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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 March 2004 (04.03.2004)

PCT

(10) International Publication Number
WO 2004/019239 A2

(51) International Patent Classification[7]: G06F 17/50

(21) International Application Number:
PCT/EP2003/009286

(22) International Filing Date: 21 August 2003 (21.08.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
02447158.3 21 August 2002 (21.08.2002) EP

(71) Applicant *(for all designated States except US)*: **WINDMILL MICROSYSTEMS HOLDING BV** [NL/NL]; Blokvenlaan 47, NL-5581 GM Waalre (NL).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **VAN DALEN, Rokus, H., J.** [NL/NL]; Corlaerstraat 6, NL-3863 ZD Nijkerk (NL).

(74) Agents: **BRANTS, Johan, Philippe, Emile** et al.; De Clercq, Brants & Partners, E. Gevaertdreef 10a, B-9830 Sint-Martens-Latem (BE).

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

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

Declaration under Rule 4.17:
— *of inventorship (Rule 4.17(iv)) for US only*

Published:
— *without international search report and to be republished upon receipt of that report*

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

(54) Title: OBJECT-ORIENTED DESIGN METHOD FOR THE TIME-EFFECTIVE AND COST-EFFECTIVE DEVELOPMENT OF PRODUCTION-GRADE EMBEDDED SYSTEMS BASED ON A STANDARDIZED SYSTEM ARCHITECTURE

(57) Abstract: The present invention relates to a system architecture for embedded systems based on hardware and software objects, in which the hardware objects are integrated with a hardware framework, and the software objects serve as software driver interface with the embedded application software for the hardware objects.

WO 2004/019239 A2

Object-oriented design method for the time-effective and cost-effective development of production-grade embedded systems based on a standardized system architecture

The invention relates to an object-oriented design method for the development of production-grade embedded systems (hardware and software) based on a platform-independent standardized system architecture. In this system architecture, hardware objects and software objects implement the system functionality through standardized hardware and software interfaces. The hardware objects are integrated with a prototype hardware framework or a production hardware framework, both of which are functionally identical.

The invention further relates to an apparatus which implements said prototyping hardware framework.

The invention further relates to a platform- independent architecture for driver software.

The invention further relates to a Graphical User Interface (GUI) utility which configures and integrates the hardware and software objects of said system architecture.

Field of the invention

The invention relates to an object-oriented design method for the rapid development of production-grade embedded systems (hardware and software) based on a platform-independent system architecture of hardware and software objects, class model for hardware objects, software driver architecture, hardware framework architecture, prototyping apparatus therefor, and a configuration and integration GUI utility with associated software object class hierarchy for the abstraction of the hardware objects.

In a first aspect, the present invention relates to a system architecture for embedded systems based on hardware and software objects, in which the hardware objects are integrated with a hardware framework, and the software objects serve as driver interface with the embedded application software. The preferred embodiment of this aspect of the invention is referred to as bf3Board architecture.

In a second aspect, the present invention relates to an object-oriented design method for composing embedded systems from hardware objects with either a prototype hardware framework or production hardware framework. This enables the rapid development of new

embedded systems. The preferred embodiment of this aspect of the invention is referred to as bf3DesignComposer method.

New is that the prototype hardware of the embedded system is immediately available for software development and that it is functionally identical to the production-grade counterpart, thus enabling a significant reduction of the hardware and software development time, cost, and risk. New is also that by employing the bf3DesignComposer design method, the development of new hardware objects which are compatible with the bf3Board architecture can take place asynchronously of the embedded system development.

A third aspect of the present invention relates to an apparatus which physically implements the prototype hardware framework. The preferred embodiment of this aspect of the present invention is referred to as bf3BaseBoard.

A fourth aspect of the present invention relates to a platform-independent architecture for driver software. The preferred embodiment of this aspect of the present invention is referred to as bf3Driver architecture.

An fifth aspect of the present invention relates to a Graphical User Interface (GUI) utility program which is used to configure the hardware objects through the software driver objects, and to integrate the software driver objects with the application software of the embedded system. A class hierarchy is used as abstraction of the hardware objects. The preferred embodiment of this aspect of the present invention is referred to as bf3BoardComposer software.

Background of the invention

Currently, the design methodology observed for the hardware and software development of embedded systems offers little options for the reuse of development results obtained in previous projects. More often than not, the embedded system hardware design is custom for the project. Because the hardware design process is very time-consuming, the software development typically starts with a so-called processor reference platform which is acquired from a third party, often the processor vendor. This reference platform serves as development target for the application software of the embedded system until the hardware designed for the project becomes available. In many cases, the architecture of the reference platform only slightly resembles the architecture of the embedded system under development. This development flow has six important disadvantages:

1. Much work needs to be done to adapt the software developed with the reference platform as target to the embedded system hardware, increasing the development time, cost and risk.

2. The hardware design of the embedded system makes little or no use of hardware developed in previous projects. Essentially, the value of the engineering efforts invested in earlier projects is thus reduced to zero. At the same time, the hardware design of the embedded system is of little or no value for reuse, making them virtually worthless for future projects.

3. The hardware design process offers little flexibility for quick additions or changes. Once the hardware is developed, additions or changes require a new iteration in the hardware development process.

4. The system design is commonly based on proprietary hardware and software architectures, allowing little or no possibilty to integrate hardware and software available from third parties in the embedded system under development.

5. The integration of hardware and software is mostly manual labor, and therefore extremely error-prone and time-consuming. Because this effort has to be done both for the reference platform and for the embedded system under development, the same effort is basically done twice. This again increases the development time, cost and risk.

6. The integration of hardware and software offers little or no option for automation with development tools.

All these items put a significant claim on the project schedule for the development of new embedded systems, along with the associated time, cost and risk to carry through the development of such systems.

To avoid these difficulties in the prior art, it would be desirable to provide a system architecture and method which together enhance the development of embedded systems by:

1. Using the same architecture for the reference platform (hardware prototype) and hardware product,

2. Providing a reference platform which can be used to instantly create a hardware prototype,

3. Enabling a seamless transition from hardware prototype to hardware product,

4. Enabling complete reuse of development results from earlier projects,
5. Allowing the integration of hardware and software from third parties,
6. Automating the integration of hardware and software with development tools.

The invention provides therefor a system architecture according to claims 1, 2, 3 and 4, and a development method according to claim 5.

Regarding the third aspect of the invention, a physical implementation of the hardware framework is required to enable the rapid construction of hardware prototypes which are functionally identical to the embedded system under development. The preferred embodiment of the invention is given in claim 6.

Regarding the fourth aspect of the invention, a uniform architecture for driver software is required to enable the automated integration of drivers with the embedded system application software with development tools. The definition of such an architecture significantly simplifies the development of driver software, and the integration of such software with the application software of the embedded system. The preferred embodiment of the invention is given in claim 7.

Regarding the fifth aspect of the invention, currently, Graphical User Interface (GUI) utility programs for the configuration and integration of hardware objects are difficult to develop due to the lack of a uniform architecture for driver software. Making use of the driver architecture of the present invention, the process of integrating the hardware and software of the emdedded system can be significantly shortened with the GUI utility program of the present invention. The preferred embodiment of the invention is given in claims 8 and 9.

US 5,870,588, Halambi *et al.* in "Automatic software toolkit generation embedded systems-on-chip". ICVC '99. 6th International Conference on Seoul, South Korea, 26-27 Oct, 1999, and Sutarwala *et al.* in "Flexible modeling environment for embedded systems design" Hardware/software co-design, 1994, Proceedings of the third international workshop on Grenoble, France 22-24 Sept. 1994, Los Alomos, CA, USA, IEEE Comput. Soc. all relate to hard/software co-design methodologies for System-on Chip (ASIC). US 6 2002 147 B1 and EP 1 056 001 describe design techniques for the development of driver software.

Summary of the invention

The present invention is a platform-independent system architecture, an object-oriented design method, integration software and apparatus for the rapid development of production-grade embedded systems (hardware and software).

The system architecture of the present invention is based on the definition of hardware objects and a corresponding class model. These hardware objects, each represent a hardware function which is physically implemented with electronic circuitry.

The hardware objects implement a framework interface corresponding to the class model to which they belong. The following hardware object classes are defined in the present invention:

1. Processor Core Module hardware object class,
2. Processor Core Extension Module hardware object class,
3. Memory Bus Expansion Module hardware object class,
4. Serial Bus Expansion Module hardware object class,
5. Debug Interface Expansion Module hardware object class,
6. Power Distribution Module hardware object class.

The hardware objects are integrated to a hardware design by means of a hardware framework. The hardware frameworks do not provide any functionality, other than the integration of hardware objects itself. Because of this, the functional behavior of the embedded system hardware is not dependent of the selected framework.

Two classes of hardware frameworks are defined in the present invention:

1. A prototype hardware framework class,
2. A production hardware framework class.

A hardware framework provides interfaces with the hardware object classes it is able to integrate; these interfaces are compatible with the class model for hardware objects of the present invention.

Internally, a hardware framework according to the the present invention consists mainly of a bus architecture which provides the interconnect structure between the hardware objects which together constitute the hardware design. This aspect is common to both hardware framework classes. In case of the prototype hardware framework, signal

buffering and electromechanical interconnection is added. The preferred embodiment of the bus architecture of the present invention is referred to as bf3Bus.

An essential feature of the present invention is that a hardware prototype of an embedded system can be created, which is functionally equivalent with the product-grade hardware. Also, because of the definition of the class model for hardware objects, two objects derived from the same class can be interchanged. For instance, a processor core object with a processor of type A can be exchanged for another processor core object with a processor of type B, without affecting the rest of the hardware design. Both features greatly facilitate the development of embedded systems.

The second aspect of the present invention is the object-oriented bf3DesignComposer design method for composing embedded systems (hardware and software) which enables the rapid development of new embedded systems by:

1. Selecting existing and defining new hardware objects which will together constitute the embedded system hardware,
2. Designing and developing said new hardware objects in accordance with the hardware object class definition.
3. Integrating the hardware objects with the prototyping hardware framework to build a hardware prototype,
4. Integrating the hardware objects with the production hardware framework to generate a production-grade hardware design,
5. Configuring and integrating the software objects associated with their respective hardware counterparts into the embedded application.

By following the bf3DesignComposer design methodology for the development of embedded system hardware, the reference platform of the prior art is replaced by a hardware prototype which is conceptually and functionally identical to the embedded system under development, thus avoiding the difficulties of the prior art. In addition, hardware objects can be developed asynchronously of the development of the embedded system, for instance by third parties from which they can be acquired. Furthermore, the bf3DesignComposer design methodology is beneficial for the reuse of hardware objects.

The third aspect of the present invention is an apparatus, referred to as bf3BaseBoard, which physically implements the prototype hardware framework class and the bf3Bus bus architecture. The bf3BaseBoard provides a limited number of electromechanical interfaces for each hardware object class. Thus, a hardware prototype of an embedded system can

be built from existing hardware objects. New hardware objects can be added to the hardware prototype as they become available.

A fourth aspect of the present invention is the bf3Driver platform-independent software architecture for driver software objects which provide the interface between the hardware objects and the application software of the embedded system.

The bf3Driver architecture for driver software is an essential feature of the present invention, since it enables formalizing the integration of the hardware and software of an embedded system. Without this formalization, automation of this integration process is sheer impossible.

The fifth aspect of the present invention is the bf3BoardComposer Graphical User Interface (GUI) utility program. Making use of the bf3Driver architecture of the present invention, it automates the process of integrating the hardware and software of the embedded system, thus avoiding the difficulties of the prior art. The bf3BoardComposer program is designed to operate under the Microsoft Windows family of operating systems.

Because the bf3BoardComposer Graphical User Interface (GUI) utility program does not interact directly with the hardware objects it configures, an abstraction model for the hardware objects is required. An object-oriented software class hierarchy serves this purpose. Technically, the software abstractions of the hardware objects are implemented as dynamic linked libraries (DLLs).

Brief description of the drawings

In the drawings, five sets of figures are given. The same reference numbers have been used for each of the separate sets A, B, C, D, and E. A, B, C, D, and E respectively are illustrations of embodiments of the first, second, third, fourth, and fifth aspect of the present invention.

Figure A.1 is an overview of the bf3Board system architecture for embedded systems, according to the present invention, including a hardware framework [HwFw1] which integrates five (5) hardware objects [HwObj1:5]. The hardware objects are addressed by the application software of the embedded system through five (5) corresponding software objects [SwObj1:5].

Figure A.2 shows the class definition of a Processor Core Module hardware object [PcHwObj1], according to the class model for hardware objects of the present invention.

Figure A.3 shows the class definition of a Processor Core Extension Module hardware object [PcExHwObj1], according to the class model for hardware objects of the present invention.

Figure A.4 shows the class definition of a Memory Bus Expansion Module hardware object [MbeHwObj1], according to the class model for hardware objects of the present invention.

Figure A.5 shows the class definition of a Serial Bus Expansion Module hardware object [SbeHwObj1], according to the class model for hardware objects of the present invention.

Figure A.6 shows the class definition of a Debug Expansion Module hardware object [DbgHwObj1], according to the class model for hardware objects of the present invention.

Figure A.7 shows the class definition of a Power Distribution Module hardware object [PdHwObj1], according to the class model for hardware objects of the present invention.

Figure A.8 shows the structure of the bf3Bus bus architecture according to the present invention, for the interconnection of a Processor Core Module hardware object [PcHwObj2], a group of Memory Bus Expansion Module hardware objects [MbeHwObjGr1], a Debug Interface Expansion Module hardware object [DbgHwObj2], a group of Serial Bus Expansion Module hardware objects [SbeHwObjGr1], and a Power Distribution hardware object [PdHwObj2].

Figure A.9 shows the role of the Processor Core Extension Module hardware object within the context of the bf3Bus architecture, according to the present invention, for the interconnection of a Processor Core Module hardware object [PcHwObj3], a Memory Bus Expansion Module hardware object [MbeHwObj2], a Debug Interface Expansion Module hardware object [DbgHwObj3], a Serial Bus Expansion Module hardware object [SbeHwObj2], and a Power Distribution hardware object [PdHwObj3].

Figure A.10 shows an overview of the production hardware framework [HwFw2], according to the present invention, which is able to integrate:

1. one (1) hardware object of the Processor Core Module class [PcHwObj4]
2. one (1) hardware object of the Processor Core Extenstion Module class [PcExHwObj3]
3. five (5) hardware objects of the Memory Bus Expansion Module class [MbeHwObj3:7]

4. four (4) hardware objects of the Serial Bus Expansion Module class [SbeHwObj3:6]
5. one (1) hardware object of the Debug Interface Expansion Module class [DbgHwObj4]
6. one (1) hardware object of the Power Distribution Module class [PdHwObj4]

by means of a bf3Bus interconnect structure [Bus1], according to the system architecture of the present invention.

Figure A.11 shows an overview of the prototype hardware framework [HwFw3], according to the present invention, which is able to integrate:

1. one (1) hardware object of the Processor Core Module class [PcHwObj5]
2. one (1) hardware object of the Processor Core Extenstion Module class [PcExHwObj4]
3. five (5) hardware objects of the Memory Bus Expansion Module class [MbeHwObj8:12]
4. four (4) hardware objects of the Serial Bus Expansion Module class [SbeHwObj7:10]
5. one (1) hardware object of the Debug Interface Expansion Module class [DbgHwObj5]
6. one (1) hardware object of the Power Distribution Module class [PdHwObj5]

by means of a bf3Bus interconnect structure [Bus2], according to the system architecture of the present invention, in combination with a corresponding number of connector pairs and signal buffering entities.

Figure B.1 is a flow diagram describing the prior art design flow for the hardware and software development of embedded systems.

Figure B.2 is a flow diagram describing the bf3DesignComposer design method for the hardware and software development of embedded systems, according to the present invention.

Figure C.1 shows an embodiment of the bf3BaseBoard, according to the present invention. It provides:

1. one (1) slot to accomodate a hardware object of the Processor Core Module class [PcHwObj6]; and

2. five (5) slots to accomodate hardware objects of the Memory Bus Expansion Module class [MbeHwObj13:17]; and
3. four (4) slots to accomodate hardware objects of the Serial Bus Expansion Module class [SbeHwObj11:14]; and
4. one (1) slot to accomodate a hardware object of the Debug Interface Expansion Module class [DbgHwObj6].

which are connected by means of a bf3Bus interconnect structure [Bus3], according to the system architecture of the present invention, in combination with a corresponding number of connector pairs and signal buffering entities.

The embodiment of the bf3BaseBoard as shown in figure C.1 incorporates a hardware object of the Power Distribution Module class [PdHwObj6] , and a hardware object of the Processor Core Extension Module class [PcExHwObj5].

Figure D.1 provides an abstraction model for the description of the bf3Driver architecture for driver software, according to the present invention, including a model for a Software Object [SwObj], a model for a Data Object [DataObj], a model for a Driver Table Entry Object [DrvObj], and a model for a Callback Function Object [CbObj].

Figure D.2 is a diagram of the bf3Driver architecture for driver software, according to the present invention, including three Hardware Objects [HwObj0:2], three associated Software Objects [SwObj0:2], three Data Objects [DataObj0:2], three Driver Table Entry Objects [DrvObj0:2], and three Callback Function Objects [Callback0:2];

Figure D.3 is the definition of the bf3Driver uniform driver interface in the C Programming Language, according to the present invention.

Figure E.1 shows an example of the bf3BoardComposer Graphical User Interface (GUI) utility program, according to the present invention.

Figure E.2 shows the class hierarchy for the abstraction of hardware objects by the bf3BoardComposer GUI utility program, according to the present invention.

Detailed description of the invention

Figure A.1 shows the general concept of the object-oriented bf3Board system architecture **101** of the present invention, consisting of five hardware objects **102**, **103**, **104**, **105**, and **106** which are integrated to a hardware design with hardware framework **107**. Five

software objects **108**, **109**, **110**, **111**, and **112** perform the function of driver software and provide the programming interface for the embedded application software **113**.

Figure A.2 provides the abstraction model **201** for hardware objects of the Processor Core Module class, according to the class model of the present invention. The interface of the Processor Core Module hardware object class is divided into signal groups which are each represented with a port symbol, as shown in legend **202**.

As the name indicates, an object of the Processor Core Module hardware object class, performs the core processing function within the bf3Board system architecture of the present invention. This implies that in general terms, the functionality of a hardware object of the Processor Core Module hardware object class will be implemented by one or more embedded processors, with a memory configuration. All other functionality provided by the hardware design is implemented with objects of the other hardware object classes. The interface ports of the Processor Core Module hardware object class are mainly defined for interaction with objects of the other hardware object classes, according to the class model of the present invention.

The Processor Core Module hardware object class definition includes the following interface ports which together constitute a memory bus interface:

1. The address bus signal group **203**
2. The data bus signal group **204**
3. The Unit-Select signal group **205**
4. The Read strobe signal **206**
5. The Write strobe signal **207**
6. The Output-Enable strobe signal **208**
7. The Address strobe signal **209**
8. The bus clock signal **210**
9. The bus arbitration signal group **211**
10. The Direct Memory Access (DMA) arbitration signal group **212**

The signals of the Unit-Select signal group **205** each activate an individual hardware object within the hardware framework. The signals of the bus arbitration signal group **211** are used to grant access to the memory bus interface by one of the hardware objects within the hardware framework. The signals of the Direct Memory Access (DMA)

arbitration signal group **212** are used allow a hardware object within the hardware framework to directly access memory within a hardware object of the Processor Core Module hardware object class.

An object of the Processor Core Module hardware object class provides the Interrupt Request signal group **213**. The signals of this signal group can be used by other hardware objects within the hardware framework to raise an interrupt request. An object of the Processor Core Module hardware object class implements the Interrupt Acknowledgement signal group **214** to acknowledge the interrupt to the hardware object which raised it.

An object of the Processor Core Module hardware object class implements the General-Purpose Control signal group **215** to drive control inputs of other hardware objects within the hardware framework.

Timer signals can be accepted and/or provided by an object of the Processor Core Module hardware object class through the Timer In signal group **217** and Timer Out signal group **216**, respectively.

Two types of reset signals exist within the bf3Board system architecture of the present invention: system reset and common reset. A system reset signal can only be generated by an object of the Processor Core Module hardware object class, for which it implements the System Reset interface port **219**. The common reset can be generated by any hardware object within the hardware framework. An object of the Processor Core Module hardware object class implements the Common Reset interface port **218** for this purpose.

An object of the Processor Core Module hardware object class has interface ports for the following serial communications interfaces:

1. Inter-Integrated Circuit (I^2C) serial bus **220**
2. Serial Peripheral Interface (SPI) **221**
3. One-wire serial bus **222**
4. Controller Area Network (CAN) serial bus **223**
5. Two Universal Asynchronous Receiver and Transmitter (UART) ports **224** and **225**
6. Six Serial Communications Channels (SCC) **226**, **227**, **228**, **229**, **230**, and **231**

To monitor the integrity of the system power supply, an object of the Processor Core Module hardware object class, provides the Power Good input signal **232**. The power supply itself is provided through the Power & Ground interface port **234**.

An object of the Processor Core Module hardware object class may implement an object-specific interface **233** to provide a processor-specific interface with compatible hardware objects within the hardware framework.

Figure A.3 provides the abstraction model **301** for hardware objects of the Processor Core Extension Module class, according to the class model of the present invention. The interface of the Processor Core Extension Module hardware object class is divided into signal groups which are each represented with a port symbol, as shown in legend **302**.

An object of the Processor Core Extension Module hardware object class can be integrated into a hardware framework for either of the following purposes:

1. To increase the number of General-Purpose Control signals available to an object of the Processor Core Module hardware object class.
2. To increase the number of Unit-Select signals available to an object of the Processor Core Module hardware object class.
3. To create a reset signal distribution by which each hardware object in the hardware framework can receive an individual reset signal, rather than the global System Reset.

An object of the Processor Core Extension Module hardware object class is connected to the memory bus interface of an object of the object of the Processor Core Module class through the following interface ports:

1. The address bus signal group **303**
2. The data bus signal group **304**
3. The Address strobe signal **305**
4. The Write strobe signal **306**
5. The Extension-Select signal **307**

The Extension-Select signal **307** is connected to a signal of the Unit Select signal group of an object of the Processor Core Module hardware object class.

An object of the Processor Core Extension Module hardware object class provides the following (output) signal groups to achieve the aforementioned functionality:

1. Unit-Select signal group **309**
2. General-Purpose Control signal group **310**
3. Unit-Reset signal group **311**

The signals of the Unit-Reset signal group **311** are used to individually activate the Unit Reset input signals of the hardware objects within the hardware framework. When the System Reset signal **308** is activated by an object of the Processor Core Module hardware object class, all signals of the Unit Reset signal group **309** are also activated.

The power supply for an object of the Processor Core Extension Module hardware object class is provided through the Power & Ground interface port **312**.

Figure A.4 provides the abstraction model **401** for hardware objects of the Memory Bus Expansion Module class, according to the class model of the present invention. The interface of the Memory Bus Expansion Module hardware object class is divided into signal groups which are each represented with a port symbol, as shown in legend **402**.

An object of the Memory Bus Expansion Module hardware object class extends the functionality of an object of the Processor Core Module hardware object class, for which it provides a number of interface ports.

The Memory Bus Expansion Module hardware object class definition includes the following interface ports which together constitute the interface with the memory bus interface of an object of the Processor Core Module hardware object class:

1. The address bus signal group **403**
2. The data bus signal group **404**
3. The Unit-Select signal **405**
4. The Read strobe signal **406**
5. The Write strobe signal **407**
6. The Output-Enable strobe signal **408**
7. The Address strobe signal **409**
8. The bus clock signal **410**
9. The bus arbitration signal group **411**
10. The Direct Memory Access (DMA) arbitration signal group **412**

The Unit-Select signal **405** is connected to a signal of the Unit-Select signal group of an object of the Processor Core Module hardware object class or of an object of the Processor Core Extension Module hardware object class.

An object of the Memory Bus Expansion Module hardware object class can raise an interrupt request at an object of the Processor Core Module hardware object class through

Interrupt Request signal **413**. An object of the Memory Bus Expansion Module hardware object class can detect the acknowledgement of an interrupt request through the Interrupt Acknowledgement signal group **414**.

An object of the Memory Bus Expansion Module hardware object class provides the General-Purpose Control input signal group **415** to provide direct access to 'on/off' functions within an object of the Memory Bus Expansion Module hardware object class . These functions are object-specific and must be published for each hardware object of the Memory Bus Expansion Module hardware object class which implements this interface port.

A timer signal can be accepted and/or provided by an object of the Memory Bus Expansion Module hardware object class through the Timer In signal **416** and Timer Out signal **417**, respectively.

An object of the Memory Bus Expansion Module hardware object class can be reset in either of two ways: (1) the System/Unit Reset signal **419** is activated, or (2) the Common Reset signal **418** is activated. An object of the Memory Bus Expansion Module hardware object class can also drive the Common Reset signal **418** to reset all hardware objects within the hardware framework. The System/Unit Reset signal **419** is either driven by the System Reset signal of an object of the Processor Core Module hardware object class, or by a Unit Reset signal of an object of the Processor Core Extension Module hardware object class.

An object of the Memory Bus Expansion Module hardware object class provides two interface ports for the following serial communications interfaces:

1. Inter-Integrated Circuit (I^2C) serial bus **420**
2. Serial Communications Channel **421**

The power supply for an object of the Memory Bus Expansion Module hardware object class is provided through the Power & Ground interface port **422**.

An object of the Memory Bus Expansion Module hardware object class may implement an object-specific interface **423** to be compatible with hardware objects of the Processor Core Module hardware object class which implement an object-specific interface.

Figure A.5 provides the abstraction model **501** for hardware objects of the Serial Bus Expansion Module class, according to the class model of the present invention. The interface of the Serial Bus Expansion Module hardware object class is divided into signal groups which are each represented with a port symbol, as shown in legend **502**.

An object of the Serial Bus Expansion Module hardware object class extends the functionality of an object of the Processor Core Module hardware object class, for which it provides a number of interface ports.

An object of the Serial Bus Expansion Module hardware object class has interface ports for the following serial communications interfaces:

1. Serial Communications Channel (SCC) **503**
2. Inter-Integrated Circuit (I^2C) serial bus **504**
3. Serial Peripheral Interface (SPI) **505**
4. One-wire serial bus **506**
5. Controller Area Network (CAN) serial bus **507**
6. Universal Asynchronous Receiver and Transmitter (UART) ports **508**

An object of the Serial Bus Expansion Module hardware object class can raise an interrupt request at an object of the Processor Core Module hardware object class through Interrupt Request signal **509**.

An object of the Serial Bus Expansion Module hardware object class provides the General-Purpose Control input signal group **510** to provide direct access to 'on/off' functions within an object of the Serial Bus Expansion Module hardware object class. These functions are object-specific and must be published for each hardware object of the Serial Bus Expansion Module hardware object class which implements this interface port.

An object of the Serial Bus Expansion Module hardware object class can be reset in either of two ways: (1) the System/Unit Reset signal **512** is activated, or (2) the Common Reset signal **511** is activated. An object of the Serial Bus Expansion Module hardware object class can also drive the Common Reset signal **511** to reset all hardware objects within the hardware framework. The System/Unit Reset signal **512** is either driven by the System Reset signal of an object of the Processor Core Module hardware object class, or by a Unit Reset signal of an object of the Processor Core Extension Module hardware object class.

The power supply for an object of the Serial Bus Expansion Module hardware object class is provided through the Power & Ground interface port **513**.

Figure A.6 provides the abstraction model **601** for hardware objects of the Debug Interface Expansion Module class, according to the class model of the present invention. The interface of the Debug Interface Expansion Module hardware object class is divided

into signal groups which are each represented with a port symbol, as shown in legend **602**.

An object of the Debug Interface Expansion Module hardware object class extends the functionality of an object of the Processor Core Module hardware object class by displaying status information and logging events, for which it provides a number of interface ports.

The Memory Bus Expansion Module hardware object class definition includes the following interface ports which together constitute the interface with the memory bus interface of an object of the Processor Core Module hardware object class:

1. The address bus signal group **603**
2. The data bus signal group **604**
3. The Unit-Select signal **605**
4. The Read strobe signal **606**
5. The Write strobe signal **607**
6. The Address strobe signal **608**
7. The bus clock signal **609**

The Unit-Select signal **605** is connected to a signal of the Unit-Select signal group of an object of the Processor Core Module hardware object class or of an object of the Processor Core Extension Module hardware object class.

An object of the Debug Interface Expansion Module hardware object class provides the General-Purpose Control input signal group **610** to provide direct access to 'on/off' functions within an object of the Debug Interface Expansion Module hardware object class. These functions are object-specific and must be published for each hardware object of the Debug Interface Expansion Module hardware object class which implements this interface port.

An object of the Debug Interface Expansion Module hardware object class can be reset in either of two ways: (1) the System/Unit Reset signal **612** is activated, or (2) the Common Reset signal **611** is activated. An object of the Debug Interface Expansion Module hardware object class can also drive the Common Reset signal **611** to reset all hardware objects within the hardware framework. The System/Unit Reset signal **612** is either driven by the System Reset signal of an object of the Processor Core Module hardware object

class, or by a Unit Reset signal of an object of the Processor Core Extension Module hardware object class.

The power supply for an object of the Debug Interface Expansion Module hardware object class is provided through the Power & Ground interface port **613**.

Figure A.7 provides the abstraction model **701** for hardware objects of the Power Distribution Module class, according to the class model of the present invention. The interface of the Power Distribution hardware object class is divided into signal groups which are each represented with a port symbol, as shown in legend **702**.

The Unit Power & Ground interface port **703** provides the power supply for the hardware objects within the hardware framework, which physically distributes the power. The integrity of the power supply is indicated by the Power Good signal **704**, which is monitored by an object of the Processor Core Module hardware object class. To control the operation of an object of the Power Distribution Module class, a Power Control interface port **705** is provided. The signals of the Power Control interface port **705** are driven with General-Purpose Control signals of an object of the Processor Core Module hardware object class, or of an object of the Processor Core Extension Module hardware object class.

Figure A.8 provides an overview of the bf3Bus bus architecture, according to the present invention. The bf3Bus bus architecture serves as interconnect structure for a collection of hardware objects. In figure A.8, the bf3Bus bus architecture connects a hardware object of the Processor Core Module class **801** with:

1. A group of hardware objects of the Memory Bus Expansion Module class **802**; and
2. A hardware object of the Debug Interface Expansion Module class **803**; and
3. A group of hardware objects of the Serial Bus Expansion Module class **804**; and
4. A hardware object of the Power Distribution Module class **805**.

The interconnect structure implemented by the bf3Bus architecture of the present invention connects the hardware objects of the class model for hardware objects of the present invention as described for figures A.2, A.3, A.4, A.5, A.6, and A.7. Not shown in figure A.8 are the Power & Ground interface ports of all hardware objects, nor the object-specific interface ports of the hardware objects of the Processor Core Module and Memory Bus Expansion Module classes.

Figure A.9 shows the function of a hardware object of the Processor Core Extension Module class **906**, according to the class model for hardware objects of the present invention, within the context of the bf3Bus bus architecture of the present invention.

The hardware object of the Processor Core Extension Module class **906** extends the functionality of a hardware object of the Processor Core Module class **901** by generating an additional:

1. General-Purpose Control output signal group **907**
2. Unit-Select output signal group **908**
3. Unit-Reset output signal group **909**

The (output) signals of these signal groups are used to drive the corresponding inputs of a hardware object of the:

1. Memory Bus Expansion Module class **902**
2. Debug Interface Expansion Module class **903**
3. Serial Bus Bus Expansion Module class **904**
4. Power Distribution Module class **905**

In case of the hardware object of the Serial Bus Expansion Module class **904**, the Unit-Select signal is used to drive the Slave-Select signal of the Serial Peripheral Interface (SPI) interface port.

The signals of the Unit-Reset signal group **909** are used to individually reset the hardware objects which are interconnected by the bf3Bus bus architecture of the present invention.

Figure A.10 provides an overview of a production hardware framework **1001**, according to the bf3Board system architecture of the present invention. This production hardware framework **1001** provides a number of so-called hardware object slots, which can accommodate a corresponding number of compatible hardware objects, according to the class model for hardware objects of the present invention.

The production hardware framework **1001** shown in figure A.10 can accommodate:

1. One (1) hardware object of the Processor Core Module class **1002**; and
2. One (1) hardware object of the Processor Core Extension Module class **1003**; and
3. Five (5) hardware objects of the Memory Bus Expansion Module class **1004**, **1005**, **1006**, **1007**, and **1008**; and

4. Four (4) hardware objects of the Serial Bus Expansion Module class **1009**, **1010**, **1011**, and **1012**; and
5. One (1) hardware object of the Debug Interface Expansion Module class **1013**; and
6. One (1) hardware object of the Power Distribution Module class **1014**.

The production hardware framework **1001** connects these hardware objects by means of an instance of the bf3Bus architecture **1015** of the present invention. Thus, the production hardware framework **1001** integrates the hardware objects to a production-grade hardware design for an embedded system, according to the bf3Board system architecture of the present invention.

Figure A.11 provides an overview of a prototype hardware framework **1101**, according to the bf3Board system architecture of the present invention. This prototype hardware framework **1101** provides a number of so-called hardware object slots, which can accommodate a corresponding number of compatible hardware objects, according to the class model for hardware objects of the present invention.

The prototype hardware framework **1101** shown in figure A.11 can accommodate:

1. One (1) hardware object of the Processor Core Module class **1102**; and
2. One (1) hardware object of the Processor Core Extension Module class **1103**; and
3. Five (5) hardware objects of the Memory Bus Expansion Module class **1104**, **1105**, **1106**, **1107**, and **1108**; and
4. Four (4) hardware objects of the Serial Bus Expansion Module class **1109**, **1110**, **1111**, and **1112**; and
5. One (1) hardware object of the Debug Interface Expansion Module class **1113**; and
6. One (1) hardware object of the Power Distribution Module class **1114**.

The prototype hardware framework **1101** connects these hardware objects by means of an instance of the bf3Bus architecture **1115** of the present invention.

The main purpose of the prototype hardware framework **1101** of the present invention, is to enable the construction of a prototype of the embedded system hardware at the beginning of the system development, which is functionally identical with an embedded system in which the hardware objects are integrated with a production hardware framework as described for figure A.10.

In practice, the hardware objects and the prototype hardware framework will be implemented as a set of Printed Circuit Boards (PCBs) with electronic circuitry, which implements the functionality of the hardware object or the prototype hardware framework.

The hardware objects of each class, according to the class model for hardware objects of the present invention, and the prototype hardware framework will have standard mechanical dimensions. To integrate the hardware object PCBs and the prototype hardware framework PCB, both must provide a part of a mating connector pair, with a standardized signal-to-pin assignment, and mechanical connector locations. For this reason, in the prototype hardware framework, a connector pair is added to each interface between a hardware object and the bf3Bus interconnect structure. The symbol used to depict a connector pair is contained in the symbol legend **1116**.

In case the standardized mechanical dimensions imply that the signal traces which physically implement the bf3Bus interconnect, are too long to maintain the electrical integrity of the signals carried by the bf3Bus interconnect structure, additional signal buffering will be required. For this reason, in the prototype hardware framework, a bi-directional signal buffer (transceiver) is added to each interface between a hardware object and the bf3Bus interconnect structure. The symbol used to depict a bi-directional signal buffer is contained in the symbol legend **1116**.

It is important to point out that, although the signal buffering entity and one part of the connector pair will be implemented on the PCB carrying the electronic circuitry which implements the functionality of a hardware object, these two elements are still considered to be part of the prototype hardware framework **1101**, according to the bf3Board architecture for embedded systems of the present invention.

Thus, the prototype hardware framework **1101** integrates the hardware objects to a hardware prototype of an embedded system which will be functionally identical with the production version of the same embedded system, which is acquired by integrating the hardware objects with the production framework, according to the bf3Board system architecture of the present invention.

Figure B.1 provides a flow diagram of the development of a new embedded system, as it is currently common practice (prior art), consisting of a number of discrete development steps.

After the Project Start **101**, the Product Requirements and System Architecture are determined in step **102**. After the Requirements and Architecture phase **102**, the product development flow of the embedded system is divided into two mainly independent paths:

1. One path starting with the definition **104** of the Hardware Architecture of the embedded system under development; and

2. Another path starting with the selection **103** of a suitable Reference Platform.

The reason for this approach is the lead time which is inherently associated with the development of the embedded system hardware. Because of the nature of embedded systems, where the embedded system is designed to perform a project-specific set of functions, most projects require a unique hardware and software architecture.

To partially overcome this deficiency, in the prior art, commercially available hardware or the hardware of a previously developed embedded system is initially used for the development of the embedded software, until the Hardware Design phase **105** and Hardware Test phase **114** have been completed.

In many cases, the reference platform and embedded system under development will have disimilar hardware architectures. Because a software architecture is strongly dependent of the architecture of the underlying embedded system hardware, this implies that two distinct software architecture definition steps are required:

1. A Software Architecture definition **106**, based on the hardware architecture of the selected reference platform; and

2. A Software Architecture definition **108**, based on the hardware architecture of the embedded system under development.

As a result, two distinct Software Design steps **107** and **109** are also required. Within the area marked by the dashed line **110**, the design engineers involved in the development of the embedded system will be forced to develop software which is only suitable for either of both target platforms. This creates additional overhead in the development schedule.

When the Software Design phase **107** for the reference platform has been completed, the developed software will be integrated with reference platform hardware in step **111**. Upon completion of this phase, the Integration Testing phase **112** commences. This phase will be terminated (step **113**) when the Hardware Test phase **114** is completed, and the hardware for the embedded system under development becomes available.

Now, a new integration phase (step **115**) is started, in which the software developed specifically for the embedded system under development will be integrated with the hardware which has recently become available. In essence, the work done in step **111** for the reference platform is repeated for the embedded system hardware in step **115**. The

same applies for the Integration Testing phase **116**, which repeats the work of phase **112**. The Project End **117** is reached, when the Integration Testing phase **116** is completed.

Figure B.2 shows the development flow of the bf3DesignComposer method, according to the present invention.

As can be concluded from the description of the development flow shown in figure B.1, the prior art process for the development of new embedded systems has the following deficiencies:

1. The hardware of the new embedded system is specifically developed to perform the required functions; it makes little use of previous development result, and at the same time offers little value for reuse.
2. The software development process is complex; it requires that the same work is done twice.

The bf3DesignComposer method, based on the bf3Board system architecture, according to the present invention, to a large extent enables the elimination of these deficiencies.

In the bf3DesignComposer design method of the present invention, after the Project Start **201** and the System Requirements and Architecture phase **202**, the hardware architecture of the embedded system, is based on the bf3Board system architecture of the present invention. In phase **203**, the hardware objects and the hardware framework which together constitute the hardware of the embedded system are selected.

The bf3DesignComposer design method of the present invention enables the effective reuse hardware objects developed in previous projects. Alternatively, hardware objects which are commercially acquired from third parties, can be integrated into the hardware design. Depending on the result of the decision **211** if new hardware and corresponding software objects are required for the embedded system under development, the new hardware and software objects are developed and tested in phases **209** and **210**, respectively. When all hardware objects required for the embedded system are available, they can be integrated with the production hardware framework (phase **212**). Thus, the hardware development process is reduced to the development of genuinely new functionality.

Meanwhile, the software development starts by building a prototype of the embedded system by integrating existing and commercially acquired hardware objects with a prototype hardware framework (phase **204**), as described for figure A.11.

Since the hardware architectures of both the prototype embedded system and its production version are identical in the bf3DesignComposer design method of the present invention, only software which is actually required for the product needs to be developed.

Thus, the software development process can be simplified to a single flow consisting of:

1. Definition of a Software Architecture **205**
2. Software Design phase **206**
3. Hardware/Software Integration phase **207**
4. Integration Testing phase **208**

The bf3DesignComposer design method reduces the software development process to the development of functionality which is actually required for the production embedded system.

Once the production version of the embedded system becomes available with the completion of Hardware Integration phase **212**, the new hardware and software objects which were developed are integrated with the software in step **213**, after which Integration Testng **214** for these new hardware and software objects takes place. When the Integration Testing phase **214** has been completed, the Project End is **215** is reached.

Figure C.1 shows an embodiment **101** of the bf3BaseBoard, according to the present invention. The bf3BaseBoard physically implements the prototype hardware framework of the present invention, as described for figure A.11. As such, it provides slots to accommodate:

1. One (1) hardware object of the Processor Core Module class **102**; and
2. Five (5) hardware objects of the Memory Bus Expansion Module class **104**, **105**, **106**, **107**, and **108**; and
3. Four (4) hardware objects of the Serial Bus Expansion Module class **109**, **110**, **111**, and **112**; and
4. One (1) hardware object of the Debug Interface Expansion Module class **113**.

In addition, the embodiment of the bf3BaseBoard as shown in figure C.1 incorporates a hardware object of the Power Distribution Module class **114** , and a hardware object of the Processor Core Extension Module class **103**.

The bf3BaseBoard physically consists of a printed circuit board (PCB) which carries the electronic circuitry which implements the functionality of the bf3Bus interconnect structure

115, the hardware object of the Power Distribution Module class **114**, and the hardware object of the Processor Core Extension Module class **103**.

The bf3BaseBoard implement a connector sockets for each implementation of a hardware object. The mechanical location of the connectors on the bf3BaseBoard PCB is defined for each hardware object class, according to the class model for hardware objects of the present invention. The pin assignment of the connector configuration for each slot is also defined.

Hardware objects which the bf3BaseBoard is able to integrate must implement a matching connector configuration, as well as bi-directional signal buffering, to account for the length of the signal traces of the bf3Bus interconnect structure **115** on the bf3BaseBoard PCB.

For reasons of brevity, the bi-directional signal buffering and both parts of the connector pairs have been depicted in the bf3BaseBoard diagram of figure C.1. The symbols used to depict a bi-directional signal buffer and a connector pair is contained in the symbol legend **116**.

The bf3BaseBoard enables the immediate creation of embedded system prototypes with hardware objects which have been developed to be compatible with the mechanical and electrical properties of the bf3BaseBoard.

Figure D.1 provides an abstraction model for the description of the bf3Driver architecture for driver software, according to the present invention. The following object are defined as part of the bf3Driver architecture of the present invention:

1. Software Object **101**
2. Data Object **102**
3. Driver Table Entry Object **103**
4. Callback Function Object **104**

Not shown in figure C.1 is a model of the hardware object, according to the bf3Board System Architectrue of the present invention; abstraction models for such hardware objects were shown in figures A.2, A.3, A.4, A.5, A.6, and A.7.

The Software Object **101** of the bf3Board System Architecture of the present invention provides a number of interface for the integration of the other objects of the bf3Driver architecture of the present invention. The Object Data interface **105** is used to access Data Objects **102** in the memory of the embedded system. The Base Address interface **106** is used by the application software of the embedded to convey the base address of

an array of Data Objects **102** in the memory of the embedded system to a Software Object **101**. The Address interface 107 is used by a Software Object **107** to access a Data Object **102** in the memory of the embedded system. The Fxns interface **108** is used by the application software of the embedded application to access the internal function table of the Software Object **101**. The format of said function table will described with figure D.3. The Application Callback interface **109** is used by a Software Object **101** to notify the application software of the embedded system of events which require the attention of said application software through a Callback Function Object **104**. The Interrupt Request interface **110** is used by a hardware object of the bf3Board System Architecture of the present invention to raise an interrupt at the Software Object **101**. In practice, an interrupt request raised by a hardware object of the present invention will cause an Interrupt Service Routine (ISR) of the Software Object **101** to be invoked by the processor of the embedded system. This procedure is well-known to a professional skilled in the art.

The Data Object **102** provides a Data interface **111**, and an Address interface **112**. Both interfaces are used by a Software Object **101** to access data stored by the Data Object **102**.

The Driver Table Entry Object **103** links the application software of the embedded system to an instance of a Software Object **101**. It provides a single interface: the Fxns interface **113** is used by the application software which addresses the Driver Table Entry Object **103** to gain access to the internal function table of the Software Object **101**.

The Callback Function Object **104**, which is implemented by the application software of the embedded system, is accessed by a Software Object **101** to notify said application software of an event which requires the attention of said application software. It provides the Callback interface 114 as entry point for the Software Object **101**.

Figure D.2 is a diagram of the bf3Driver architecture for driver software, according to the present invention.

Figure D.2 shows three hardware objects **216**, **217**, and **218**, together with three associated software objects **213**, **214**, and **215**, respectively, according to the bf3Board System Architecture of the present invention. In figure D.2, hardware objects **216** and **217** are identical (indicated by the same fill pattern), **218** implements functionality which differs from hardware objects **217** and **218**.

Each software object is identified by a Device Identifier. Software objects are grouped by the functionality of their corresponding hardware objects. Each software object in such a group is assigned a Device Identfier which is unique within a group of software objects. In

case of software objects **214** and **213**, the Device Identifiers are set to '0' and '1', respectively. Within a group, the assignment of Device Identifiers must start at '0'. This rule also applies for groups of software objects which contain only a single software object.

The software objects **213**, **214**, and **215** shown in figure D.2 each have a data object associated with them (**201**, **203**, and **205**, respectively). The data objects corresponding with a group of software objects are organized as an object array in the memory of the embedded system. The Device Identifiers of the software objects in a group of software objects serve as index of a data object in an array of said data objects. In figure data object array **202** contains data objects **201** and **203**, which are associated with software objects **213** and **214**, respectively. Data object array **204** contains a single data object **205**, associated with software object **215**.

A driver table entry object acts as entry point to a software object for the application software **219** of the embedded system. Therefore, for each software object in the embedded system a driver table entry object must exist. The driver table entry objects are organized as an array to create a driver table **209** in the memory of the embedded system. The entries **210**, **211**, and **212** of the driver table **209** provide access to software objects **213**, **214**, and **215**, respectively. The index of a driver table entry object in the driver table **209** can be seen as a unique Driver Identifier of a software object within the embedded system. The value of this Driver Identifier is set for software object **213**, **214**, and **215** is set to '0', '1', and '2', respectively.

Three callback function objects **208**, **207**, and **206** enable the software objects **213**, **214** and **215**, respectively to notify the application software of the embedded system **219** of relevant events related to the operation of their corresponding hardware objects.

Figure D.3 is the definition of the bf3Driver uniform driver interface in the C Programming Language, according to the present invention, along with the required type definitions. The format of the IBF3DRIVER_Fxns function table is immediately understood by a professional skilled in the art.

Figure E.1 shows an example of the bf3BoardComposer Graphical User Interface (GUI) utility program **101**, according to the present invention. The bf3BoardComposer user interface **101** contains a menu **102** for the selection of the framework which is used to integrate the hardware objects, according to the bf3Board System Architecture of the present invention. The available hardware objects are listed in tree view **103**, grouped by

their hardware object class, according to the class model for hardware objects of the present invention.

The user interface **101** also contains a section **104** in which the hardware objects can be assigned to a slot within the hardware framework selected by menu **102**.

Figure E.2 shows the class hierarchy for the abstraction of hardware objects by the bf3BoardComposer GUI utility program, according to the present invention.

All objects within the class hierarchy for the abstraction of hardware objects of the present invention are derived from the CBf3BoardObject root class **201**. For each hardware object class of the class model for hardware objects of the present invention, a subclass is derived from the CBf3BoardObject root class **201**:

1. The CBf3HwFramework class **202** represents an implementation of a hardware framework, according to the bf3Board System Architecture of the present invention.
2. The CBf3ProcessorCore class **203** serves as base class for hardware objects of the Processor Core Module hardware object class, according to the class model for hardware objects of the present invention.
3. The CBf3ProcessorCoreExt class **204** serves as base class for hardware objects of the Processor Core Extension Module hardware object class, according to the class model for hardware objects of the present invention.
4. The CBf3MemBusExpansion class **205** serves as base class for hardware objects of the memory Bus Expansion Module hardware object class, according to the class model for hardware objects of the present invention.
5. The CBf3SerialBusExpansion class **206** serves as base class for hardware objects of the Memory Bus Expansion Module hardware object class, according to the class model for hardware objects of the present invention.
6. The CBf3DebugInterface class **207** serves as base class for hardware objects of the Debug Interface Expansion Module hardware object class, according to the class model for hardware objects of the present invention.
7. The CBf3PowerDistribution class **208** serves as base class for hardware objects of the Power Distribution Module hardware object class, according to the class model for hardware objects of the present invention.

For each hardware object which is developed, a new class must be derived from the corresponding base class within the class hierarchy for the abstraction of hardware objects, according to the present invention.

Claims

1. A system architecture for embedded systems based on hardware and software objects, in which the hardware objects are integrated with a hardware framework, and the software objects serve as software driver interface with the embedded application software for the hardware objects.

2. A class model for hardware objects according to the previous claim 1 which categorizes hardware objects by their functional behavior and defines their system interface by one of the following classes:

    a. Processor Core Module hardware object class; or

    b. Processor Core Extension Module hardware object class; or

    c. Memory Bus Expansion Module hardware class; or

    d. Serial Bus Expansion Module hardware object class; or

    e. Debug Interface Expansion Module hardware object class; or

    f. Power and Reset Distribution Module hardware object class.

3. A bus architecture which serves as interconnect structure for the hardware objects derived from the class model according to previous claim 2.

4. A class model for hardware frameworks according to the previous claim 1 which categorizes hardware frameworks by their primary application:

    a. Prototype Hardware Framework class; or

    b. Production Hardware Framework class.

5. An object-oriented design method for the development of embedded systems based on the architecture according to the previous claim 1, comprising the steps of:

    a. Selecting existing and defining new hardware objects which together constitute the embedded system hardware,

    b. Designing and developing said new hardware objects in accordance with the hardware object class definition.

    c. Integrating the hardware objects with the prototyping hardware framework to build a hardware prototype,

    d. Integrating the hardware objects with the production hardware framework to generate a production-grade hardware design,

e. Configuring and integrating the software objects associated with their respective hardware counterparts into the embedded application.

6. An apparatus which physically implements the prototype hardware framework of previous claim 4a.

7. A platform-independent architecture for driver software.

8. A Graphical User Interface (GUI) which is able to perform step e of the previous claim 5.

9. A software class hierarchy which serves as abstraction model for the hardware objects for use by the Graphical User Interface utility according to the previous claim 8.

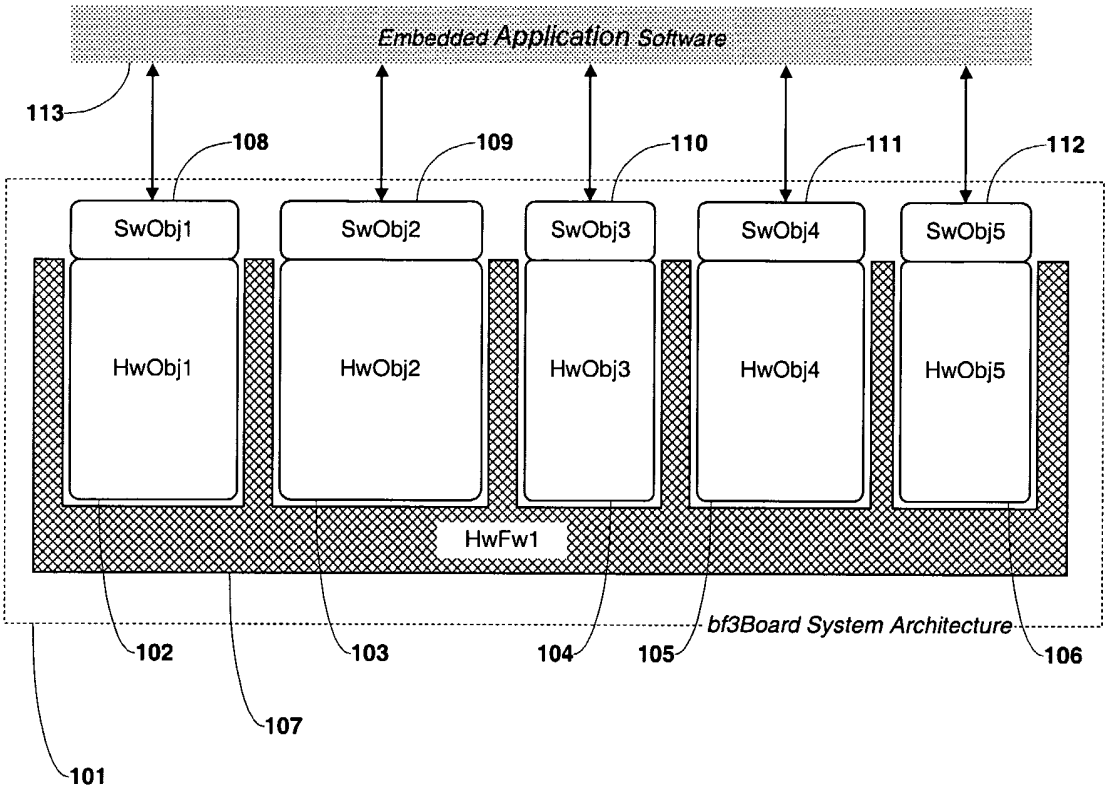
Embedded Application Software
113
108
109
110
111
112
SwObj1
SwObj2
SwObj3
SwObj4
SwObj5
HwObj1
HwObj2
HwObj3
HwObj4
HwObj5
HwFw1
bf3Board System Architecture
102
103
104
105
106
107
101

Figure A.1

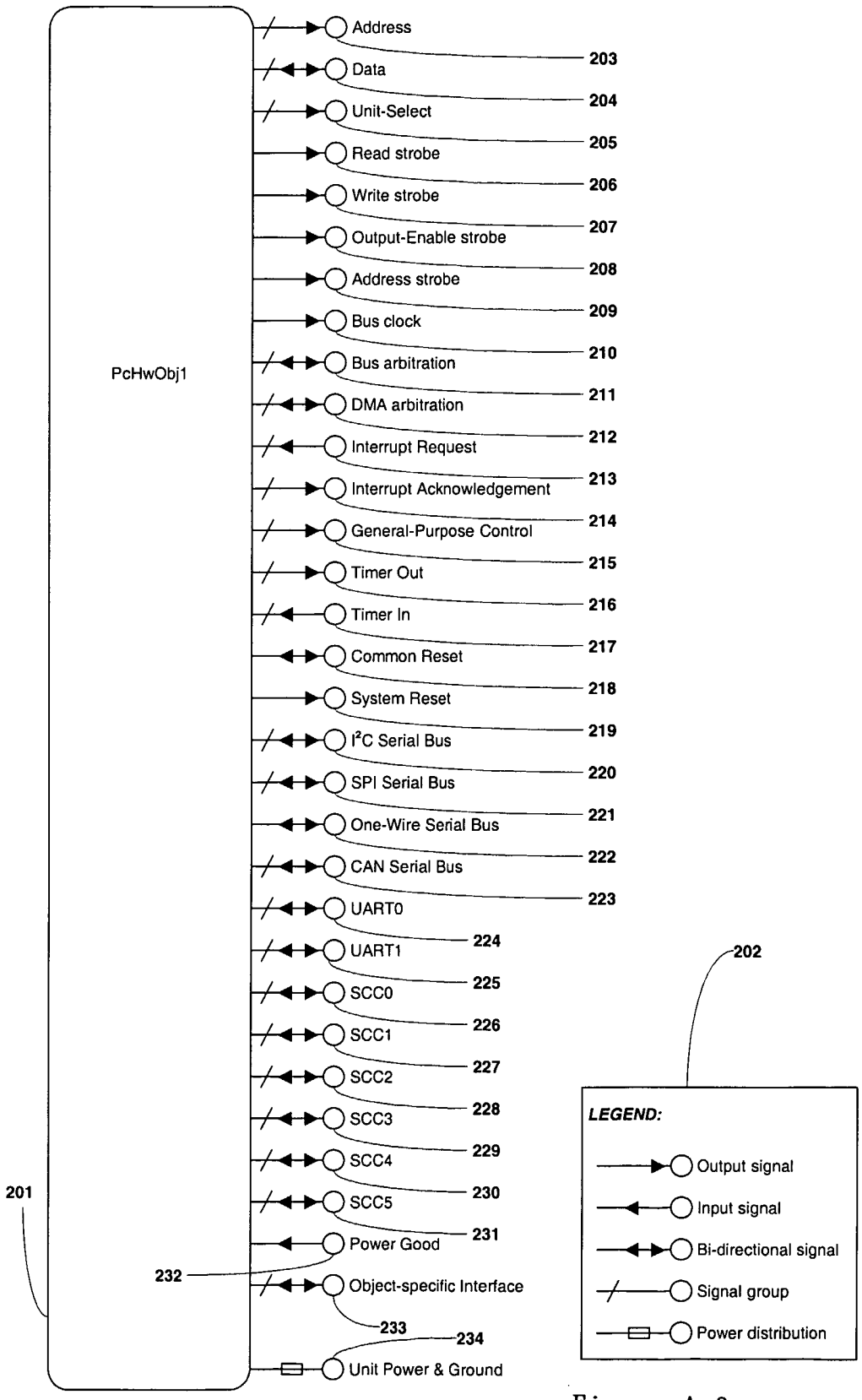
PcHwObj1
Address
Data
Unit-Select
Read strobe
Write strobe
Output-Enable strobe
Address strobe
Bus clock
Bus arbitration
DMA arbitration
Interrupt Request
Interrupt Acknowledgement
General-Purpose Control
Timer Out
Timer In
Common Reset
System Reset
I²C Serial Bus
SPI Serial Bus
One-Wire Serial Bus
CAN Serial Bus
UART0
UART1
SCC0
SCC1
SCC2
SCC3
SCC4
SCC5
Power Good
Object-specific Interface
Unit Power & Ground
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
LEGEND:
Output signal
Input signal
Bi-directional signal
Signal group
Power distribution

Figure A.2

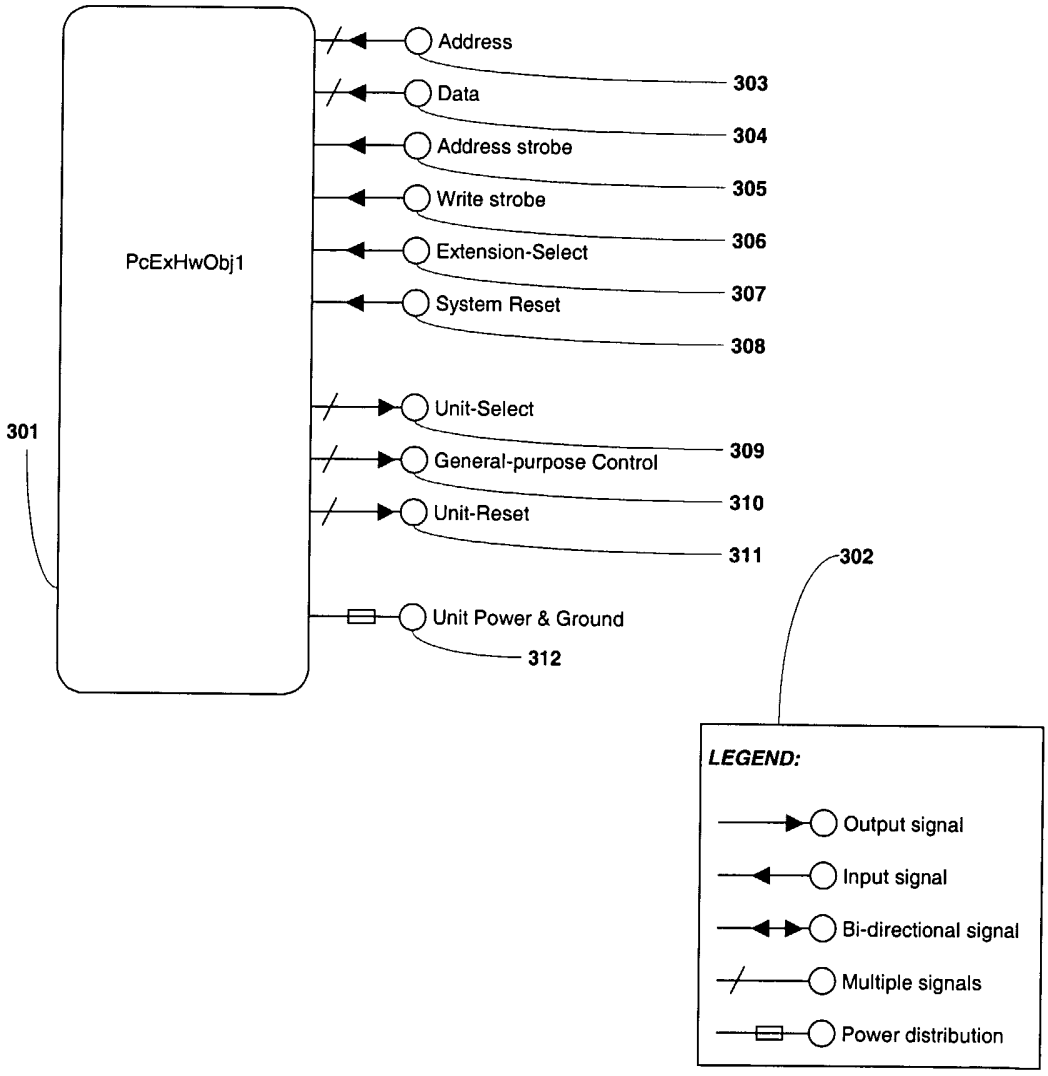
PcExHwObj1
301
Address
303
Data
304
Address strobe
305
Write strobe
306
Extension-Select
307
System Reset
308
Unit-Select
309
General-purpose Control
310
Unit-Reset
311
Unit Power & Ground
312
302
LEGEND:
Output signal
Input signal
Bi-directional signal
Multiple signals
Power distribution

Figure A.3

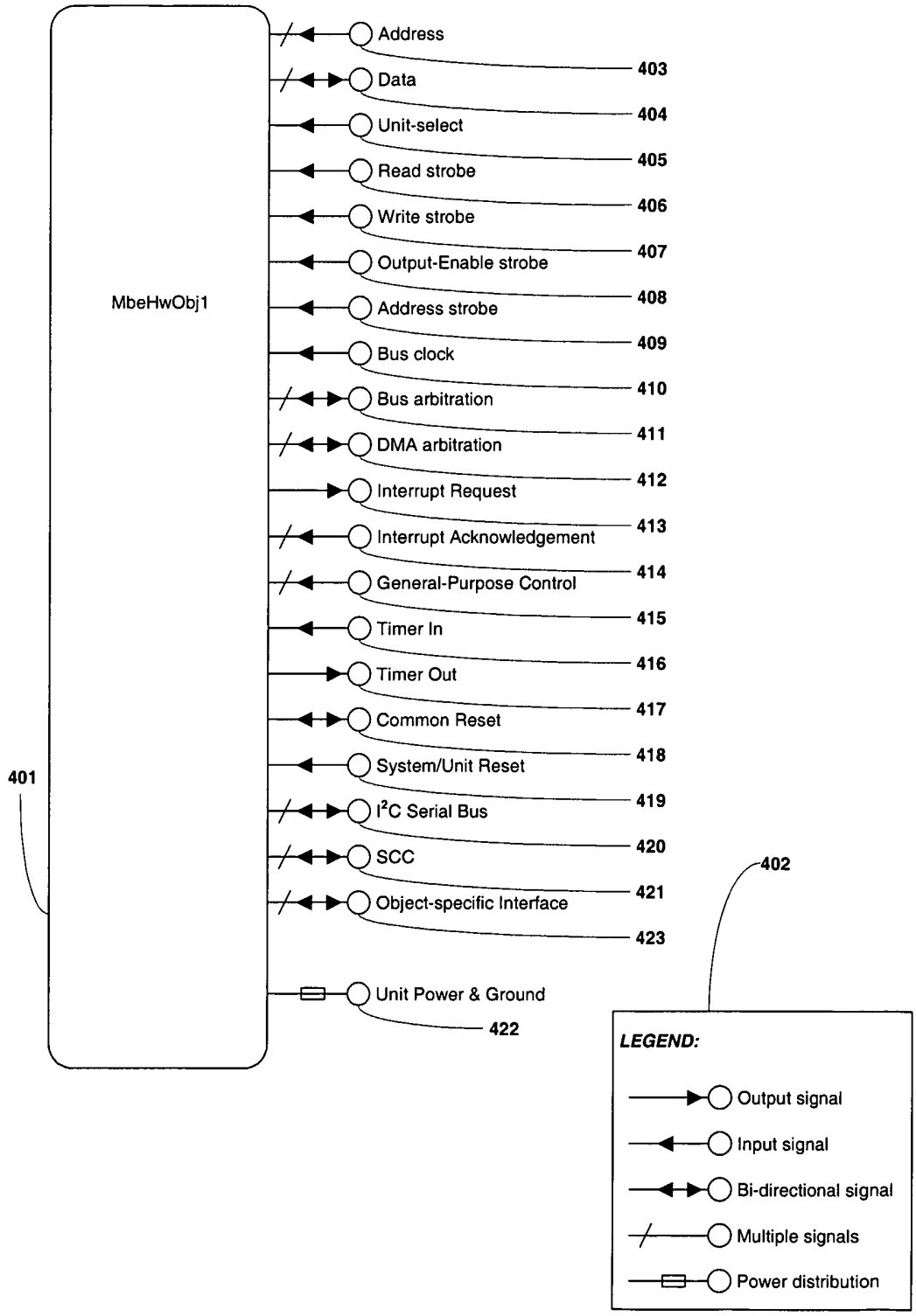
MbeHwObj1
401
Address
403
Data
404
Unit-select
405
Read strobe
406
Write strobe
407
Output-Enable strobe
408
Address strobe
409
Bus clock
410
Bus arbitration
411
DMA arbitration
412
Interrupt Request
413
Interrupt Acknowledgement
414
General-Purpose Control
415
Timer In
416
Timer Out
417
Common Reset
418
System/Unit Reset
419
I²C Serial Bus
420
SCC
421
Object-specific Interface
423
Unit Power & Ground
422
402
LEGEND:
Output signal
Input signal
Bi-directional signal
Multiple signals
Power distribution

Figure A.4

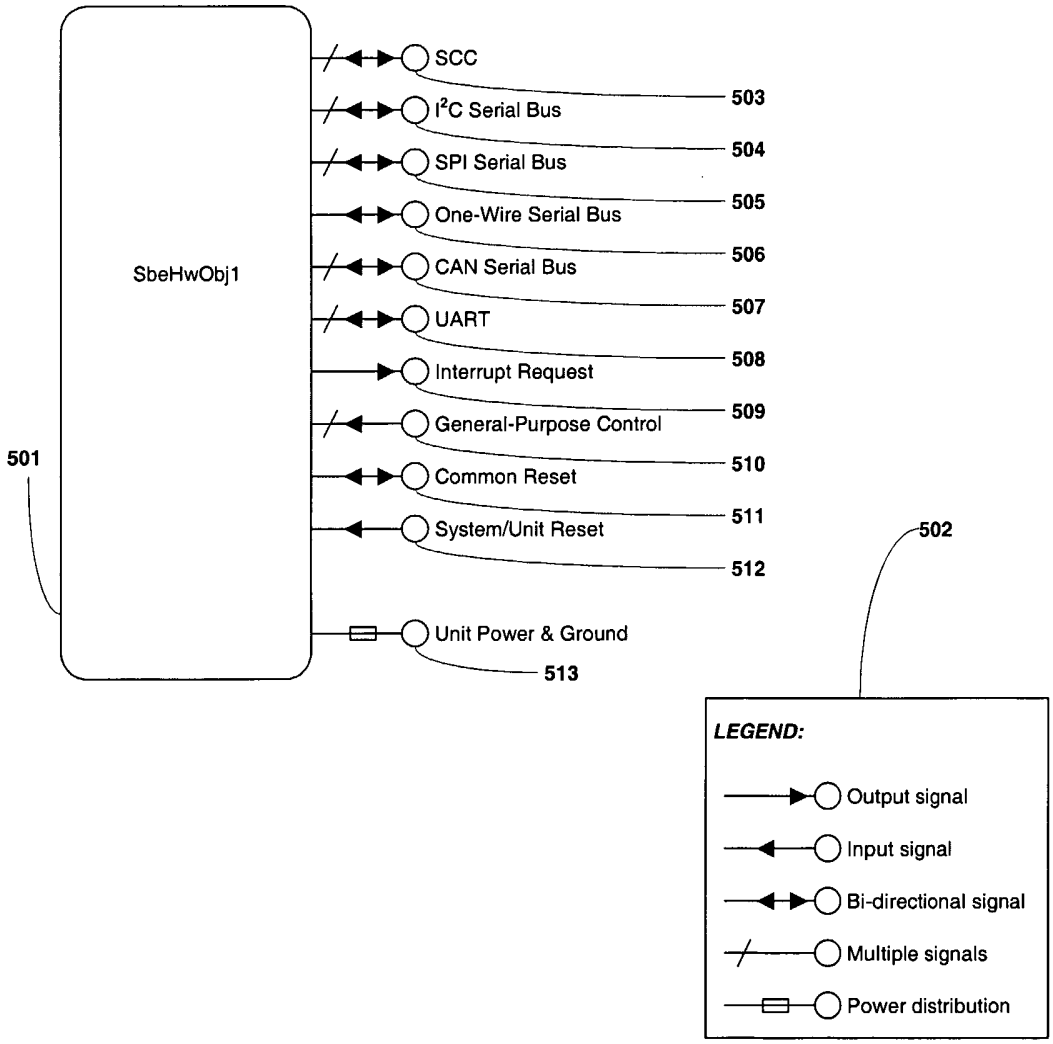
SbeHwObj1
501
SCC
I²C Serial Bus
SPI Serial Bus
One-Wire Serial Bus
CAN Serial Bus
UART
Interrupt Request
General-Purpose Control
Common Reset
System/Unit Reset
Unit Power & Ground
503
504
505
506
507
508
509
510
511
512
513
502
LEGEND:
Output signal
Input signal
Bi-directional signal
Multiple signals
Power distribution

Figure A.5

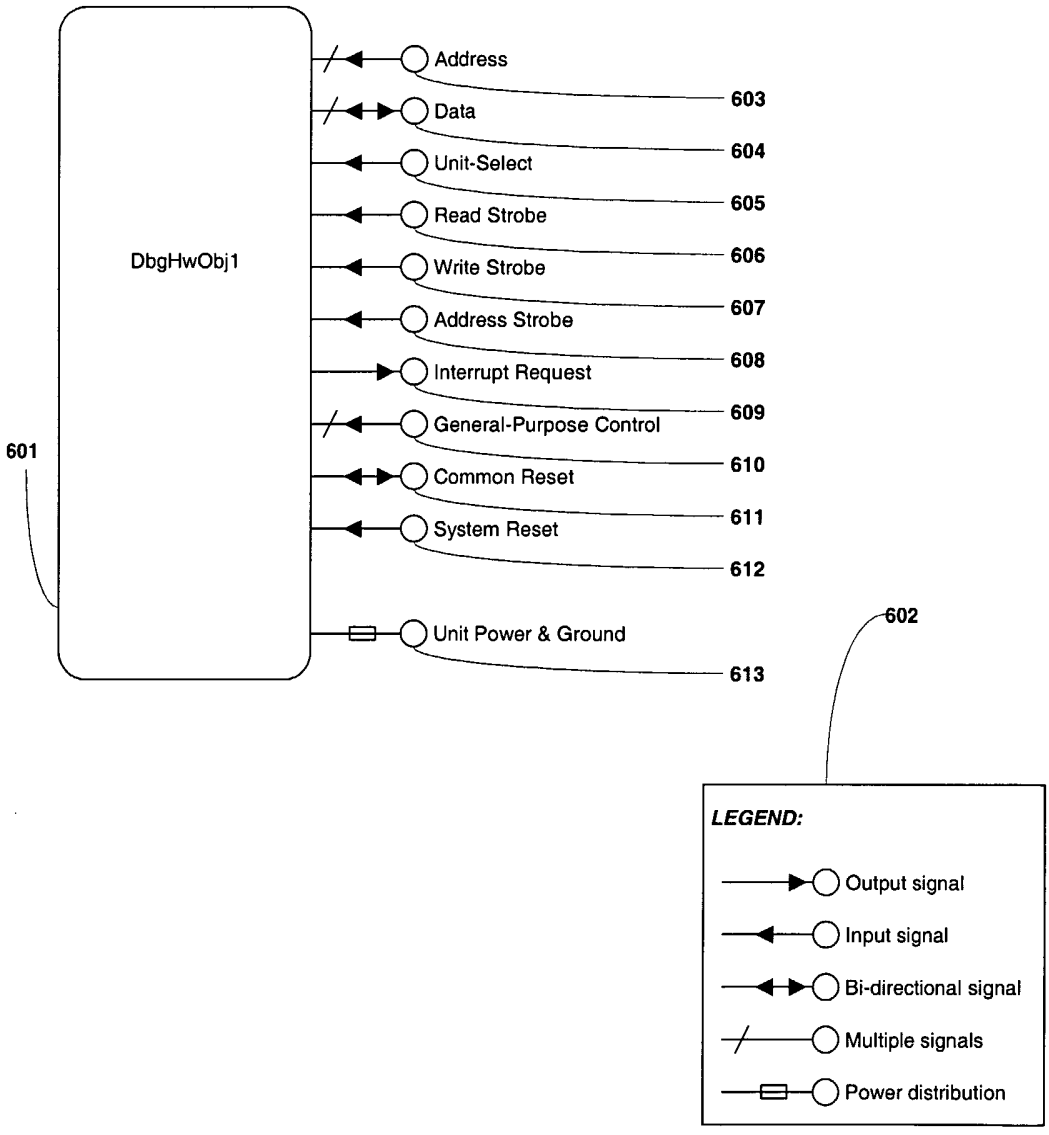
DbgHwObj1
601
Address
603
Data
604
Unit-Select
605
Read Strobe
606
Write Strobe
607
Address Strobe
608
Interrupt Request
609
General-Purpose Control
610
Common Reset
611
System Reset
612
Unit Power & Ground
613
602
LEGEND:
Output signal
Input signal
Bi-directional signal
Multiple signals
Power distribution

Figure A.6

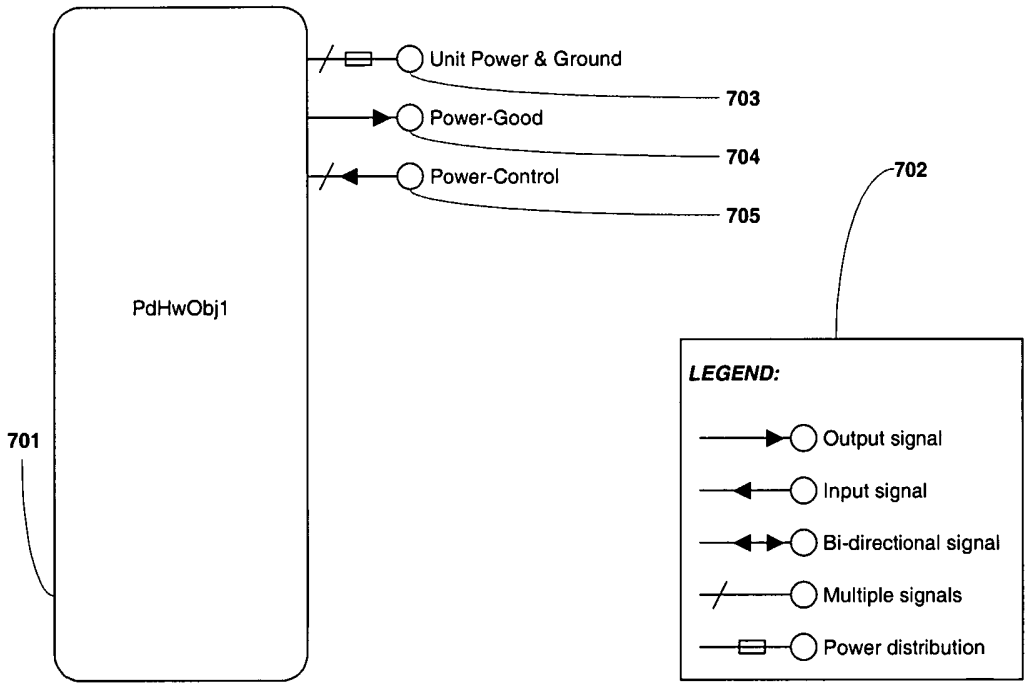
Unit Power & Ground
703
Power-Good
704
Power-Control
705
702
PdHwObj1
701
LEGEND:
Output signal
Input signal
Bi-directional signal
Multiple signals
Power distribution

Figure A.7

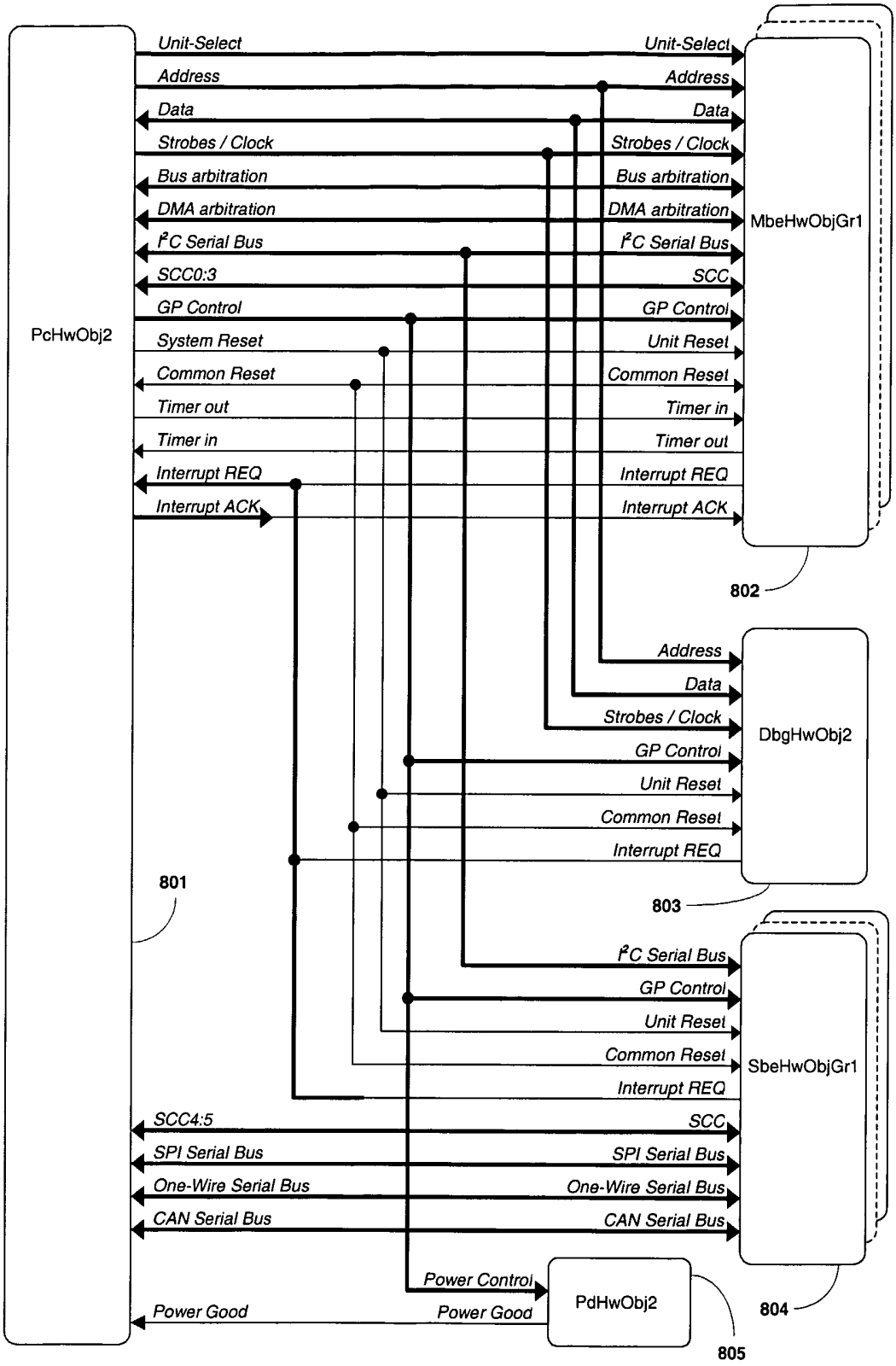
PcHwObj2
Unit-Select
Address
Data
Strobes / Clock
Bus arbitration
DMA arbitration
I²C Serial Bus
SCC0:3
GP Control
System Reset
Common Reset
Timer out
Timer in
Interrupt REQ
Interrupt ACK
MbeHwObjGr1
802
DbgHwObj2
803
801
SbeHwObjGr1
804
SCC4:5
SPI Serial Bus
One-Wire Serial Bus
CAN Serial Bus
Power Control
Power Good
PdHwObj2
805

Figure A.8

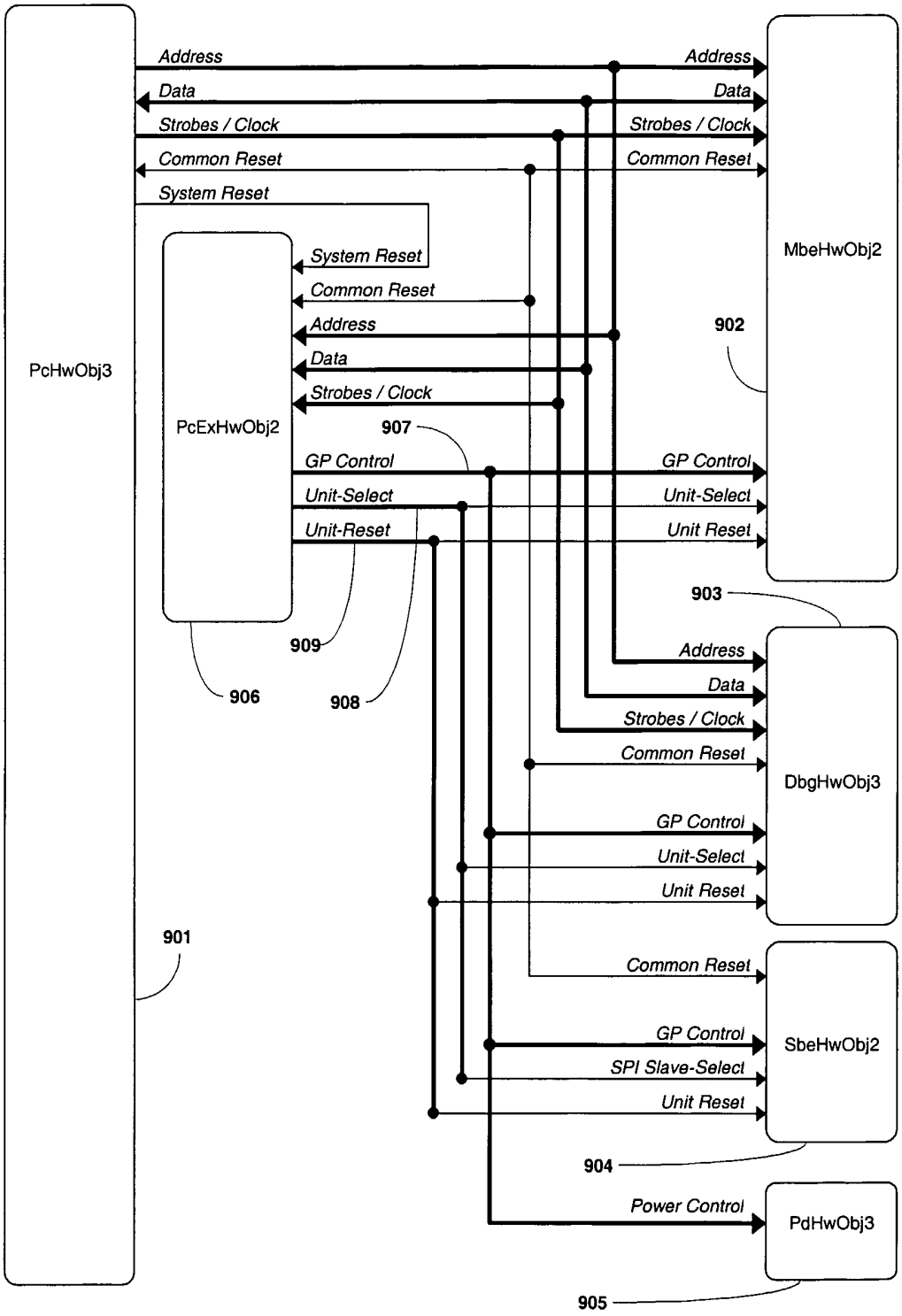
PcHwObj3
PcExHwObj2
MbeHwObj2
DbgHwObj3
SbeHwObj2
PdHwObj3
Address
Data
Strobes / Clock
Common Reset
System Reset
GP Control
Unit-Select
Unit-Reset
Unit Reset
SPI Slave-Select
Power Control
901
902
903
904
905
906
907
908
909

Figure A.9

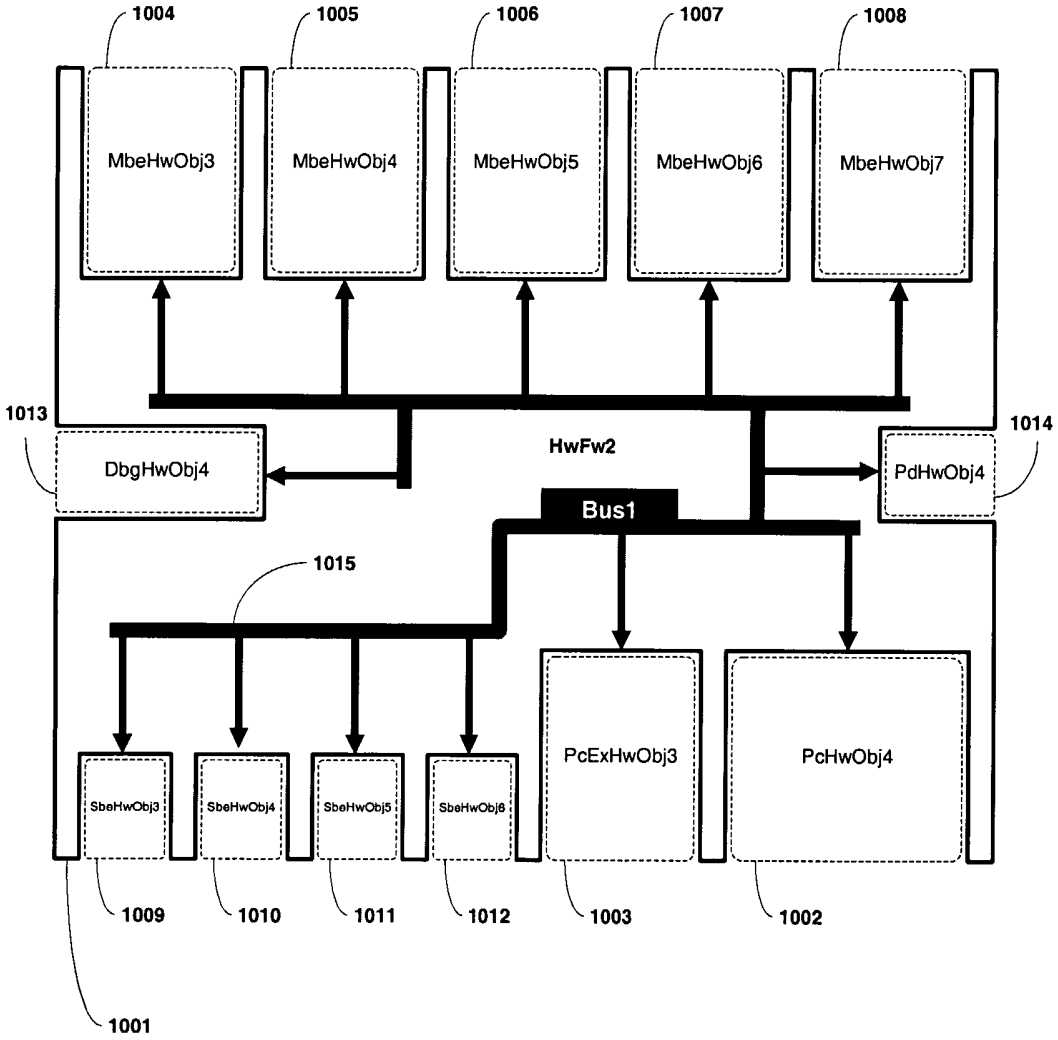
1004
1005
1006
1007
1008
MbeHwObj3
MbeHwObj4
MbeHwObj5
MbeHwObj6
MbeHwObj7
1013
DbgHwObj4
HwFw2
PdHwObj4
1014
Bus1
1015
SbeHwObj3
SbeHwObj4
SbeHwObj5
SbeHwObj6
PcExHwObj3
PcHwObj4
1009
1010
1011
1012
1003
1002
1001

Figure A.10

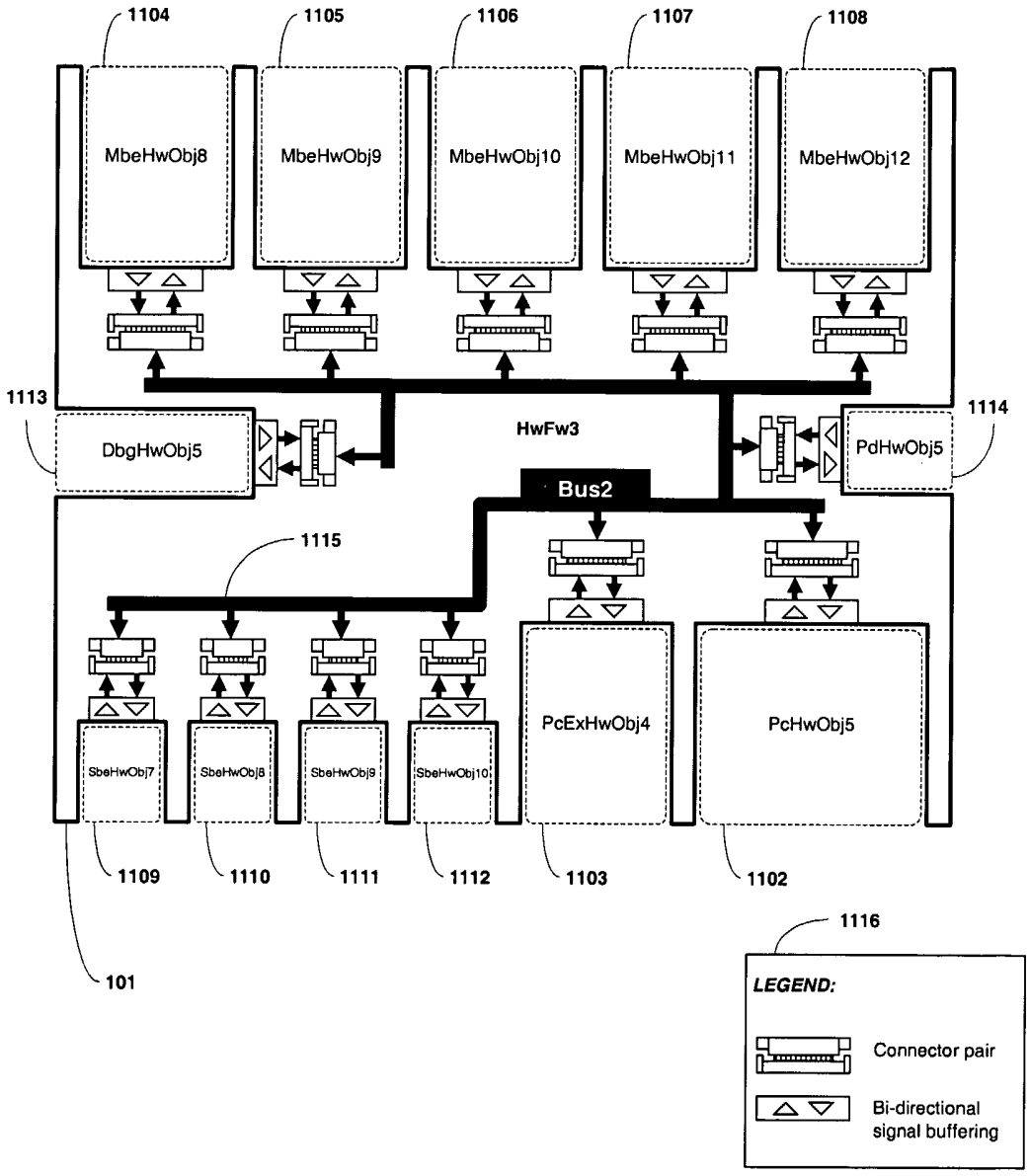
1104
1105
1106
1107
1108
MbeHwObj8
MbeHwObj9
MbeHwObj10
MbeHwObj11
MbeHwObj12
1113
DbgHwObj5
HwFw3
PdHwObj5
1114
Bus2
1115
SbeHwObj7
SbeHwObj8
SbeHwObj9
SbeHwObj10
PcExHwObj4
PcHwObj5
1109
1110
1111
1112
1103
1102
101
1116
LEGEND:
Connector pair
Bi-directional signal buffering

Figure A.11

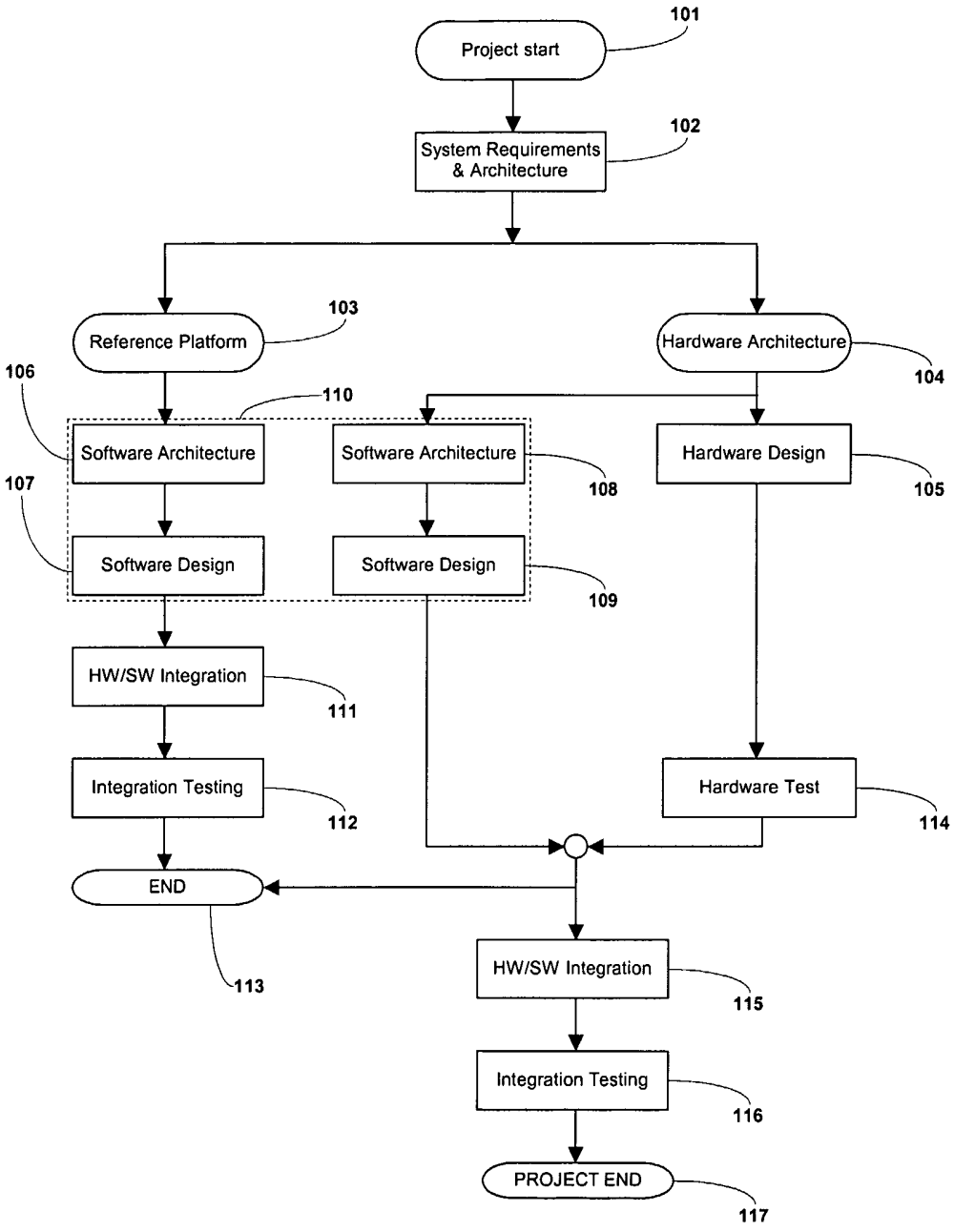
Project start
101
System Requirements & Architecture
102
Reference Platform
103
Hardware Architecture
104
106
110
Software Architecture
Software Architecture
Hardware Design
108
105
107
Software Design
Software Design
109
HW/SW Integration
111
Integration Testing
112
Hardware Test
114
END
113
HW/SW Integration
115
Integration Testing
116
PROJECT END
117

Figure B.1

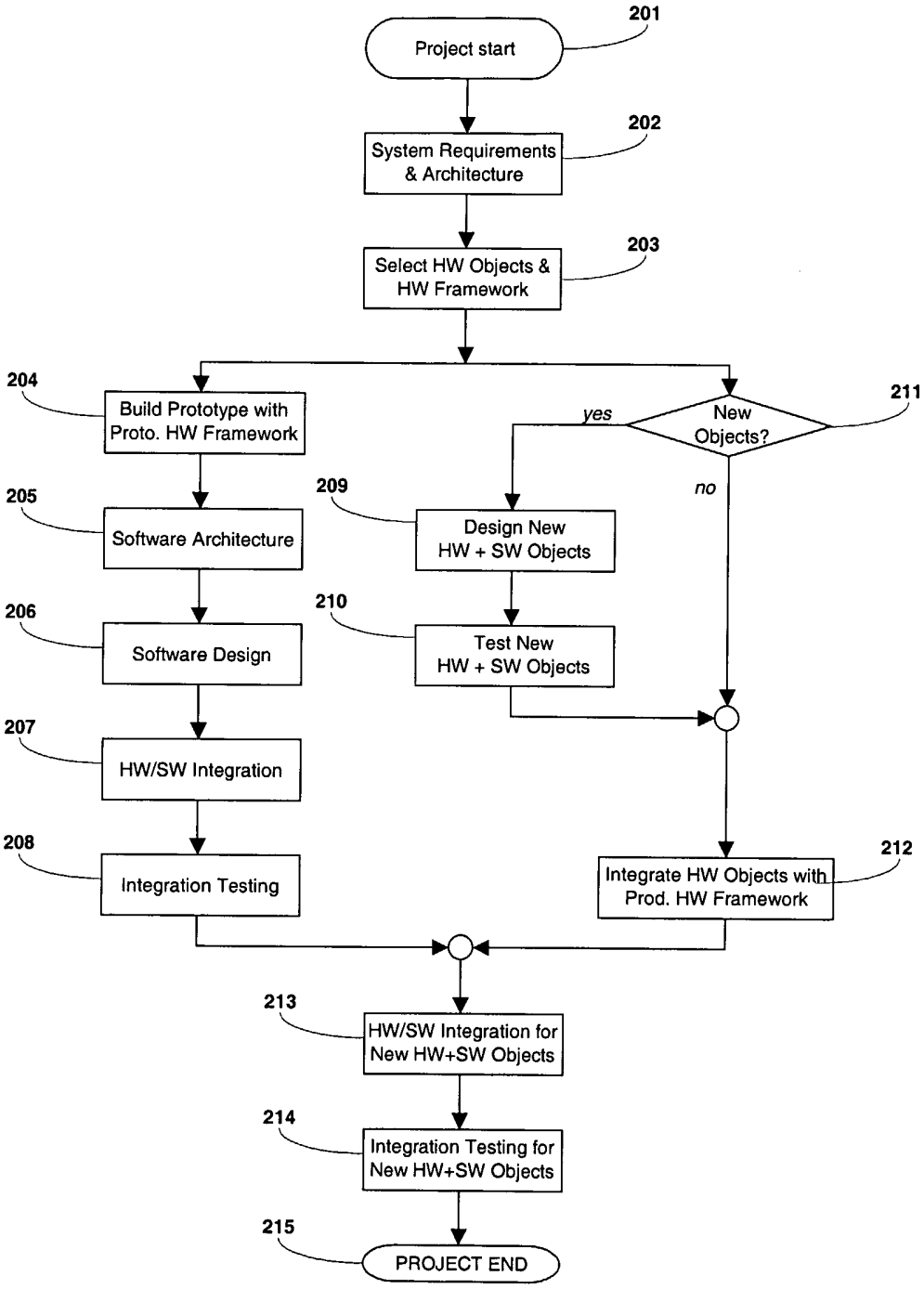
201
Project start
202
System Requirements & Architecture
203
Select HW Objects & HW Framework
204
Build Prototype with Proto. HW Framework
205
Software Architecture
206
Software Design
207
HW/SW Integration
208
Integration Testing
209
Design New HW + SW Objects
210
Test New HW + SW Objects
211
New Objects?
yes
no
212
Integrate HW Objects with Prod. HW Framework
213
HW/SW Integration for New HW+SW Objects
214
Integration Testing for New HW+SW Objects
215
PROJECT END

Figure B.2

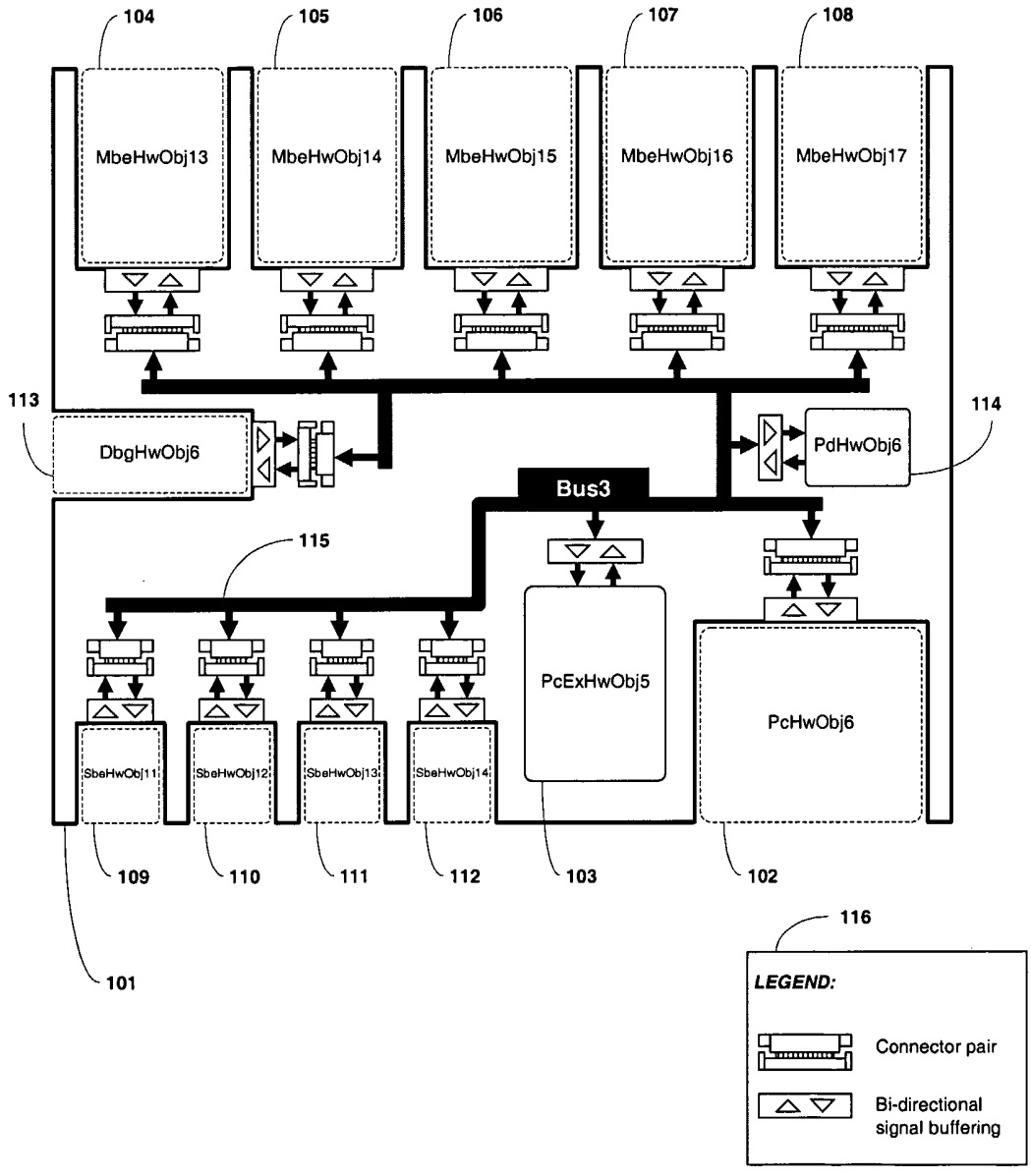
104
105
106
107
108
MbeHwObj13
MbeHwObj14
MbeHwObj15
MbeHwObj16
MbeHwObj17
113
DbgHwObj6
PdHwObj6
114
Bus3
115
PcExHwObj5
PcHwObj6
SbeHwObj11
SbeHwObj12
SbeHwObj13
SbeHwObj14
109
110
111
112
103
102
101
116
LEGEND:
Connector pair
Bi-directional signal buffering

Figure C.1

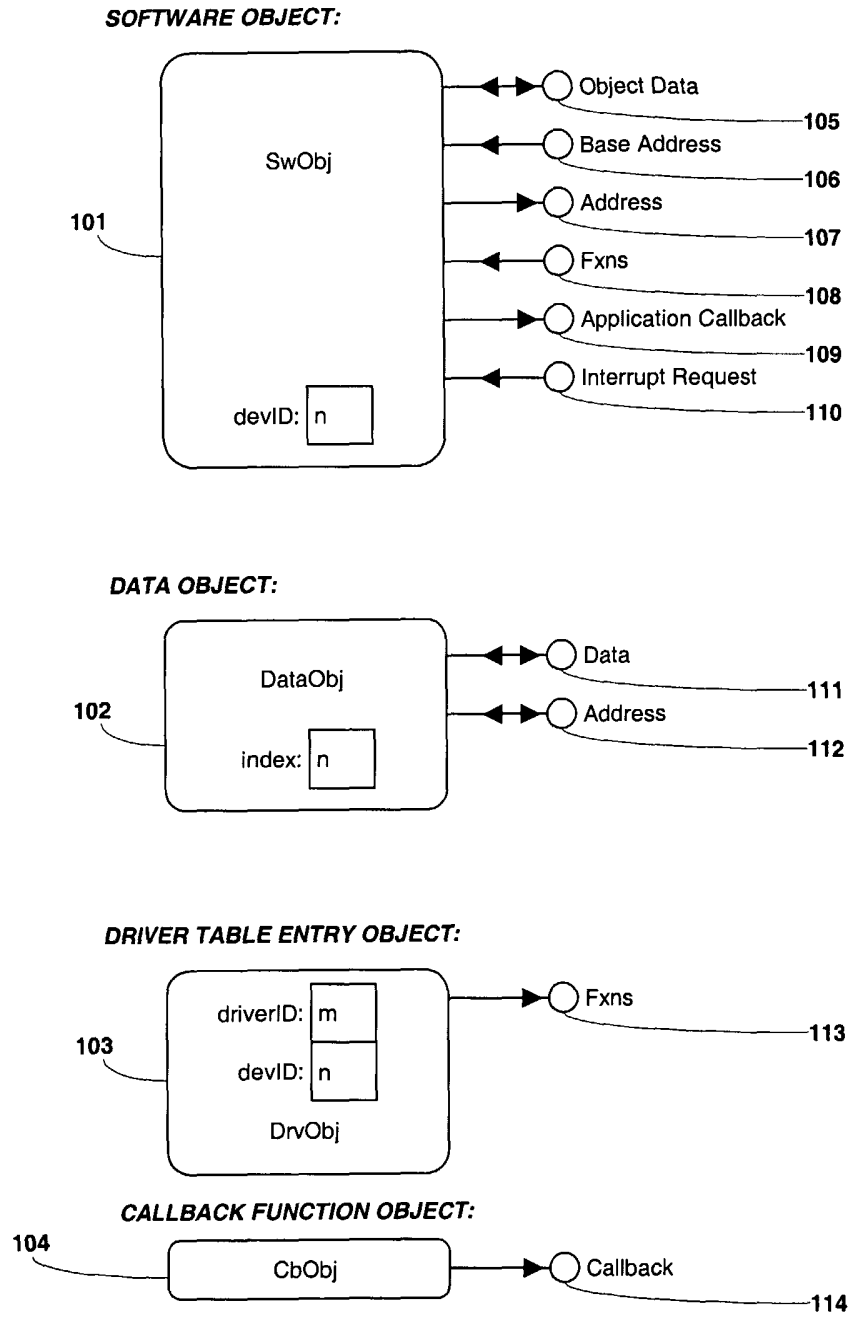
SOFTWARE OBJECT:
SwObj
devID: n
101
Object Data
105
Base Address
106
Address
107
Fxns
108
Application Callback
109
Interrupt Request
110
DATA OBJECT:
DataObj
index: n
102
Data
111
Address
112
DRIVER TABLE ENTRY OBJECT:
driverID: m
devID: n
DrvObj
103
Fxns
113
CALLBACK FUNCTION OBJECT:
CbObj
104
Callback
114

Figure D.1

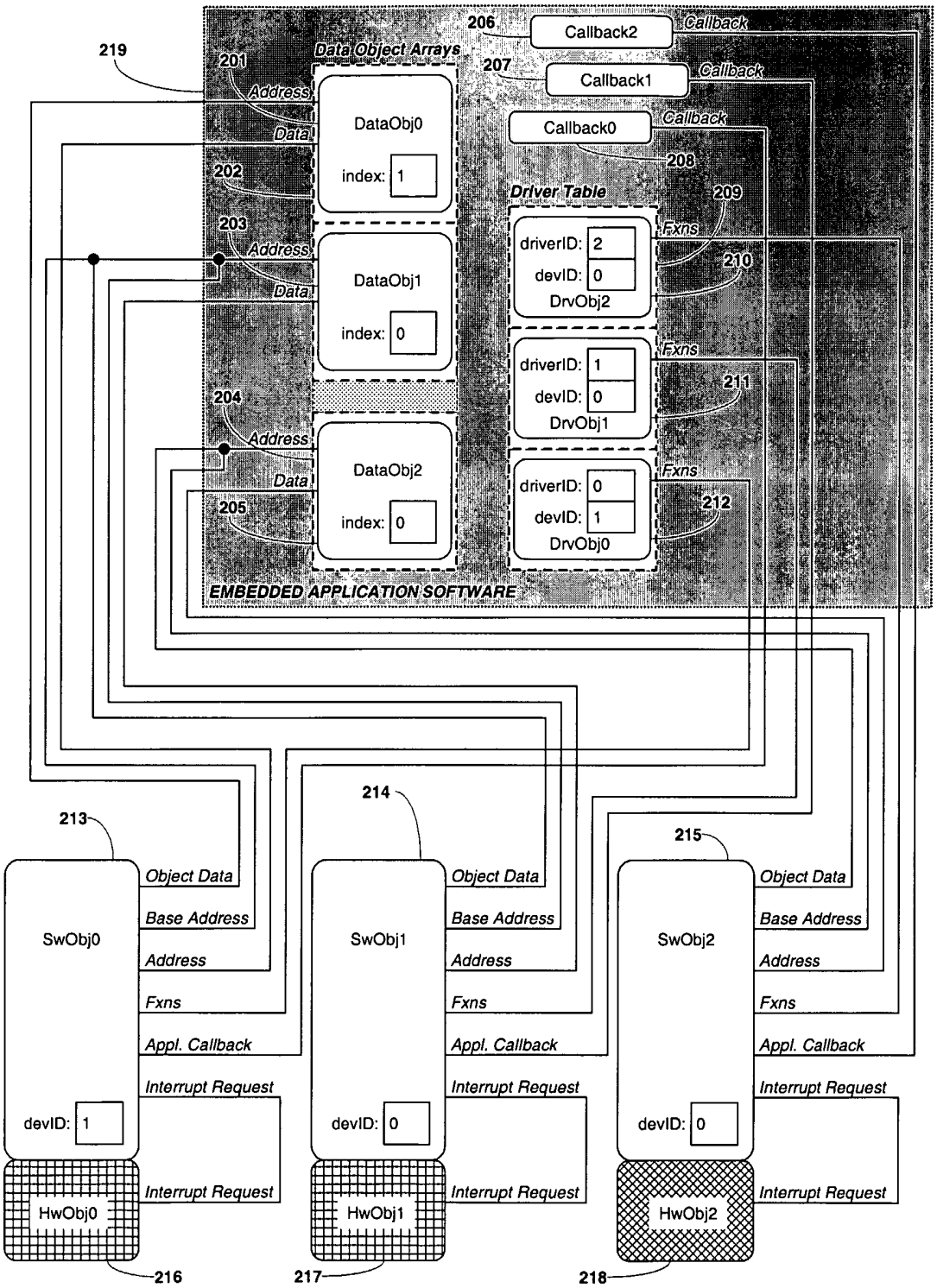
219
Data Object Arrays
201
Address
Data
202
203
Address
Data
204
Address
Data
205
DataObj0
index: 1
DataObj1
index: 0
DataObj2
index: 0
206
Callback2
Callback
207
Callback1
Callback
Callback0
Callback
208
Driver Table
209
Fxns
driverID: 2
devID: 0
DrvObj2
210
Fxns
driverID: 1
devID: 0
DrvObj1
211
Fxns
driverID: 0
devID: 1
DrvObj0
212
EMBEDDED APPLICATION SOFTWARE
213
SwObj0
Object Data
Base Address
Address
Fxns
Appl. Callback
Interrupt Request
devID: 1
Interrupt Request
HwObj0
216
214
SwObj1
Object Data
Base Address
Address
Fxns
Appl. Callback
Interrupt Request
devID: 0
Interrupt Request
HwObj1
217
215
SwObj2
Object Data
Base Address
Address
Fxns
Appl. Callback
Interrupt Request
devID: 0
Interrupt Request
HwObj2
218

Figure D.2

```
/******** IBF3DRIVER type definition  ******************************
 *
 */
typedef unsigned Uns;
typedef int Int;
typedef int Arg;
typedef unsigned short Bool;
typedef void *Ptr;
#define Void void


/******** IBF3DRIVER_MemRec type definition  ******************************
 *
 *   Data type   : struct IBF3DRIVER_MemRec
 *
 *   Description : The IBF3DRIVER_MemRec struct is used to exchange
 *                 memory requirements between applications and drivers
 *                 which implement the IBF3DRIVER interface.
 *
 *   Remarks     :
 */
typedef struct IBF3DRIVER_MemRec {

    Int             size;       /* size in MAU of allocation */
    Int             alignment;  /* alignment requirement (MAU) */
    Void            *base;      /* base address of allocated buf */

} IBF3DRIVER_MemRec;


/******** function prototype definition ******************************
 *
 *   Function type   : IBF3DRIVER_TcallBack
 *
 *   Description     : The IBF3DRIVER_TcallBack is the
 *                     prototype for IBF3DRIVER callback functions.
 *
 *   Remarks         : -
 */
typedef Void (*IBF3DRIVER_TcallBack)(Uns devId, Uns parent, Arg arg);


/******** IBF3DRIVER_Fxns definition ******************************
 *
 *   Data type   : struct IBF3DRIVER_Fxns
 *
 *   Description : The IBF3DRIVER_Fxns struct defines the
 *                 format of the function pointer table
 *                 of the IBF3DRIVER interface.
 *
 *   Remarks     :
 */
typedef struct IBF3DRIVER_Fxns
{
    Int      (*open)        (const Uns devId, Ptr args);
    Void     (*close)       (const Uns devId);
    Void     (*start)       (const Uns devId);
    Void     (*stop)        (const Uns devId);
    Bool     (*putBuf)      (const Uns devId, const Uns chan, Ptr buf, Uns size);
    Ptr      (*getBuf)      (const Uns devId, const Uns chan, Int *size, Int *dataSize);
    Void     (*status)      (const Uns devId, Ptr statusInfo);
    Int      (*ctrl)        (const Uns devId, const Uns command, Arg arg);
    Void     (*setCallback) (const Uns devId, IBF3DRIVER_TcallBack cb, Arg arg);
    Int      (*numAlloc)    (const Uns numDevs, Ptr params);
    Void     (*allocDriver) (const Uns numDevs, IBF3DRIVER_MemRec memTab[], const Ptr params);
    Void     (*allocDevice) (const Uns devId, IBF3DRIVER_MemRec memTab[], const Ptr params);
    Void     (*initDriver)  (const Uns numDevs, const IBF3DRIVER_MemRec memTab[],
                             const Ptr params);
    Void     (*initDevice)  (const Uns devId, const IBF3DRIVER_MemRec memTab[],
                             const Ptr params);

} IBF3DRIVER_Fxns;


/******** IBF3DRIVER_DevDesc definition ******************************
 *
 *   Data type   : struct IBF3DRIVER_DevDesc
 *
 *   Description : The IBF3DRIVER_DevDesc struct defines the
 *                 format of an IBF3DRIVER-compatible device
 *                 driver instance.
 *
 *   Remarks     :
 */
typedef struct IBF3DRIVER_DevDesc{

        Uns                 devId;
        IBF3DRIVER_Fxns*    fxns;

} IBF3DRIVER_DevDesc;
```

Figure D.3

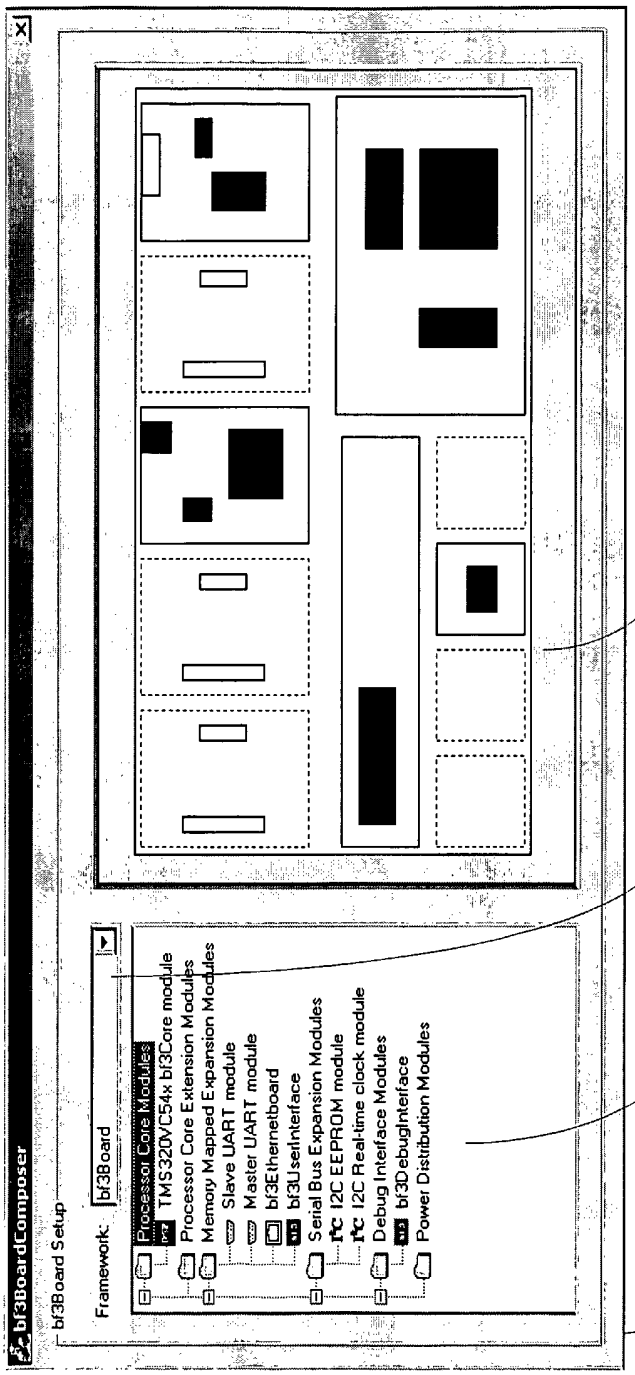
bf3BoardComposer
bf3Board Setup
Framework:
bf3Board
Processor Core Modules
TMS320VC54x bf3Core module
Processor Core Extension Modules
Memory Mapped Expansion Modules
Slave UART module
Master UART module
bf3Ethernetboard
bf3UserInterface
Serial Bus Expansion Modules
I2C EEPROM module
I2C Real-time clock module
Debug Interface Modules
bf3DebugInterface
Power Distribution Modules
101
102
103
104

Figure E.1

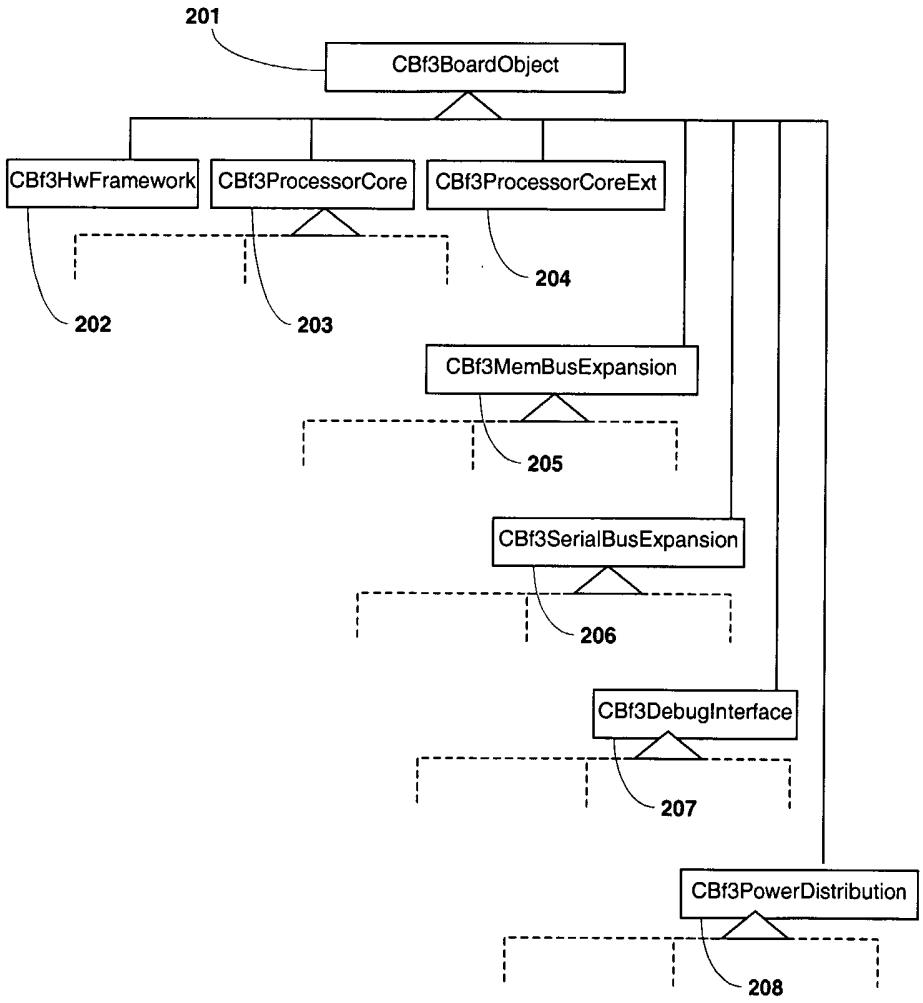
201
CBf3BoardObject
CBf3HwFramework
CBf3ProcessorCore
CBf3ProcessorCoreExt
202
203
204
CBf3MemBusExpansion
205
CBf3SerialBusExpansion
206
CBf3DebugInterface
207
CBf3PowerDistribution
208

Figure E.2