a particular functional block and is not important to understanding the invention.

[**0074**] Request Thread Identifier (ReqThreadID), in one embodiment, is a four-bit signal that provides the thread number associated with a current transaction intended for slave **1104**. All commands executed with a particular thread ID must execute in order with respect to one another, but they may execute out of order with respect to commands from other threads. Response Thread Identifier (RespThreadID) provides a thread number associated with a current response. Because responses in a thread may return out of order with respect to other threads, RespThreadID is necessary to identify which thread's command is being responded to. In one embodiment, ReqThreadID and RespThreadID are optional signals, but if one is used, both must be used.

[**0075**] Request Thread Busy (ReqThreadBusy) allows the slave to indicate to the master that it cannot take any new requests associated with certain threads. In one embodiment, the ReqThreadBusy signal is a vector having one signal per thread, and a signal asserted indicates that the associated thread is busy.

[0076] Response Thread Busy (RespThreadBusy) allows the master to indicate to the slave that it cannot take any responses (e.g., on reads) associated with certain threads. The RespThreadBusy signal is a vector having one signal per thread, and a signal asserted indicates that the associated thread is busy.

[0077] Request Connection Identifier (ReqConnID) provides the CONNID associated with the current transaction intended for the slave. CONNIDs provide a mechanism by which a system entity may associate particular transactions with the system entity. One use of the CONNID is in establishing request priority among various initiators. Another use is in associating actions or data transfers with initiator identity rather than the address presented with the transaction request.

[0078] The embodiment of FIG. **11** provides end-to-end connection identification with CONNID as well as point-to-point, or more local identification with Thread ID. A Thread ID is an identifier of local scope that simply identifies transfers between the interface module and its connected functional block. In contrast, the CONNID is an identifier of global scope that identifies transfers between two interface modules (and, if required, their connected functional blocks).

[0079] A Thread ID should be small enough to directly index tables within the connected interface module and functional block. In contrast, there are usually more CONNIDs in a system than any one interface module is prepared to simultaneously accept. Using a CONNID in place of a Thread ID requires expensive matching logic in the interface module to associate a returned CONNID with specific requests or buffer entries.

[0080] Using a networking analogy, the Thread ID is a level-2 (data link layer) concept, whereas the CONNID is more like a level-3 (transport/session layer) concept. Some functional blocks only operate at level-2, so it is undesirable to burden the functional block or its interface module with the expense of dealing with level-3 resources. Alternatively, some functional blocks need the features of level-3 connections, so in this case it is practical to pass the CONNID through to the functional block.

[0081] Referring to FIG. **4**, a CONNID is required to be unique when transferred between interface modules **1004** and **1006** on shared communications bus **1012**. The CONNID may be sent over a local interconnect, such as interconnect **1010**. In many cases, however, it is much more efficient to use only Thread ID between a functional block and its interface module. For example initiator functional block **1002** may not require all the information provided by the CONNID. Also, in some systems, multiple identical initiator functional blocks **1002** may exist with the same CONNID so that a particular target functional block **1008** receiving a transfer will not know which connection it is actually part of unless logic in initiator interface module **1004** translates the "local" CONNID to a unique "global" CONNID. The design and implementation of such a translation functionality in an interface module is complicated and expensive. In such cases, the CONNID may be sent between interface modules over shared communications bus **1012** while the Thread ID is sent between a functional block and an interface module.

[0082] In the case of an initiator functional block, a one-to-one static correspondence may exist between Thread ID and CONNID. For example if the Thread ID is "1", a single CONNID is mapped for a particular interface module, solving the problem of multiple, identical functional blocks.

[0083] In the case of a target functional block, there is a one-to-one dynamic correspondence between a Thread ID and a CONNID. If a target functional block supports two simultaneous threads, the target interface module acquires the CONNID of an open connection and associates it with a thread as needed. For example, a target interface module receives a CONNID of "7", and then maps CONNID **7** to thread "0". Thereafter, all transfers with CONNID **7** are associated with thread **0** until connection **7** is closed.

[0084] Referring to FIG. **12**, an example of a use of Thread ID, consider a series of identical direct memory access (DMA) engines in a system. In FIG. **12**, elements **1202** are identical DMA engines, each connected to an initiator interface module **1204**. Initiator interface modules **1204** are connected to shared communications bus **1212**. Target interface module **1206** is also connected to shared communications bus **1212** and transmits data from bus **1212** to DRAM controller **1208**, which is a target functional block. Target interface module **1206** is connected to DRAM controller **1208** by interconnect **1214**. DRAM controller **1208** controls access to DRAM **1213**.

[0085] A DMA a engine is an example of an initiator functional block that also functions as a target functional block. When the DMA engine is programmed by software, it acts as a target. Thereafter, the DMA engine is an initiator. Because a DMA engine performs both read and write operations, two connections can be associated with a single DMA engine. If some buffering is available in the DMA engine, read and write operations may be decoupled so that both types of operations can be performed concurrently. A read may occur from a long latency storage device which requires the read data to be buffered on the DMA engine before a write operation writes the data. In one embodiment, each of DMA engines **1202** uses a Thread ID to identify the read stream and a different Thread ID to identify the write stream. The DMA engine does not require more information, such as what other functional block participates in a transaction. Therefore, a CONNID is not required to be sent from the DMA engine **1202** to a connected interface module **1204**. Mapping of a Thread ID to a CONNID occurs in the interface module **1204**.

[0086] In one embodiment, each initiator interface module **1204** maps a unique CONNID to each of two Thread IDs from a connected DMA engine **1202**. Each of DMA engines **1202** use a single bit, for example, Thread ID of FIG. **11**, to distinguish between its two threads. For each transfer over shared communications bus a unique CONNID is sent to target interface module **1206**. The CONNID may include priority information, for example, assigning high priority to requests for graphics data. The high priority graphics data request is immediately serviced by DRAM controller **1208** while lower priority request may be required to wait.

[0087] Because intelligence is designed into the interface modules and the communications protocols, less intelligence is required of the functional block such as the DRAM controller **1208** and the DMA engines **1202**. This has the advantage of making functional blocks more portable or reusable as systems evolve. For example, a DMA engine used for a high priority application may be switched with a

DMA engine used for a lower priority application simply by changing their respective connected interface modules.

[0088] In one embodiment, target and initiator interface modules are programmed at the transistor level so that their precise function, including their CONNID assignment, is fixed at power-up. In another embodiment, the design of interface modules is in RAM so that the interface module is a reprogrammable resource. In this case, the interface module is reprogrammed, including reassignment of CONNIDs, by software.

[0089] The present invention has been described in terms of specific embodiments. For example, embodiments of the present invention have been shown as systems of particular configurations, including communications buses using particular protocols. One of ordinary skill in the art will recognize that modifications may be made without departing from the spirit and scope of the invention as set forth in the claims. For example, the present may be used in systems employing shared communications structures other than buses, such as rings, cross-bars, or meshes.

What is claimed is:

1. A communication system comprising: at least two functional blocks, wherein a first functional block communicates with a second functional block by establishing a connection, wherein the connection is a logical state in which data may pass between the first functional block and the second functional block; and a communication medium configured to carry a plurality of signals between interface modules; an initiator functional block configured to send transfer requests having a thread identifier to indicate a transaction stream that a data transfer is part of; an initiator interface module coupled to the communication medium, and coupled to receive the initiator functional block transfer requests, said initiator interface module mapping a received thread identifier to a connection identifier, said connection identifier configured to be sent with a transfer request from the initiator interface module over the communication medium, the connection identifier comprising a multi-bit value that encodes information including a transfer priority, a transfer order, and a functional block that originated the transfer, the connection identifier is one of a plurality of connection identifiers associated with the initiator functional block; a target interface module coupled to the communication medium, and coupled to a target functional block.

2. A communication system comprising: at least two functional blocks, wherein a first functional block communicates with a second functional block by establishing a connection, wherein the connection is a logical state in which data may pass between the first functional block and the second functional block; and a communication medium configured to carry a plurality of signals between interface modules; an initiator interface module coupled to the communication medium, and coupled to an initiator functional block; a target interface module coupled to the communication medium to receive transfer requests having a connection identifier, said received connection identifier having been previously configured to be sent with a transfer request from the initiator interface module over the communication medium, the connection identifier comprising a multi-bit value that encodes information including a transfer priority, a transfer order, and a functional block that originated the transfer, the connection identifier is one of a plurality of connection identifiers associated with the initiator functional block, said target interface module mapping a received connection identifier to a thread identifier to indicate a transaction stream that a data transfer is part of; and a target functional block coupled to the target interface module to receive transfer requests having the thread identifier.

* * * * *